Writing workflows in AiiDA

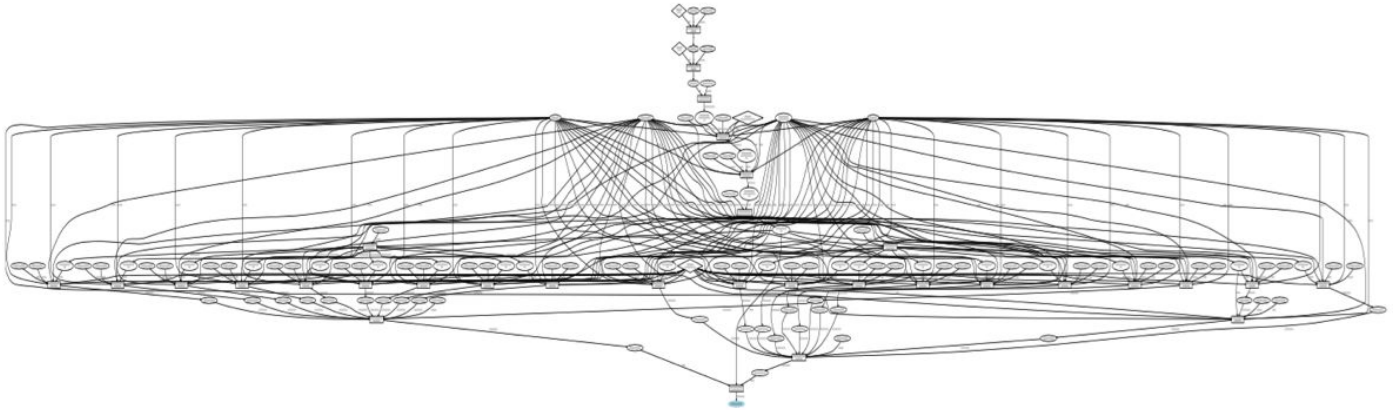
Sebastiaan P. Huber

MARVEL
NATIONAL CENTRE OF COMPETENCE IN RESEARCH
MaX DRIVING THE EXASCALE TRANSITION
swissuniversities
FNSNF SWISS NATIONAL SCIENCE FOUNDATION
INTERSECT
MarketPlace
EPFL

Automated provenance

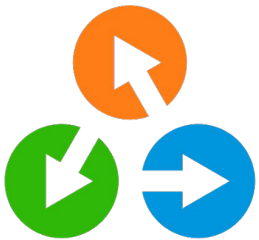
Sebastiaan P. Huber

Keeping data provenance is important…



…but imagine having to manually link everything up

Automated provenance



AiiDA

Automated Interactive Infrastructure and Database for Computational Science

So how exactly is provenance automatically stored?

Automated provenance: calculations

Sebastiaan P. Huber

Imagine the following simple arithmetic problem:

Add two numbers and multiply the sum by third

```python
def add(x, y):
    return x + y

def multiply(x, y):
    return x * y

result = multiply(add(1, 2), 3)
```

Just apply the `calcfunction` decorator

```python
from aiida.engine import calcfunction

@calcfunction
def add(x, y):
    return x + y

@calcfunction
def multiply(x, y):
    return x * y

result = multiply(add(1, 2), 3)
```

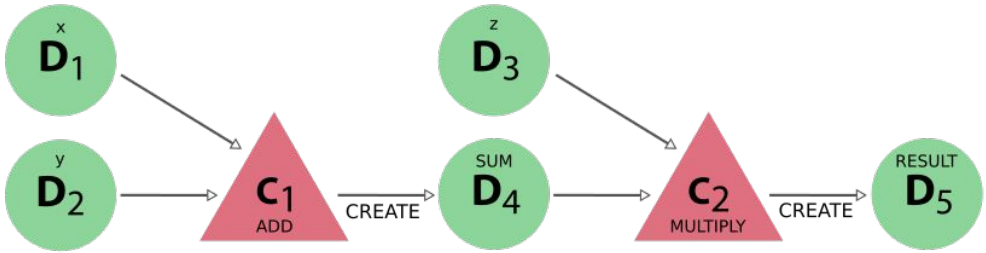
Automated provenance: calculations

Sebastiaan P. Huber

Now we just call it like a normal function while passing storable types

```python
from aiida.engine import calcfunction
from aiida.orm import Int

@calcfunction
def add(x, y):
    return x + y

@calcfunction
def multiply(x, y):
    return x * y

result = multiply(add(Int(1), Int(2)), Int(3))
```

The provenance is automatically stored in the database



Directed Acyclic Graph

- Directed: inputs go in outputs come out
- Acyclic: causality principle forbids an output being its own input

Automated provenance: external codes

But not all code is well-suited as Python code:

What about running external codes through AiiDA?

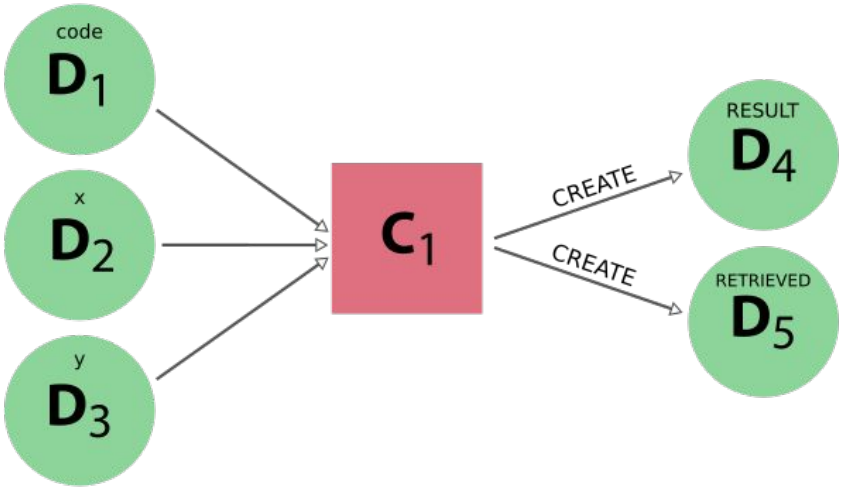
```
from aiida.engine import run
from aiida.orm import load_code, Int
from aiida.plugins import CalculationFactory

ArithmeticAddCalculation = CalculationFactory('arithmetic.add')

inputs = {
    'code': load_code('add@localhost'),
    'x': Int(1),
    'y': Int(2),
}

run(ArithmeticAddCalculation, **inputs)
```

Implementation is different as it requires a *plugin*,
but running very similar...

Automated provenance: external codes

Sebastiaan P. Huber

Generated provenance similar to that of calculation function



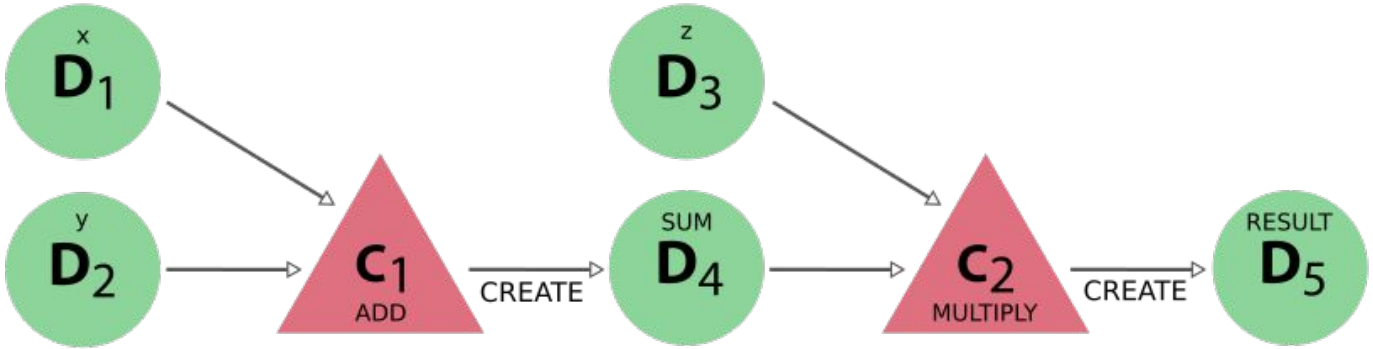
- Has additional output node containing the retrieved output files

- Can be run on remote machines through job scheduler
- Implementation independent of job scheduler
- To change machine, just change the ‘code’ input
- Many can be run in parallel by submitting to the daemon

Automated provenance: workflows

Sebastiaan P. Huber

Let's go back to the first add-multiply example



Individual sequence of calculations recorded...

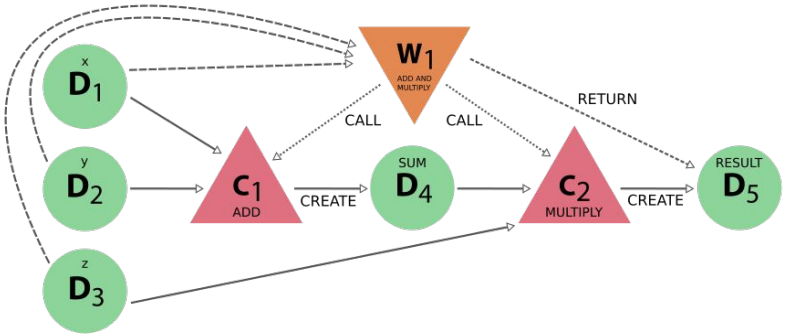
... but not the 'how' or 'why'

Automated provenance: workflows

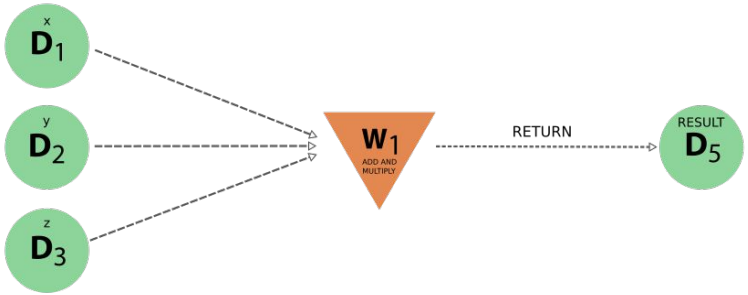
Sebastiaan P. Huber

Work functions can be used to store `logical` provenance

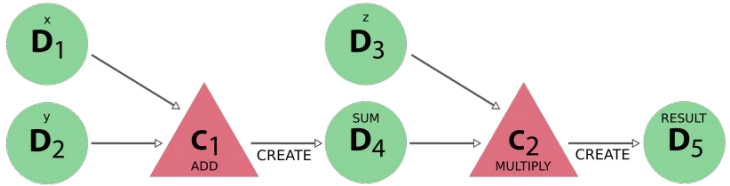
```python
from aiida.engine import calcfunction, workfunction
from aiida.orm import Int

@calcfunction
def add(x, y):
    return Int(x + y)

@calcfunction
def multiply(x, y):
    return Int(x * y)

@workfunction
def add_and_multiply(x, y, z):
    sum = add(x, y)
    product = multiply(sum, z)
    return product

result = add_and_multiply(Int(1), Int(2), Int(3))
```



We can look at the logical provenance to 'hide' the complexity



Or ignore it to retrieve the original data provenance

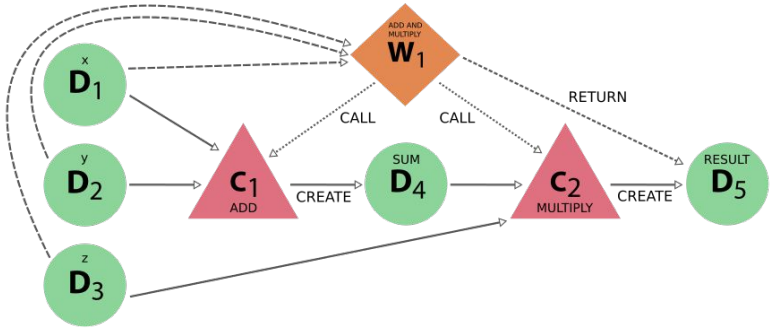


Automated provenance: workflows

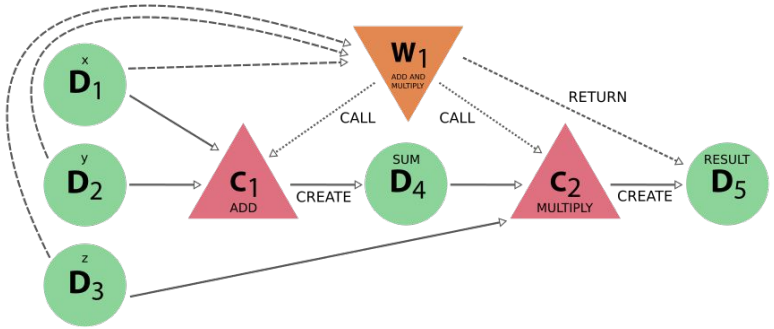
Sebastiaan P. Huber

Work chains achieve the same but save progress in between steps

```python
from aiida.engine import WorkChain, calcfunction
from aiida.orm import Int

@calcfunction
def add(x, y):
    return Int(x + y)

@calcfunction
def multiply(x, y):
    return Int(x * y)

class AddAndMultiplyWorkChain(WorkChain):

    @classmethod
    def define(cls, spec):
        super(AddAndMultiplyWorkChain, cls).define(spec)
        spec.input('x')
        spec.input('y')
        spec.input('z')
        spec.outline(
            cls.add,
            cls.multiply,
            cls.results,
        )
        spec.output('result')

    def add(self):
        self.ctx.sum = add(self.inputs.x, self.inputs.y)

    def multiply(self):
        self.ctx.product = multiply(self.ctx.sum, self.inputs.z)

    def results(self):
        self.out('result', self.ctx.product)
```



Only difference with work function solution is node type



Automated provenance: workflows

Work chains provide many advantages over work functions:

- Many can be run in parallel by submitting to the daemon
- Progress is saved between steps in checkpoints
- Process specification gives succinct but clear summary
- Captures the scientific knowledge and can be re-run
- Can be re-used as building block in more complex workflows

Automated provenance: losing provenance

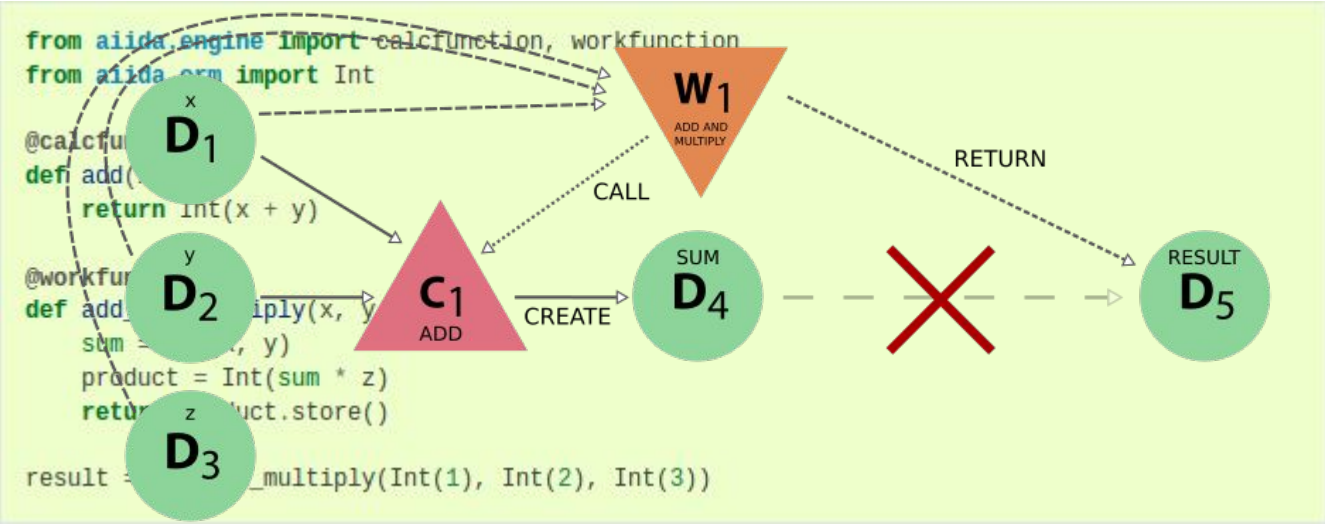
Sebastiaan P. Huber

Workflows cannot *create* new data.

Doing so anyway, will cause loss of provenance

Take first example: add two numbers and multiply with third...

... but now compute the product inside the work function

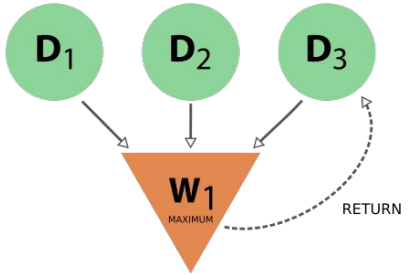


Why a difference between calculations and workflows

Sebastiaan P. Huber

Since workflows can return, they can also return their inputs

```python
from aiida.engine import workfunction
from aiida.orm import Int

@workfunction
def maximum(x, y, z):
    return sorted([x, y, z])[-1]

result = maximum(Int(1), Int(2), Int(3))
```



This breaks the acyclity and therefore the DAG

Two clearly distinct types of processes

CALCULATIONS

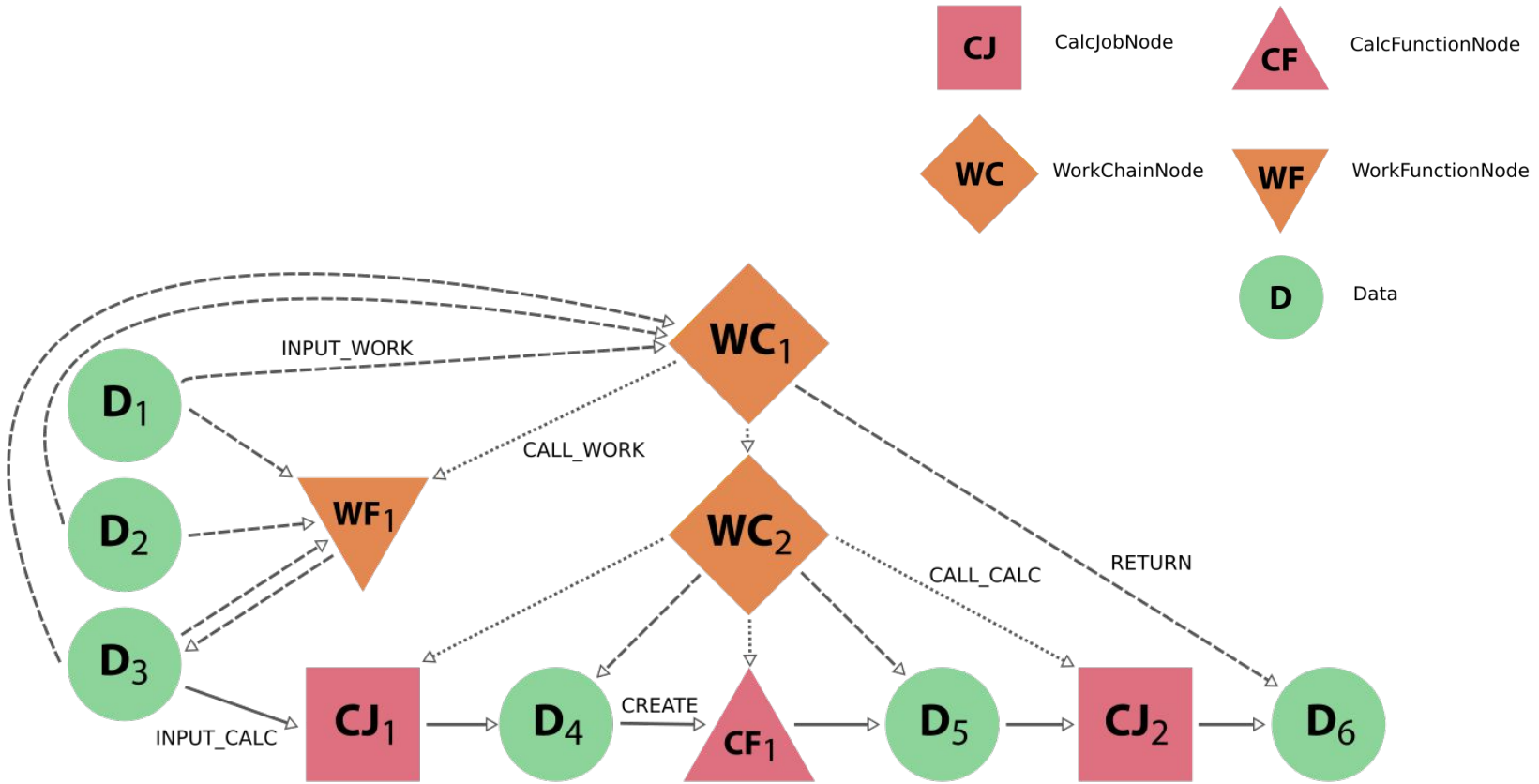
Can *create* new data

WORKFLOWS

Can *call* other processes
Can *return* existing data

Provenance design: an overview

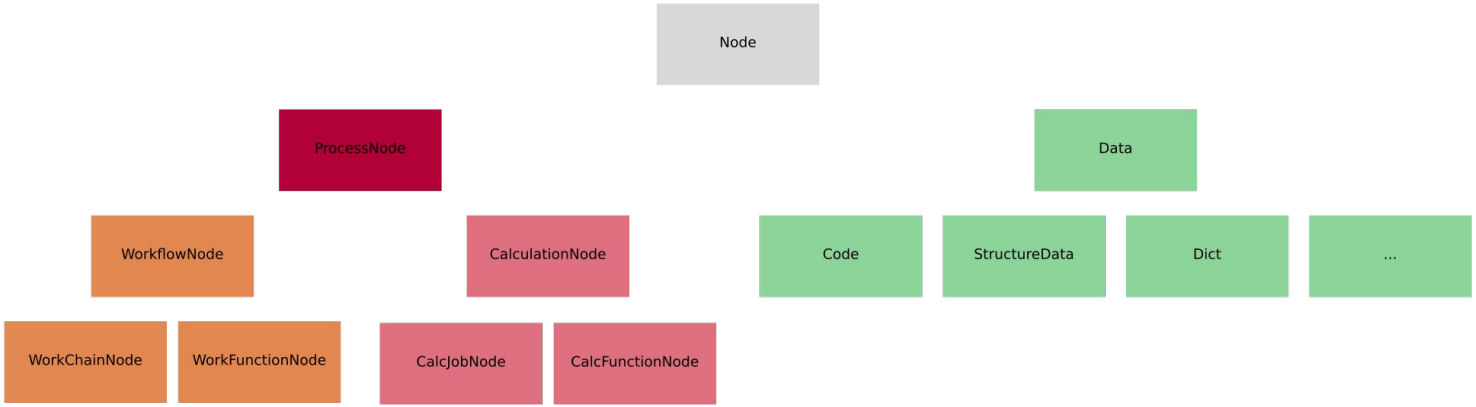
Sebastiaan P. Huber



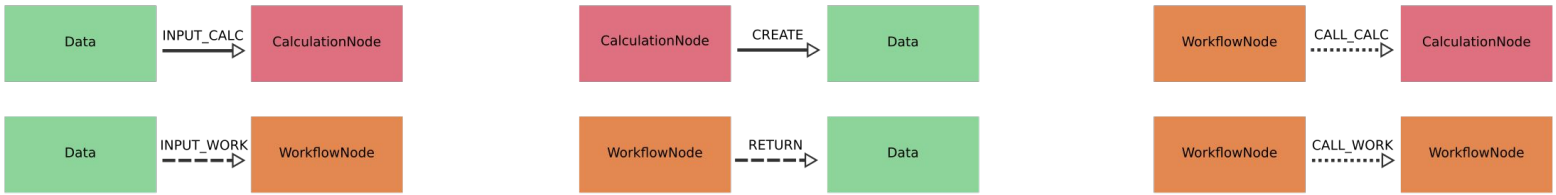
Provenance design: an overview

Sebastiaan P. Huber

ORM Hierarchy



Link Hierarchy



Processes and nodes

Sebastiaan P. Huber

The engine runs *processes* and stores a corresponding *node* in the database

Process class	Node class	Used for
`CalcJob`	`CalcJobNode`	Calculations performed by external codes
`WorkChain`	`WorkChainNode`	Workflows that run multiple sub processes
`FunctionProcess`	`CalcFunctionNode`	Python functions decorated with the `calcfunction` decorator
`FunctionProcess`	`WorkFunctionNode`	Python functions decorated with the `workfunction` decorator

Process state

Active	Terminated
Created	Killed
Running	Excepted
Waiting	Finished

Process node attributes

Property	Meaning
`process_state`	Returns the current process state
`exit_status`	Returns the exit status, or None if not set
`exit_message`	Returns the exit message, or None if not set
`is_terminated`	Returns `True` if the process was either `Killed`, `Excepted` or `Finished`
`is_killed`	Returns `True` if the process is `Killed`
`is_excepted`	Returns `True` if the process is `Excepted`
`is_finished`	Returns `True` if the process is `Finished`
`is_finished_ok`	Returns `True` if the process is `Finished` and the `exit_status` is equal to zero
`is_failed`	Returns `True` if the process is `Finished` and the `exit_status` is non-zero

Launching processes

Sebastiaan P. Huber

Four processes launchers with identical interface

`run`	Run blockingly and return result
`run_get_node`	Run blockingly and return result + node
`run_get_pk`	Run blockingly and return result + pk
`submit`	Submit to daemon and return node

Running will block the interpreter, *submitting* will not as the daemon will take care of it

Launching processes

Sebastiaan P. Huber

Four processes launchers with identical interface

`run`	Run blockingly and return result
`run_get_node`	Run blockingly and return result + node
`run_get_pk`	Run blockingly and return result + pk
`submit`	Submit to daemon and return node

Run blockingly in the current interpreter

```
from aiida import orm
from aiida.engine import run

ArithmeticAddCalculation = CalculationFactory('arithmetic.add')
result = run(ArithmeticAddCalculation, x=orm.Int(1), y=orm.Int(2))
```

Launching processes

Sebastiaan P. Huber

Four processes launchers with identical interface

`run`	Run blockingly and return result
`run_get_node`	Run blockingly and return result + node
`run_get_pk`	Run blockingly and return result + pk
`submit`	Submit to daemon and return node

Run variants to get the process node or pk in addition to the result

```
from aiida import orm
from aiida.engine import run_get_node, run_get_pk

ArithmeticAddCalculation = CalculationFactory('arithmetic.add')
result, node = run_get_node(ArithmeticAddCalculation, x=orm.Int(1), y=orm.Int(2))
result, pk = run_get_pk(ArithmeticAddCalculation, x=orm.Int(1), y=orm.Int(2))
```

Launching processes

Sebastiaan P. Huber

Four processes launchers with identical interface

`run`	Run blockingly and return result
`run_get_node`	Run blockingly and return result + node
`run_get_pk`	Run blockingly and return result + pk
`submit`	Submit to daemon and return node

Variants are available as attributes on *run* launcher requiring only single import

```
from aiida import orm
from aiida.engine import run

ArithmeticAddCalculation = CalculationFactory('arithmetic.add')
result = run(ArithmeticAddCalculation, x=orm.Int(1), y=orm.Int(2))
result, node = run.get_node(ArithmeticAddCalculation, x=orm.Int(1), y=orm.Int(2))
result, pk = run.get_pk(ArithmeticAddCalculation, x=orm.Int(1), y=orm.Int(2))
```

Launching processes

Sebastiaan P. Huber

Four processes launchers with identical interface

`run`	Run blockingly and return result
`run_get_node`	Run blockingly and return result + node
`run_get_pk`	Run blockingly and return result + pk
`submit`	Submit to daemon and return node

Submit to the daemon to immediately regain control of the interpreter

```
from aiida import orm
from aiida.engine import submit

ArithmeticAddCalculation = CalculationFactory('arithmetic.add')
node = submit(ArithmeticAddCalculation, x=orm.Int(1), y=orm.Int(2))
```

Launching processes

Four processes launchers with identical interface

`run`	Run blockingly and return result
`run_get_node`	Run blockingly and return result + node
`run_get_pk`	Run blockingly and return result + pk
`submit`	Submit to daemon and return node

Can use dictionary with keyword expansion in case of many inputs

```
from aiida import orm
from aiida.engine import submit

ArithmeticAddCalculation = CalculationFactory('arithmetic.add')
inputs = {
    'x': orm.Int(1),
    'y': orm.Int(2)
}
node = submit(ArithmeticAddCalculation, **inputs)
```

Command line interaction: `verdi process`

Your one-stop-shop for inspecting and interacting with processes

```
(aiida_dev) sphuber@theos:~$ verdi process --help
Usage: verdi process [OPTIONS] COMMAND [ARGS]...

  Inspect and manage processes.

Options:
  -h, --help  Show this message and exit.

Commands:
  kill    Kill running processes.
  list    Show a list of processes that are still running.
  pause   Pause running processes.
  play    Play paused processes.
  report  Show the log report for one or multiple processes.
  show    Show a summary for one or multiple processes.
  status  Print the status of the process.
  watch   Watch the state transitions for a process.
```

Sebastiaan P. Huber

Command line interaction: `verdi process`

Sebastiaan P. Huber

Your one-stop-shop for inspecting and interacting with processes

`verdi process list`: list active and terminated processes

```
(aiida_3dd) aiida@theossrv5:~/code/aiida/env/3dd$ verdi process list -l 10
     PK  Created    State      Process label     Process status
-------  ---------  ---------  ----------------  ----------------------------------
1285287  2D ago     ▶ Waiting  PwRelaxWorkChain
1285535  2D ago     ▶ Waiting  PwRelaxWorkChain
1286517  2D ago     ▶ Waiting  PwRelaxWorkChain
1286913  2D ago     ▶ Waiting  PwRelaxWorkChain
1287288  2D ago     ▶ Waiting  PwBaseWorkChain
1287486  2D ago     ▶ Waiting  PwRelaxWorkChain
1287517  2D ago     ▶ Waiting  PwBaseWorkChain
1287937  2D ago     ▶ Waiting  PwRelaxWorkChain
1289927  2D ago     ▶ Waiting  PwCalculation    Waiting for transport task: update
1291148  2D ago     ▶ Waiting  PwBaseWorkChain

Total results: 10

Info: last time an entry changed state: 43s ago (at 11:09:50 on 2019-03-22)
```

Command line interaction: `verdi process`

Your one-stop-shop for inspecting and interacting with processes

`verdi process status`: tree representation of call stack

```
(aiida_3dd) aiida@theossrv5:~/code/aiida/env/3dd$ verdi process status 1338418
PwRelaxWorkChain <pk=1338418> [ProcessState.FINISHED] [3:results]
    ├─ PwBaseWorkChain <pk=1338525> [ProcessState.FINISHED] [4:results]
    │   ├─ create_kpoints_from_distance <pk=1338529> [ProcessState.FINISHED]
    │   └─ CalcJobNode <pk=1338569> [FINISHED]
    ├─ PwBaseWorkChain <pk=1341443> [ProcessState.FINISHED] [4:results]
    │   ├─ create_kpoints_from_distance <pk=1341445> [ProcessState.FINISHED]
    │   └─ CalcJobNode <pk=1341463> [FINISHED]
    └─ PwBaseWorkChain <pk=1341701> [ProcessState.FINISHED] [4:results]
        ├─ create_kpoints_from_distance <pk=1341703> [ProcessState.FINISHED]
        └─ CalcJobNode <pk=1341730> [FINISHED]
```


Sebastiaan P. Huber

Command line interaction: `verdi process`

Sebastiaan P. Huber

Your one-stop-shop for inspecting and interacting with processes

`verdi process report`: complete report of log messages and scheduler stdout/stderr

```
(aiida_3dd) aiida@theossrv5:~/code/aiida/env/3dd$ verdi process report 1338418
2019-03-22 05:26:39 [553130 | REPORT]: [1338418|PwRelaxWorkChain|run_relax]: launching PwBaseWorkChain<1338525>
2019-03-22 05:28:02 [553147 | REPORT]:   [1338525|PwBaseWorkChain|run_calculation]: launching PwCalculation<1338569> iteration #1
2019-03-22 07:45:52 [554706 | REPORT]:   [1338525|PwBaseWorkChain|inspect_calculation]: PwCalculation<1338569> completed successfully
2019-03-22 07:45:52 [554707 | REPORT]:   [1338525|PwBaseWorkChain|results]: workchain completed after 1 iterations
2019-03-22 07:45:52 [554708 | REPORT]:   [1338525|PwBaseWorkChain|on_terminated]: remote folders will not be cleaned
2019-03-22 07:45:53 [554709 | REPORT]: [1338418|PwRelaxWorkChain|inspect_relax]: after iteration 1 cell volume of relaxed structure is 427.27585978
2019-03-22 07:45:54 [554710 | REPORT]: [1338418|PwRelaxWorkChain|run_relax]: launching PwBaseWorkChain<1341443>
2019-03-22 07:47:21 [554713 | REPORT]:   [1341443|PwBaseWorkChain|run_calculation]: launching PwCalculation<1341463> iteration #1
2019-03-22 08:00:47 [554854 | REPORT]:   [1341443|PwBaseWorkChain|inspect_calculation]: PwCalculation<1341463> completed successfully
2019-03-22 08:00:47 [554855 | REPORT]:   [1341443|PwBaseWorkChain|results]: workchain completed after 1 iterations
2019-03-22 08:00:47 [554856 | REPORT]:   [1341443|PwBaseWorkChain|on_terminated]: remote folders will not be cleaned
2019-03-22 08:00:48 [554857 | REPORT]: [1338418|PwRelaxWorkChain|inspect_relax]: after iteration 2 cell volume of relaxed structure is 427.275859777
2019-03-22 08:00:48 [554858 | REPORT]: [1338418|PwRelaxWorkChain|inspect_relax]: relative cell volume difference 5.54830030607e-12 smaller than convergence threshold 0.01
2019-03-22 08:00:49 [554859 | REPORT]: [1338418|PwRelaxWorkChain|run_final_scf]: launching PwBaseWorkChain<1341701> for final scf
2019-03-22 08:02:13 [554881 | REPORT]:   [1341701|PwBaseWorkChain|run_calculation]: launching PwCalculation<1341730> iteration #1
2019-03-22 08:11:19 [554931 | REPORT]:   [1341701|PwBaseWorkChain|inspect_calculation]: PwCalculation<1341730> completed successfully
2019-03-22 08:11:19 [554932 | REPORT]:   [1341701|PwBaseWorkChain|results]: workchain completed after 1 iterations
2019-03-22 08:11:19 [554933 | REPORT]:   [1341701|PwBaseWorkChain|on_terminated]: remote folders will not be cleaned
2019-03-22 08:11:20 [554934 | REPORT]: [1338418|PwRelaxWorkChain|results]: workchain completed after 2 iterations
2019-03-22 08:11:30 [554935 | REPORT]: [1338418|PwRelaxWorkChain|on_terminated]: cleaned remote folders of calculations: 1341730 1341463 1338569
```

Command line interaction: `verdi process`

Sebastiaan P. Huber

Your one-stop-shop for inspecting and interacting with processes

`verdi process pause`: pause an active process

```
(aiida_dev) sphuber@theos:~$ verdi process pause 804
Success: scheduled pause Process<804>
```

`verdi process play`: resume a paused process

```
(aiida_dev) sphuber@theos:~$ verdi process play 812
Success: scheduled play Process<812>
```

`verdi process kill`: kill an active process

```
(aiida_dev) sphuber@theos:~$ verdi process kill 820
Success: scheduled kill Process<820>
```

`verdi process pause/play/kill`: fails if process is already terminated

```
(aiida_dev) sphuber@theos:~$ verdi process kill 812
Error: Process<812> is already terminated
```

Developers

The Materials Cloud And AiiDA teams


Carl Simon Adorf (EPFL)


Casper W. Andersen (EPFL)


Marnik Bercx (EPFL)


Marco Borelli (EPFL)


Valeria Granata (EPFL)


Sebastiaan P. Huber (EPFL)


Leonid Kahle (EPFL)


Snehal P. Kumbhar (EPFL)


Elsa Passaro (EPFL)


Francisco F. Ramirez (EPFL)


Leopold Talirz (EPFL)


Aliaksandr Yakutovich (EPFL)


Chris Sewell (EPFL)


Giovanni Pizzi (EPFL)


Berend Smit (EPFL)


Joost VandeVondele (ETHZ,CSCS)


Thomas Schulthess (ETHZ,CSCS)


Nicola Marzari (EPFL)

Contributors for the 40+ plugins: Quantum ESPRESSO, Wannier90, CP2K, FLEUR, YAMBO, SIESTA, VASP, CASTEP, CRYSTAL, …

Contributors to aiida-core and former AiiDA team members —
Oscar Arbelaez, Michael Atambo, Valentin Bersier, Marco Borelli, Jocelyn Boullier, Jens Bröder, Ivano E. Castelli, Andrea Cepellotti, Keija Cui, Vladimir Dikan, Marco Dorigo, Y.-W. Fang, Fernando Gargiulo, Marco Gibertini, Davide Grassano, Dominik Gresch, Conrad Johnston, Rico Häuselmann, Daniel Hollas, Eric Hontz, Jianxing Huang, Christoph Koch, Espen Flage-Larsen, Ian Lee, Daniel Marchand, Antimo Marrazzo, Andrius Merkys, Simon Pintarelli, Nicolas Mounet, Tiziano Müller, Gianluca Prandini, Philip Rüßmann, Riccardo Sabatini, Ole Schütt, Phillippe Schwaller, Andreas Stamminger, Atsushi Togo, Daniele Tomerini, Nicola Varini, Martin Uhrin, Jason Yu, Austin Zadoks, Bonan Zhu, Mario Zic, Spyros Zoupanos