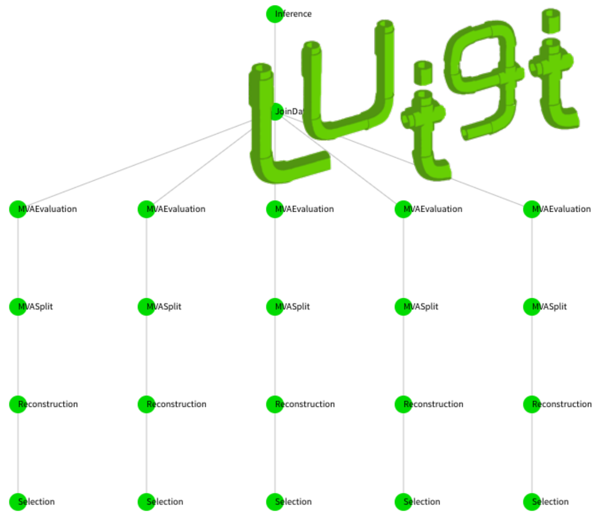
law

luigi analysis workflow



Luigi Analysis Workflows

Design Pattern for Full Analysis Automation on Local and Distributed Resources

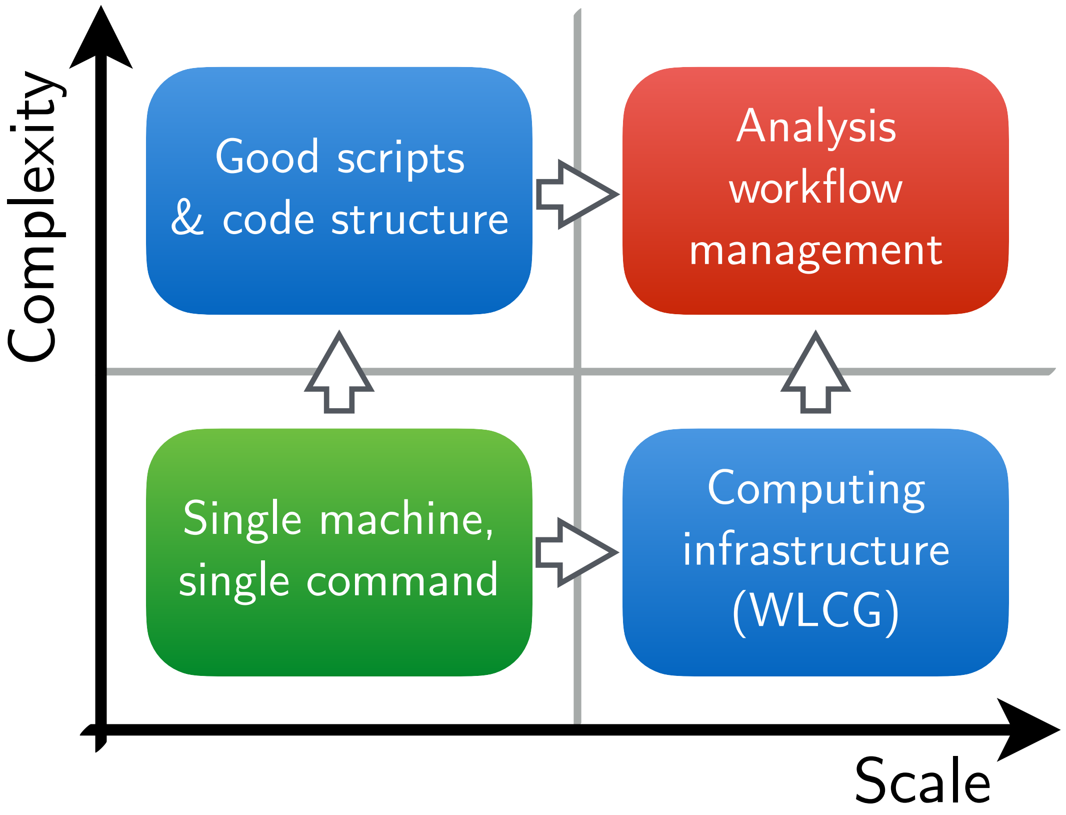
Marcel Rieger (UHH)

HSF Meeting - Workflow Management Tools

08.12.2021

UHH
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Landscape of HEP analyses


Complexity
Good scripts & code structure
Analysis workflow management
Single machine, single command
Computing infrastructure (WLCG)
Scale

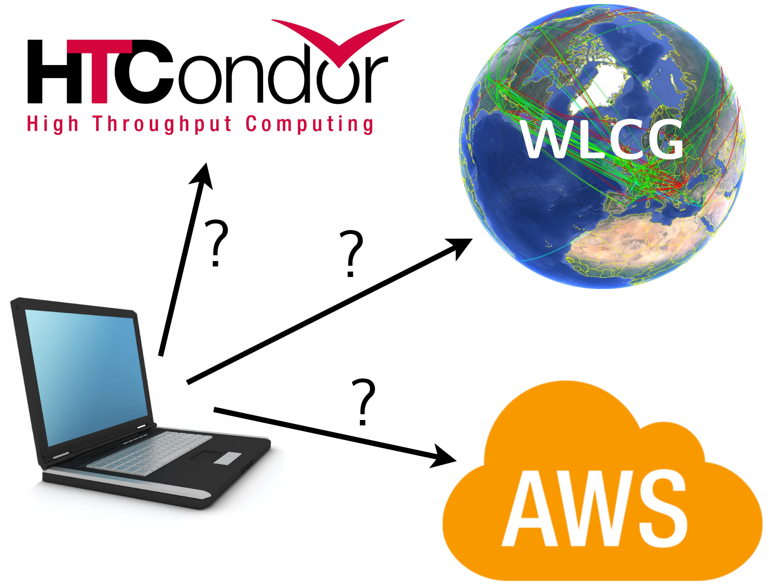
- Most analyses are both **large and complex**
  - Structure & requirements between workloads mostly undocumented
  - Manual execution & steering of jobs, bookkeeping of data across SEs, data revisions, ...

  → Error-prone & time-consuming

- From personal experience: ⅔ of time required for technicalities, ⅓ for physics

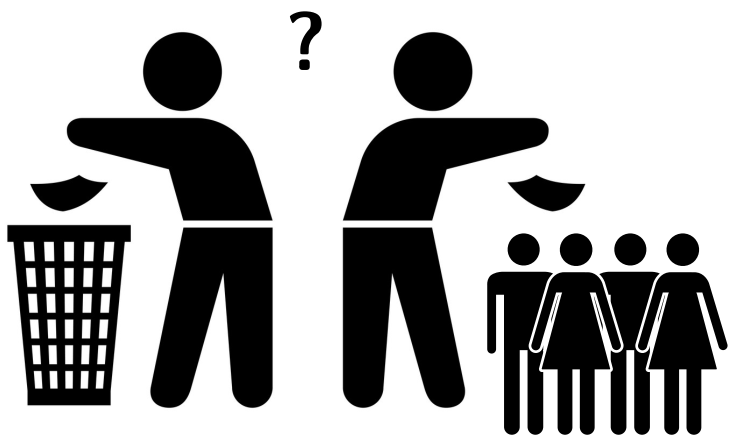
  → Physics output doubled if it was the other way round?

Motivational questions



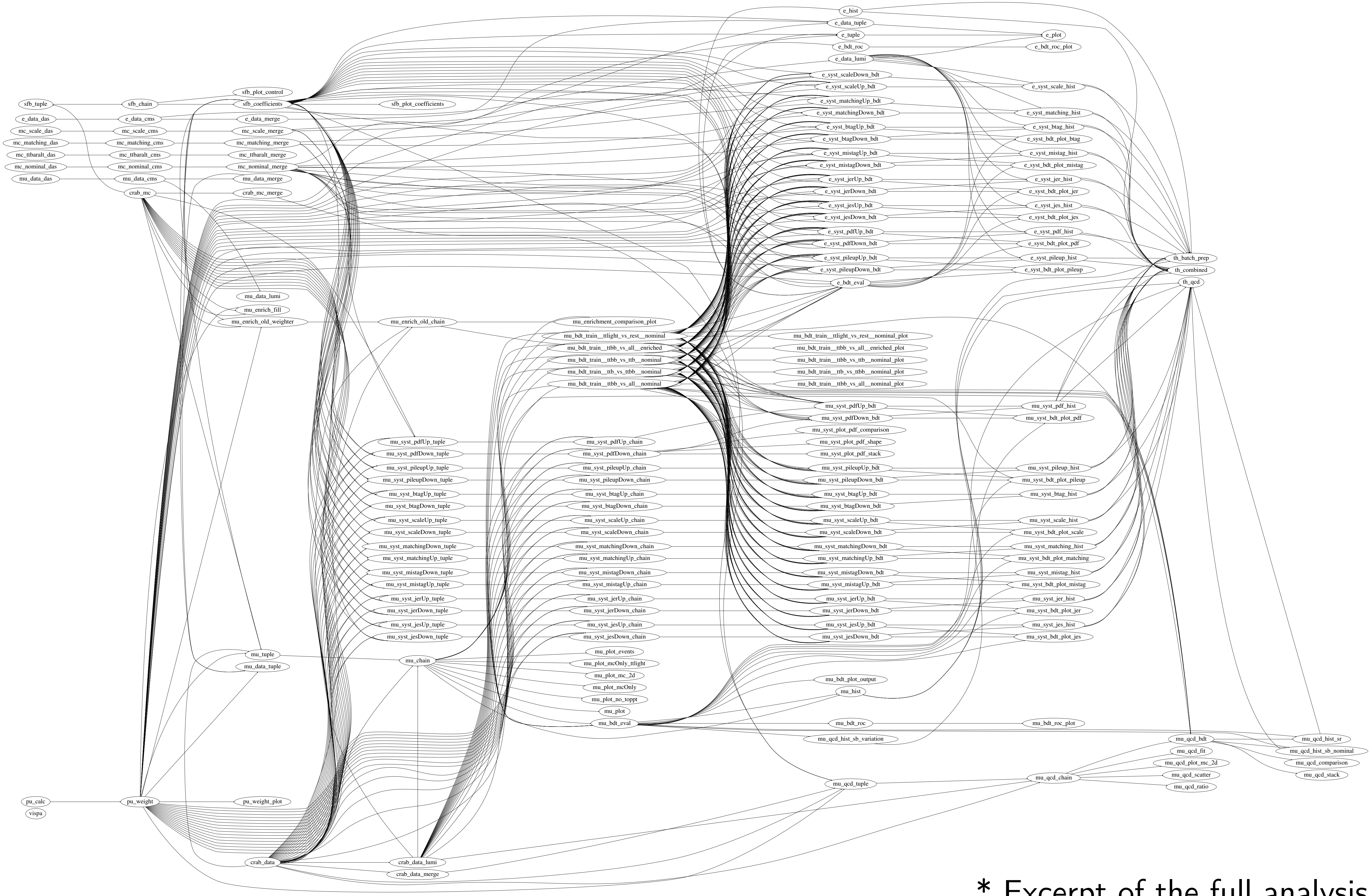
- **Portability**: Does the analysis depend on ...
  - where it runs?
  - where it stores data?
    - ▻ Execution/storage should **not** dictate code design!

- **Reproducibility**: When a postdoc / PhD student leaves, ...
  - can someone else run the analysis?
  - is there a loss of information? Is a new *framework* required?
    - ▻ Dependencies often **only** exist in the physicists head!



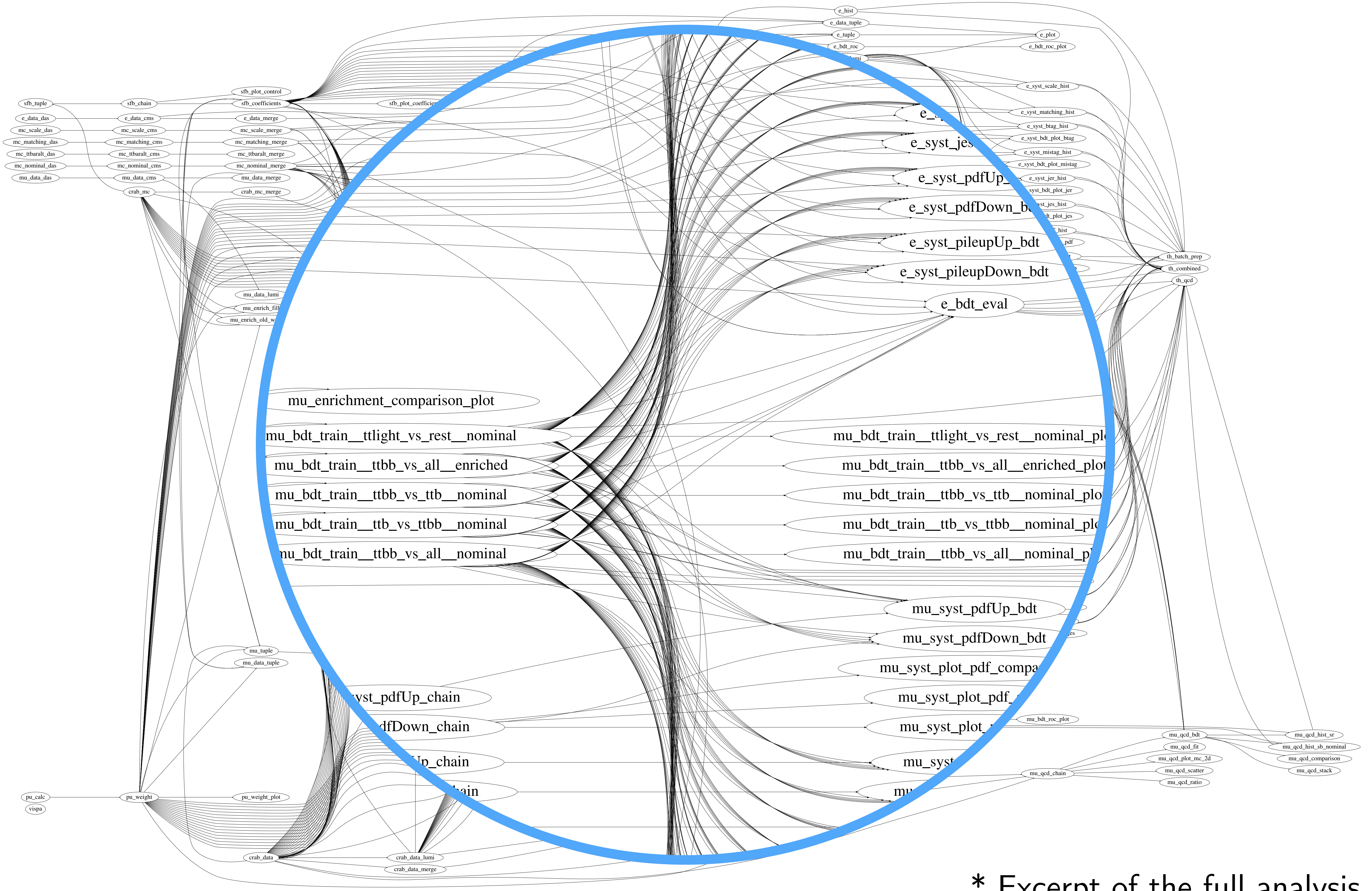
- **Preservation**: After an analysis is published ...
  - are people investing time to preserve their work?
  - can it be repeated after O(years)?
    - ▻ Daily working environment should provide preservation features **out-of-the-box**!

Example: ttbb cross section measurement



* Excerpt of the full analysis

Example: ttbb cross section measurement



* Excerpt of the full analysis

Luigi

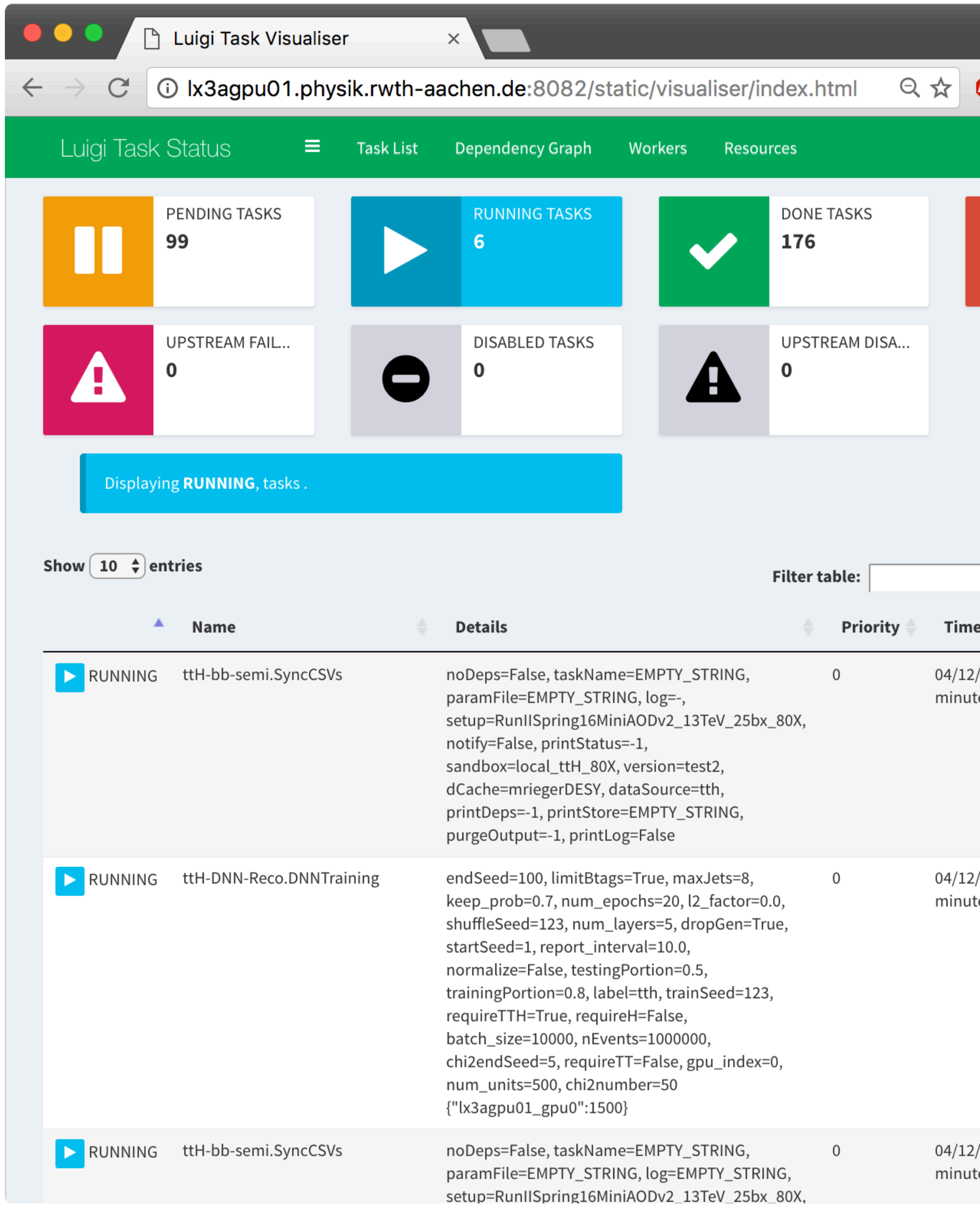
- Python package for building complex pipelines
- Development started at Spotify, now open-source and community-driven

github.com/spotify/luigi

Watch 493 | Unstar 15.2k | Fork 2.3k

<u>Building blocks</u>

1. Workloads defined as **Task** classes that can **require** other **Tasks**
2. Tasks produce output **Targets**
3. **Parameters** customize tasks & control runtime behavior



PENDING TASKS 99
RUNNING TASKS 6
DONE TASKS 176
UPSTREAM FAIL... 0
DISABLED TASKS 0
UPSTREAM DISA... 0

Displaying **RUNNING**, tasks .

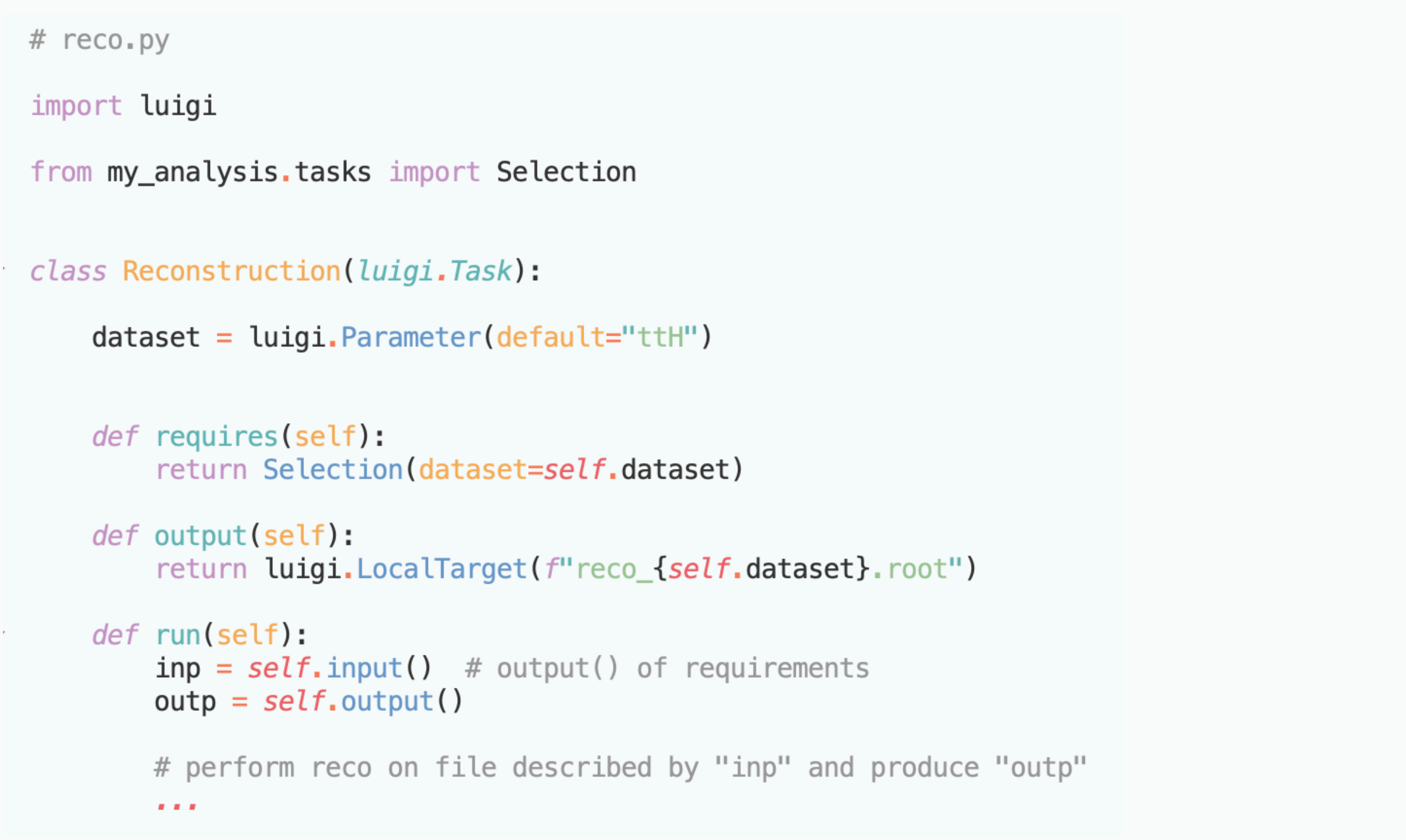
Show 10 entries

Filter table:

	Name	Details	Priority	Time
RUNNING	ttH-bb-semi.SyncCSVs	noDeps=False, taskName=EMPTY_STRING, paramFile=EMPTY_STRING, log=-, setup=RunIISpring16MiniAODv2_13TeV_25bx_80X, notify=False, printStatus=-1, sandbox=local_ttH_80X, version=test2, dCache=mriegerDESY, dataSource=tth, printDeps=-1, printStore=EMPTY_STRING, purgeOutput=-1, printLog=False	0	04/12/ minut
RUNNING	ttH-DNN-Reco.DNNTraining	endSeed=100, limitBtags=True, maxJets=8, keep_prob=0.7, num_epochs=20, l2_factor=0.0, shuffleSeed=123, num_layers=5, dropGen=True, startSeed=1, report_interval=10.0, normalize=False, testingPortion=0.5, trainingPortion=0.8, label=tth, trainSeed=123, requireTTH=True, requireH=False, batch_size=10000, nEvents=1000000, chi2endSeed=5, requireTT=False, gpu_index=0, num_units=500, chi2number=50 {"lx3agpu01_gpu0":1500}	0	04/12/ minut
RUNNING	ttH-bb-semi.SyncCSVs	noDeps=False, taskName=EMPTY_STRING, paramFile=EMPTY_STRING, log=EMPTY_STRING, setup=RunIISpring16MiniAODv2_13TeV_25bx_80X,	0	04/12/ minut

- Web UI with two-way messaging (task → UI, UI → task), automatic error handling, task history browser, collaborative features, command line interface, …

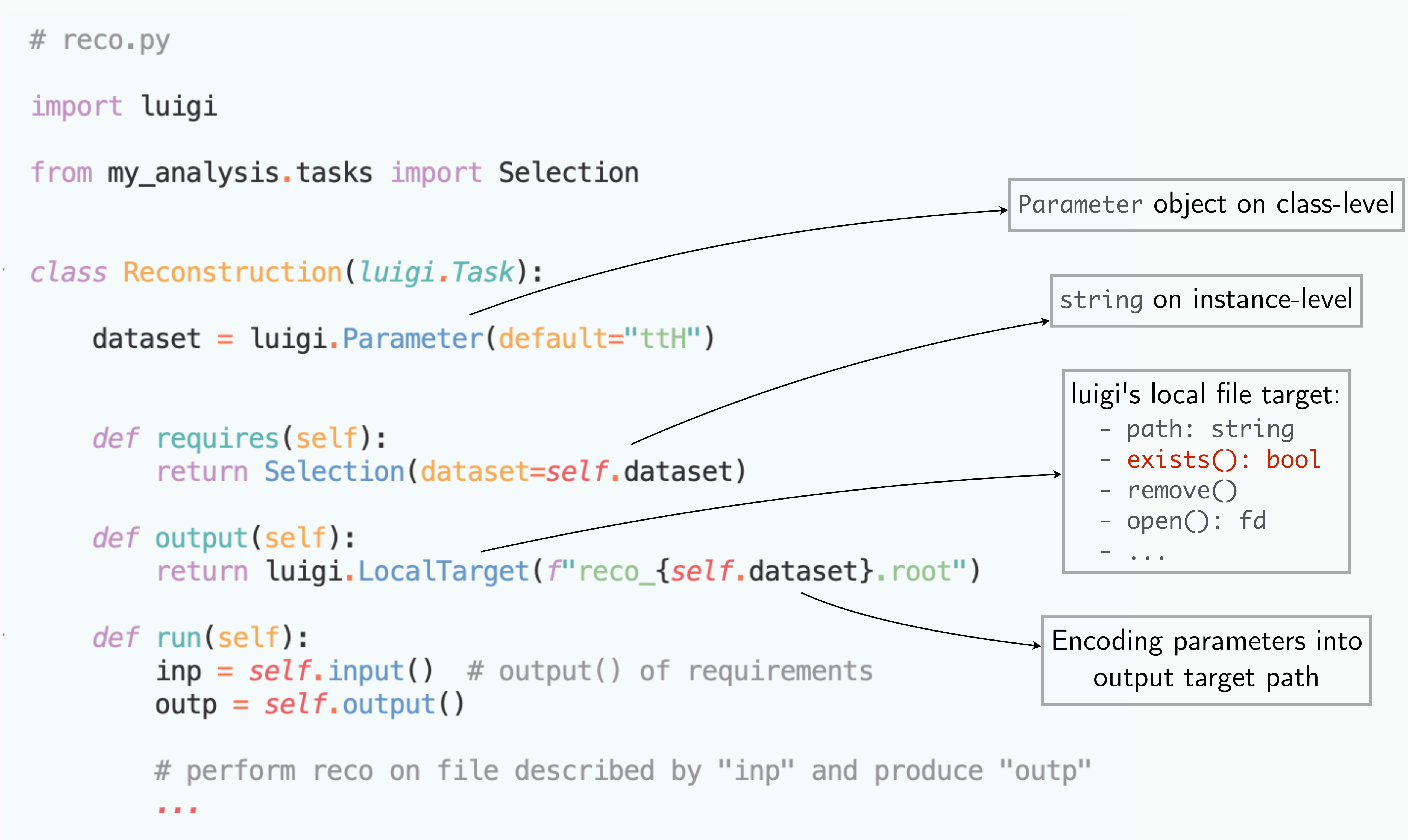
Luigi in a nutshell



```python
# reco.py

import luigi

from my_analysis.tasks import Selection


class Reconstruction(luigi.Task):

    dataset = luigi.Parameter(default="ttH")


    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return luigi.LocalTarget(f"reco_{self.dataset}.root")

    def run(self):
        inp = self.input()  # output() of requirements
        outp = self.output()

        # perform reco on file described by "inp" and produce "outp"
        ...
```

```
> python reco.py Reconstruction --dataset ttbar
```

7 Luigi in a nutshell



```python
# reco.py

import luigi

from my_analysis.tasks import Selection


class Reconstruction(luigi.Task):

    dataset = luigi.Parameter(default="ttH")


    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return luigi.LocalTarget(f"reco_{self.dataset}.root")

    def run(self):
        inp = self.input()  # output() of requirements
        outp = self.output()

        # perform reco on file described by "inp" and produce "outp"
        ...
```

Parameter object on class-level

string on instance-level

luigi's local file target:

- path: string
- exists(): bool
- remove()
- open(): fd
- ...

Encoding parameters into output target path

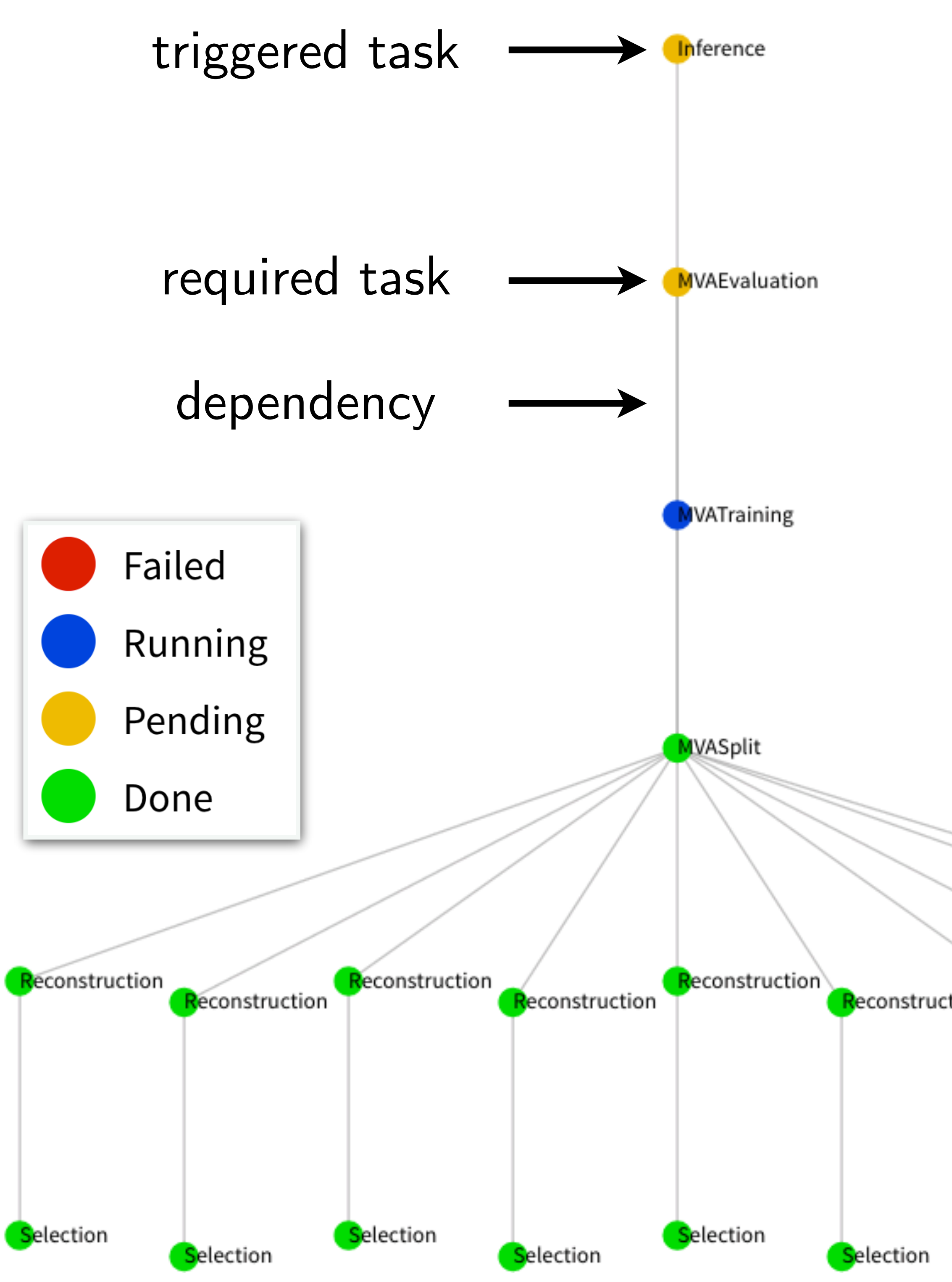
```
> python reco.py Reconstruction --dataset ttbar
```

make-like execution system

- Luigi's execution model is make-like

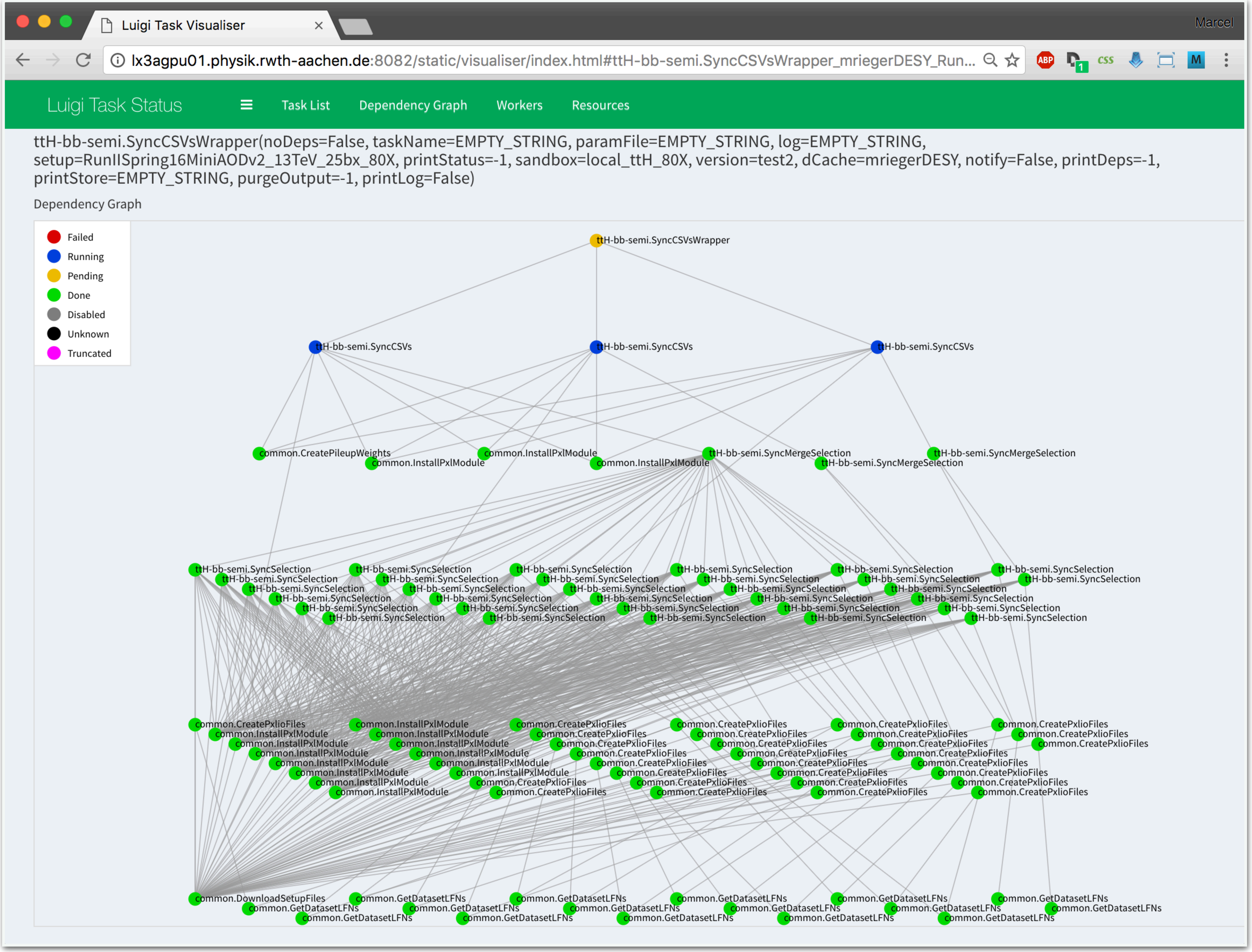
1. Create dependency tree for triggered task
2. Determine tasks to actually run:
   - Walk through tree (top-down)
   - For each path, stop if all output targets of a task exist*

- Only processes what is really necessary
- Scalable through simple structure
- Error handling & automatic re-scheduling



* in this case, the task is considered `complete`

9 Example dependency trees



Luigi Task Visualiser

lx3agpu01.physik.rwth-aachen.de:8082/static/visualiser/index.html#ttH-bb-semi.SyncCSVsWrapper_mriegerDESY_Run...

Luigi Task Status

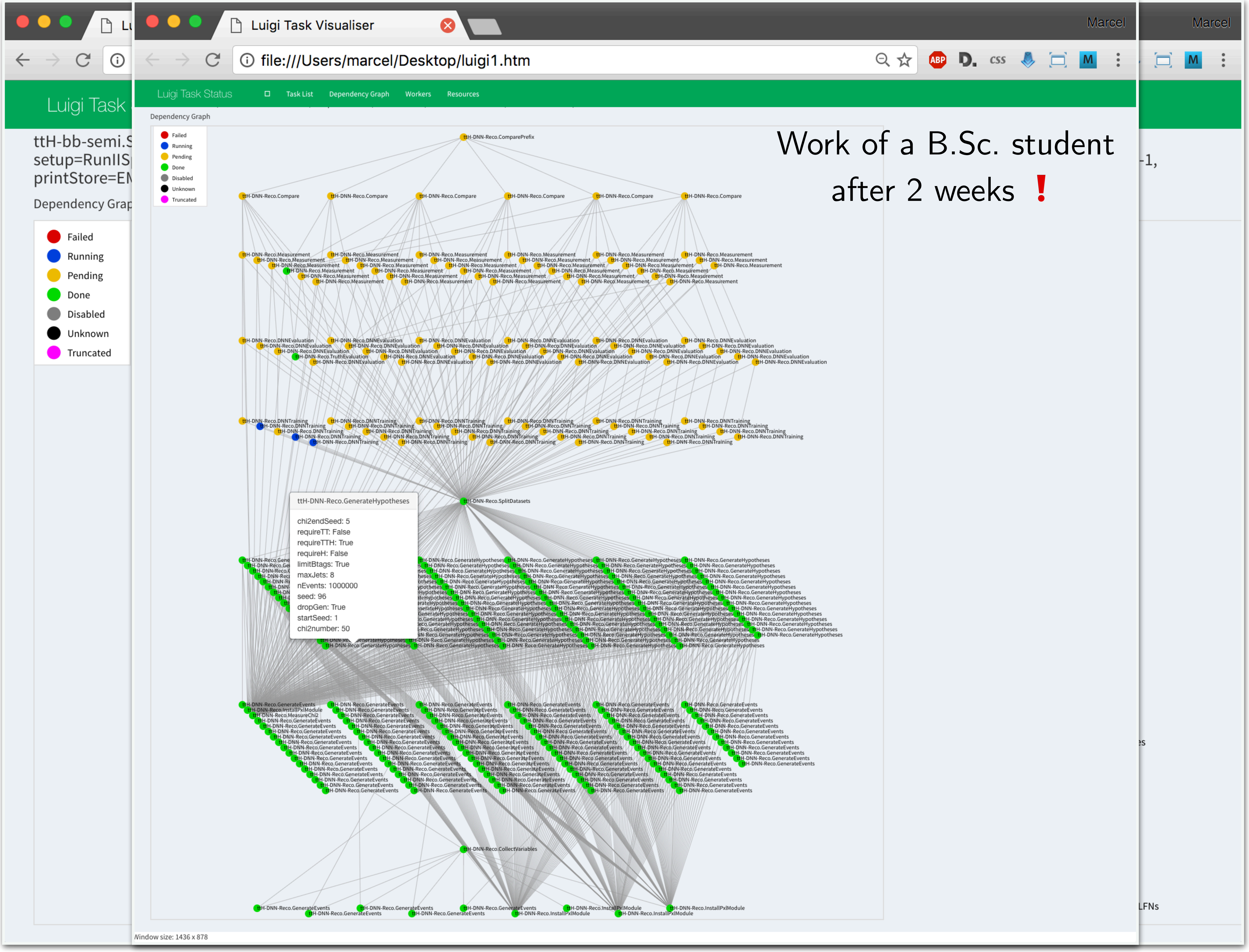
Task List Dependency Graph Workers Resources

ttH-bb-semi.SyncCSVsWrapper(noDeps=False, taskName=EMPTY_STRING, paramFile=EMPTY_STRING, log=EMPTY_STRING, setup=RunIISpring16MiniAODv2_13TeV_25bx_80X, printStatus=-1, sandbox=local_ttH_80X, version=test2, dCache=mriegerDESY, notify=False, printDeps=-1, printStore=EMPTY_STRING, purgeOutput=-1, printLog=False)

Dependency Graph

● Failed
● Running
● Pending
● Done
● Disabled
● Unknown
● Truncated

Example dependency trees



ttH-DNN-Reco.GenerateHypotheses

chi2endSeed: 5
requireTT: False
requireTTH: True
requireH: False
limitBtags: True
maxJets: 8
nEvents: 1000000
seed: 96
dropGen: True
startSeed: 1
chi2number: 50

Work of a B.Sc. student after 2 weeks !

law

luigi analysis workflow

- law: extension **on top** of *luigi* (i.e. it does not replace *luigi*)

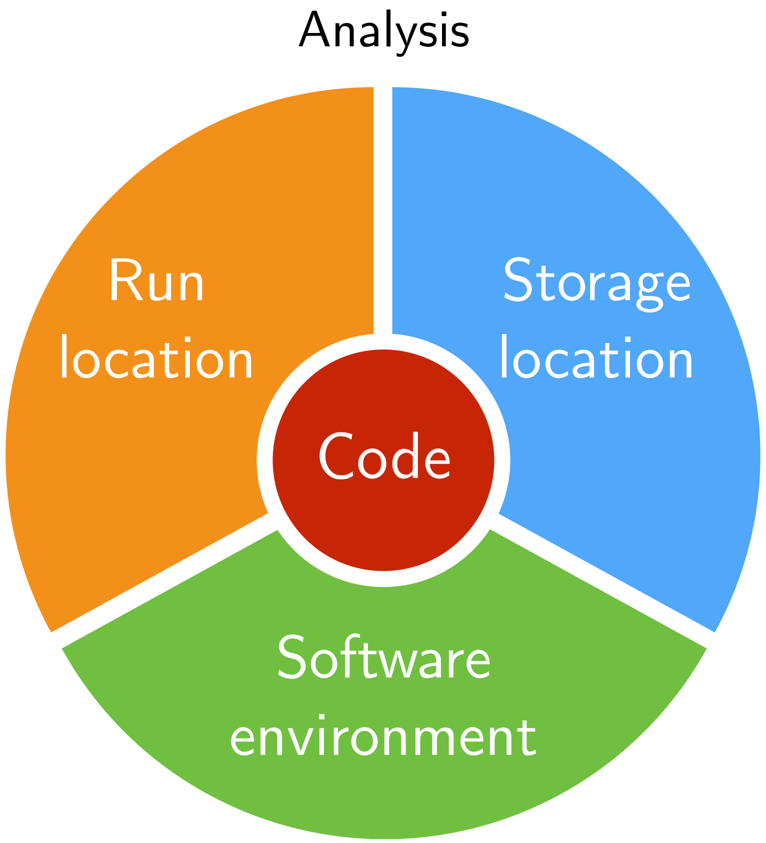
- Software design follows 3 primary goals:

  1. Experiment-agnostic core (and not even HEP-related)
  2. Scalability on HEP infrastructure (but not limited to it)
  3. Decoupling of **run locations**, **storage locations** & **software environments**
     - ▻ Not constrained to specific resources
     - ▻ All components interchangeable

- Toolbox to follow an **analysis design pattern**
  - No constraint on language or data structures

  → Not a *framework*

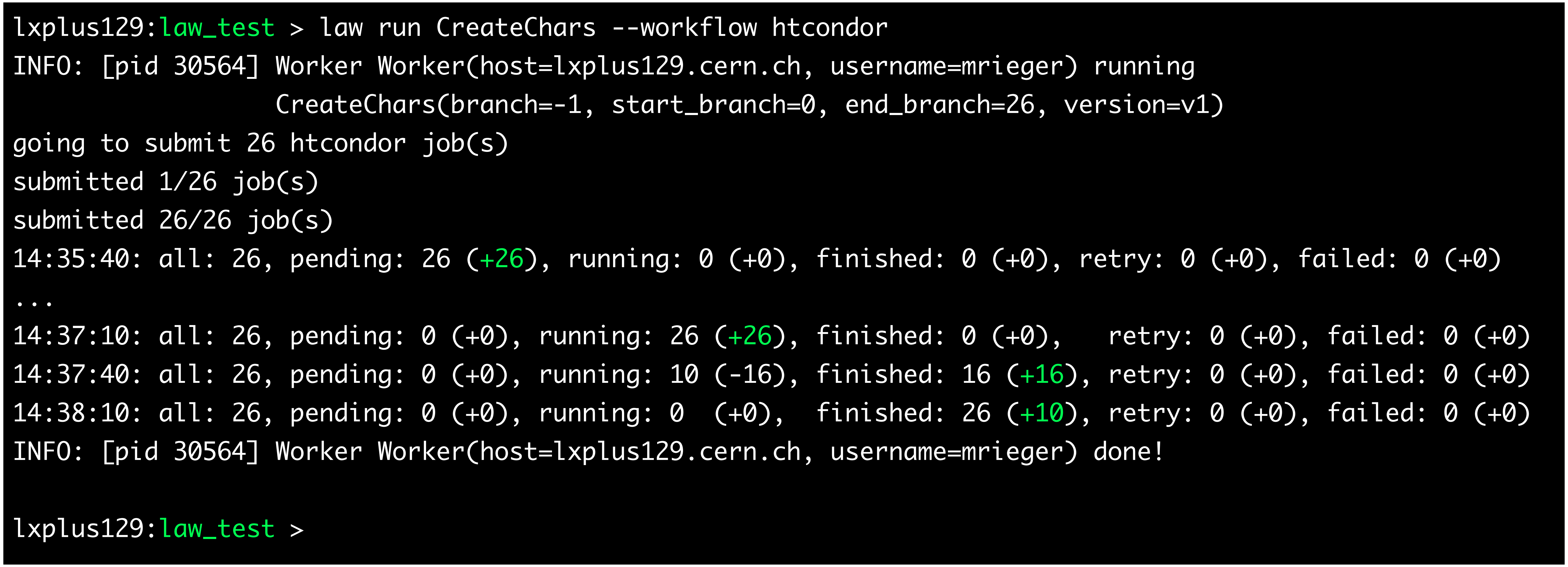
- Currently mostly used within CMS
  - O(10-15) analyses
  - Higgs, Tau, BTag, GEM, HGCAL groups



Law features (1)

1. Job submission

- Idea: submission built into tasks, **no need to write extra code**
- Currently supported job systems: HTCondor, LSF, gLite, ARC, (Slurm + CRAB in dev.)
- Mandatory features such as automatic resubmission, flexible task ↔ job matching, job files fully configurable at submission time, ...
- From the htcondor_at_cern example:



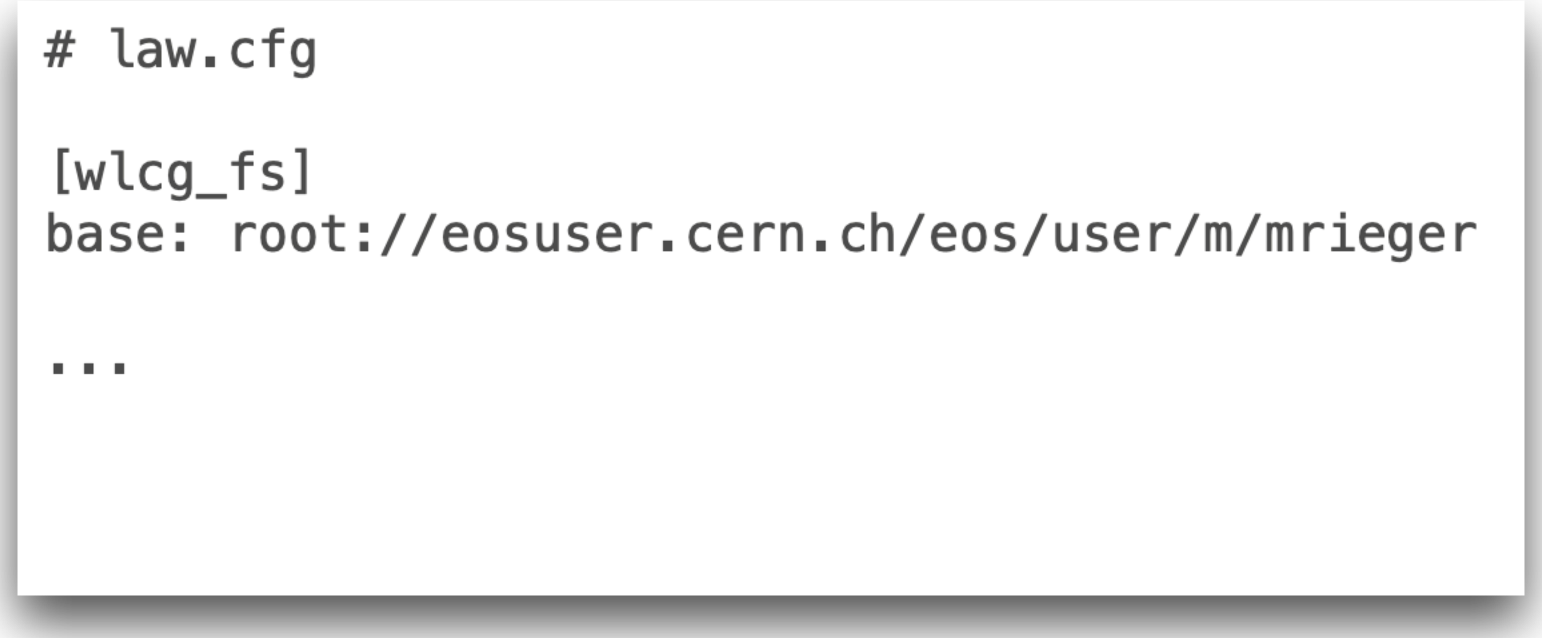
```
lxplus129:law_test > law run CreateChars --workflow htcondor
INFO: [pid 30564] Worker Worker(host=lxplus129.cern.ch, username=mrieger) running
                  CreateChars(branch=-1, start_branch=0, end_branch=26, version=v1)
going to submit 26 htcondor job(s)
submitted 1/26 job(s)
submitted 26/26 job(s)
14:35:40: all: 26, pending: 26 (+26), running: 0 (+0), finished: 0 (+0), retry: 0 (+0), failed: 0 (+0)
...
14:37:10: all: 26, pending: 0 (+0), running: 26 (+26), finished: 0 (+0),   retry: 0 (+0), failed: 0 (+0)
14:37:40: all: 26, pending: 0 (+0), running: 10 (-16), finished: 16 (+16), retry: 0 (+0), failed: 0 (+0)
14:38:10: all: 26, pending: 0 (+0), running: 0  (+0),  finished: 26 (+10), retry: 0 (+0), failed: 0 (+0)
INFO: [pid 30564] Worker Worker(host=lxplus129.cern.ch, username=mrieger) done!

lxplus129:law_test >
```

13 Law features (2)

2. Remote targets

- Idea: work with remote files **as if they were local**
- Remote targets built on top of GFAL2 Python bindings
  - ▻ Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, more info)
- Mandatory features: automatic retries, **local caching**, configurable protocols, round-robin

"FileSystem" configuration



```
# law.cfg

[wlcg_fs]
base: root://eosuser.cern.ch/eos/user/m/mrieger

...
```

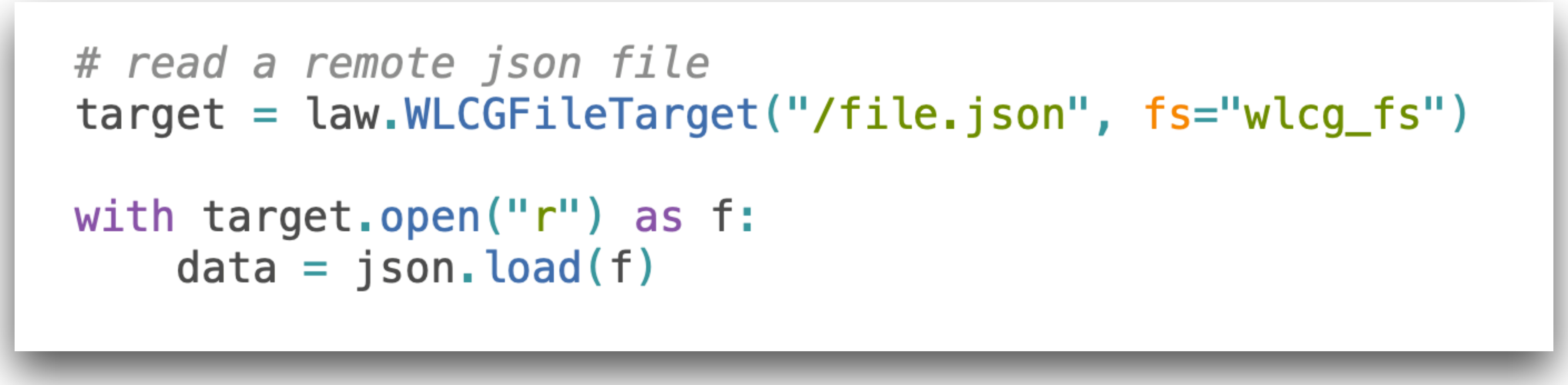
- Base path prefixed to all paths using this "fs"
- Configurable per file operation (stat, listdir, ...)
- Protected against removal of parent directories

13 Law features (2)

2. Remote targets

- Idea: work with remote files **as if they were local**
- Remote targets built on top of GFAL2 Python bindings
  - ▻ Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, more info)
- Mandatory features: automatic retries, **local caching**, configurable protocols, round-robin

Conveniently reading remote files



```
# read a remote json file
target = law.WLCGFileTarget("/file.json", fs="wlcg_fs")

with target.open("r") as f:
    data = json.load(f)
```

Law features (2)

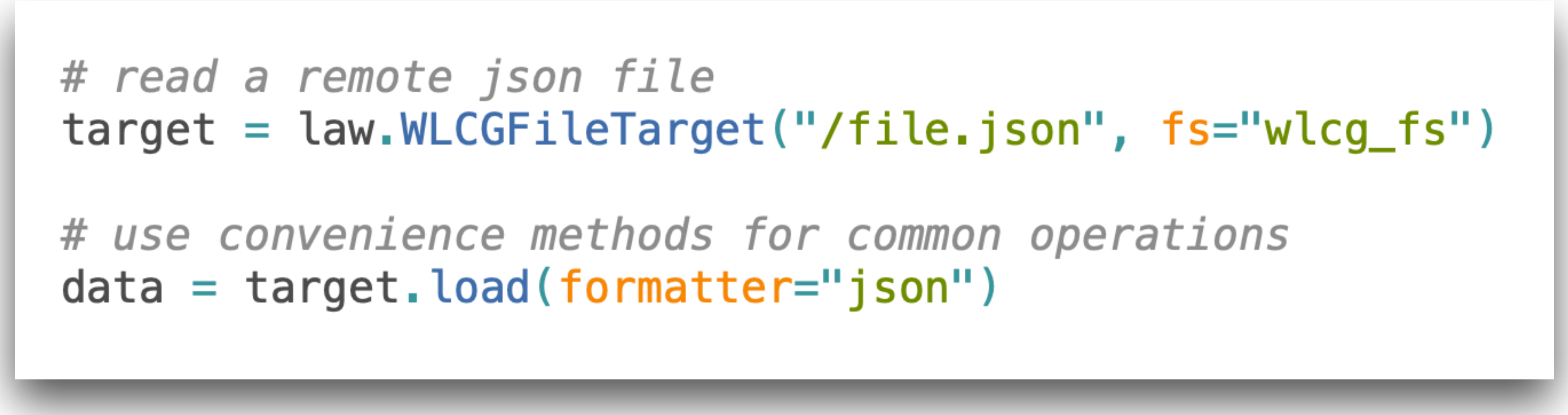
2. Remote targets

- Idea: work with remote files **as if they were local**

- Remote targets built on top of GFAL2 Python bindings
  - ▻ Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, more info)

- Mandatory features: automatic retries, **local caching**, configurable protocols, round-robin

Conveniently reading remote files



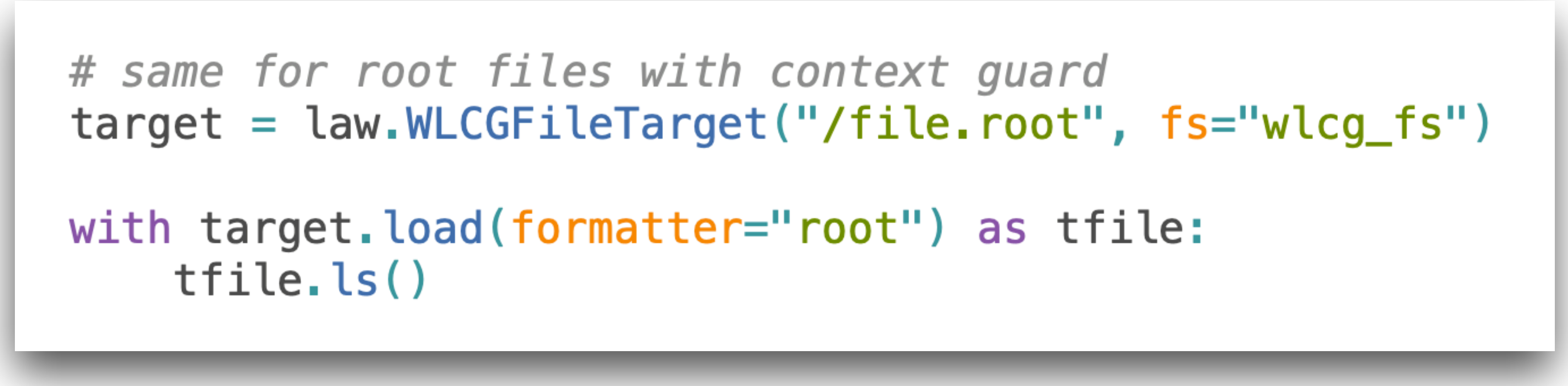
```
# read a remote json file
target = law.WLCGFileTarget("/file.json", fs="wlcg_fs")

# use convenience methods for common operations
data = target.load(formatter="json")
```

13 Law features (2)

2. Remote targets

- Idea: work with remote files **as if they were local**
- Remote targets built on top of GFAL2 Python bindings
  - ▻ Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, more info)
- Mandatory features: automatic retries, **local caching**, configurable protocols, round-robin

Conveniently reading remote files



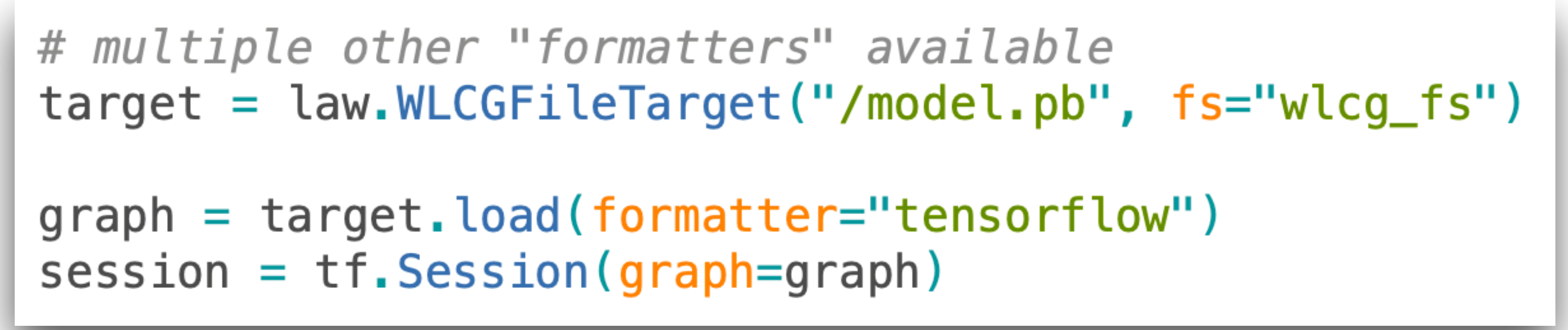
```python
# same for root files with context guard
target = law.WLCGFileTarget("/file.root", fs="wlcg_fs")

with target.load(formatter="root") as tfile:
    tfile.ls()
```

13 Law features (2)

2. Remote targets

- Idea: work with remote files **as if they were local**
- Remote targets built on top of GFAL2 Python bindings
  - ▻ Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, more info)
- Mandatory features: automatic retries, **local caching**, configurable protocols, round-robin

Conveniently reading remote files

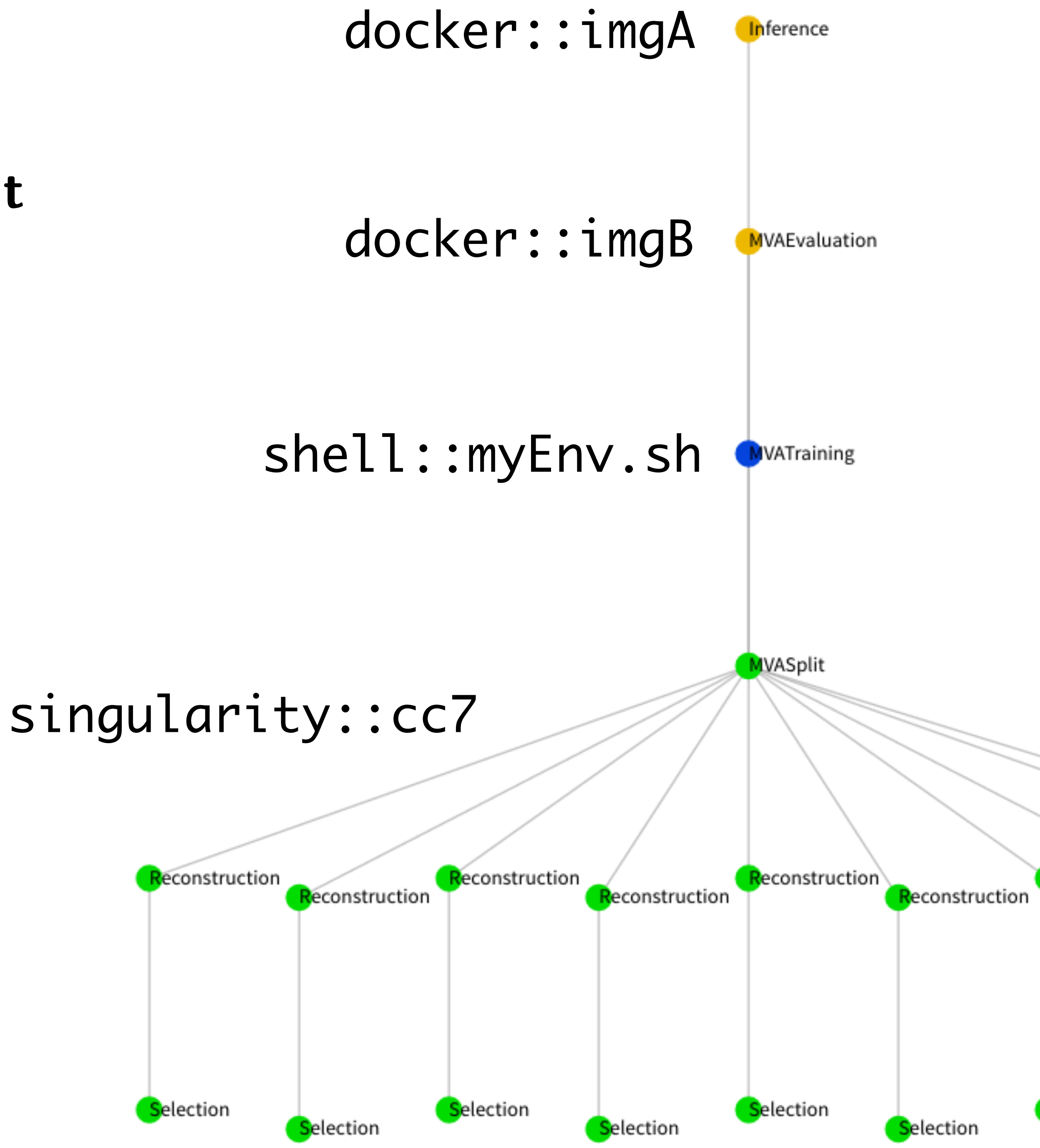


```
# multiple other "formatters" available
target = law.WLCGFileTarget("/model.pb", fs="wlcg_fs")

graph = target.load(formatter="tensorflow")
session = tf.Session(graph=graph)
```

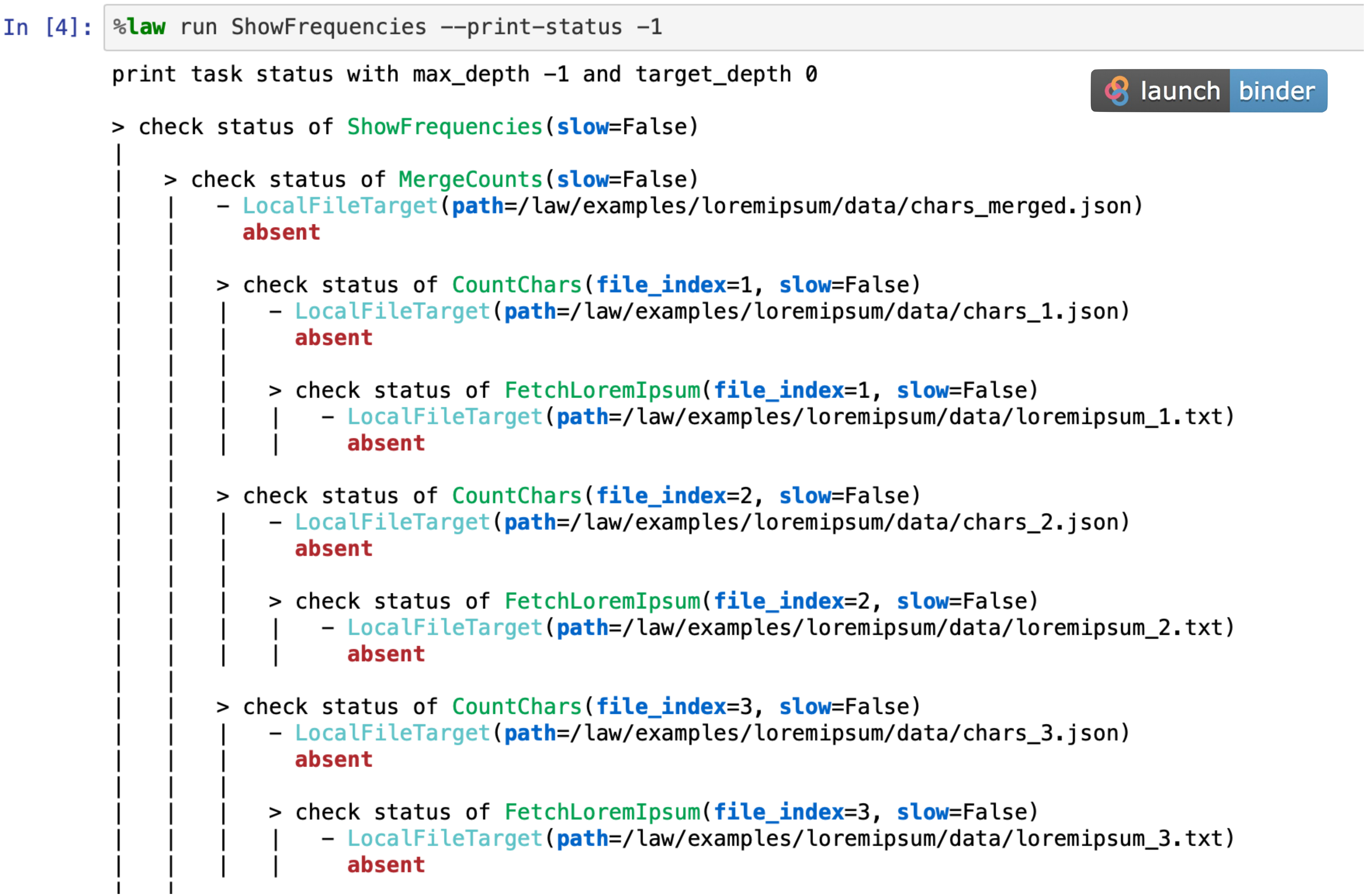
Law features (3)

3. Environment sandboxing

- Diverging software requirements between typical workloads is a great feature / challenge / problem
- Introduce sandboxing:
  - Run entire task in **different environment**
- Existing sandbox implementations:
  - Sub-shell with init file
  - Docker images
  - Singularity images



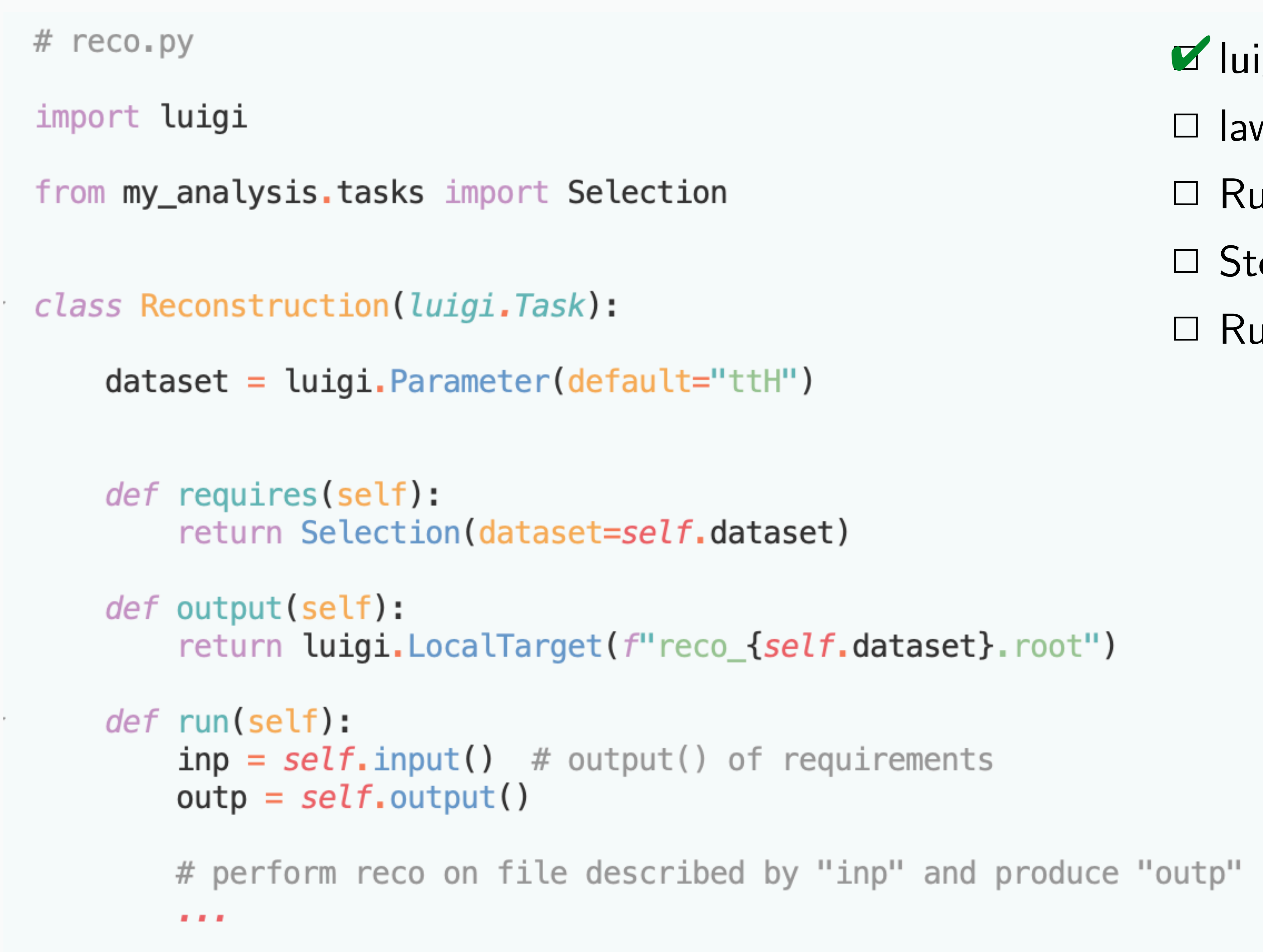
15 Rich command-line interface



```
In [4]: %law run ShowFrequencies --print-status -1
```

```
print task status with max_depth -1 and target_depth 0

> check status of ShowFrequencies(slow=False)
|
|   > check status of MergeCounts(slow=False)
|   |   - LocalFileTarget(path=/law/examples/loremipsum/data/chars_merged.json)
|   |     absent
|   |
|   |   > check status of CountChars(file_index=1, slow=False)
|   |   |   - LocalFileTarget(path=/law/examples/loremipsum/data/chars_1.json)
|   |   |     absent
|   |   |
|   |   |   > check status of FetchLoremIpsum(file_index=1, slow=False)
|   |   |   |   - LocalFileTarget(path=/law/examples/loremipsum/data/loremipsum_1.txt)
|   |   |   |     absent
|   |
|   |   > check status of CountChars(file_index=2, slow=False)
|   |   |   - LocalFileTarget(path=/law/examples/loremipsum/data/chars_2.json)
|   |   |     absent
|   |   |
|   |   |   > check status of FetchLoremIpsum(file_index=2, slow=False)
|   |   |   |   - LocalFileTarget(path=/law/examples/loremipsum/data/loremipsum_2.txt)
|   |   |   |     absent
|   |
|   |   > check status of CountChars(file_index=3, slow=False)
|   |   |   - LocalFileTarget(path=/law/examples/loremipsum/data/chars_3.json)
|   |   |     absent
|   |   |
|   |   |   > check status of FetchLoremIpsum(file_index=3, slow=False)
|   |   |   |   - LocalFileTarget(path=/law/examples/loremipsum/data/loremipsum_3.txt)
|   |   |   |     absent
|   |
```

16 Law in action



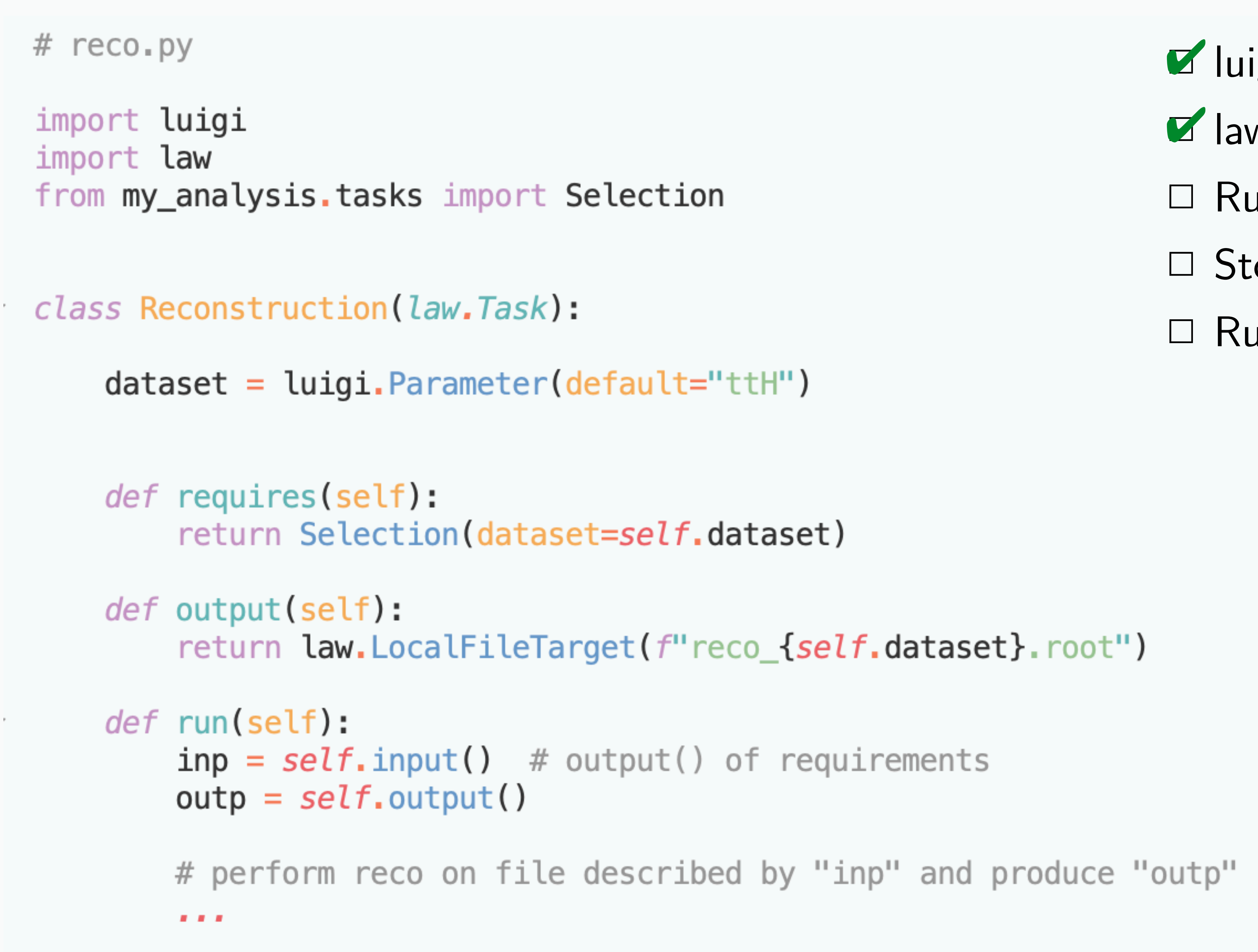
```
# reco.py

import luigi

from my_analysis.tasks import Selection


class Reconstruction(luigi.Task):

    dataset = luigi.Parameter(default="ttH")


    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return luigi.LocalTarget(f"reco_{self.dataset}.root")

    def run(self):
        inp = self.input()  # output() of requirements
        outp = self.output()

        # perform reco on file described by "inp" and produce "outp"
        ...
```

- ☑ luigi task
- □ law task
- □ Run on HTCondor
- □ Store on EOS
- □ Run in docker

Example ☞

```
> python reco.py Reconstruction --dataset ttbar
```

16 Law in action



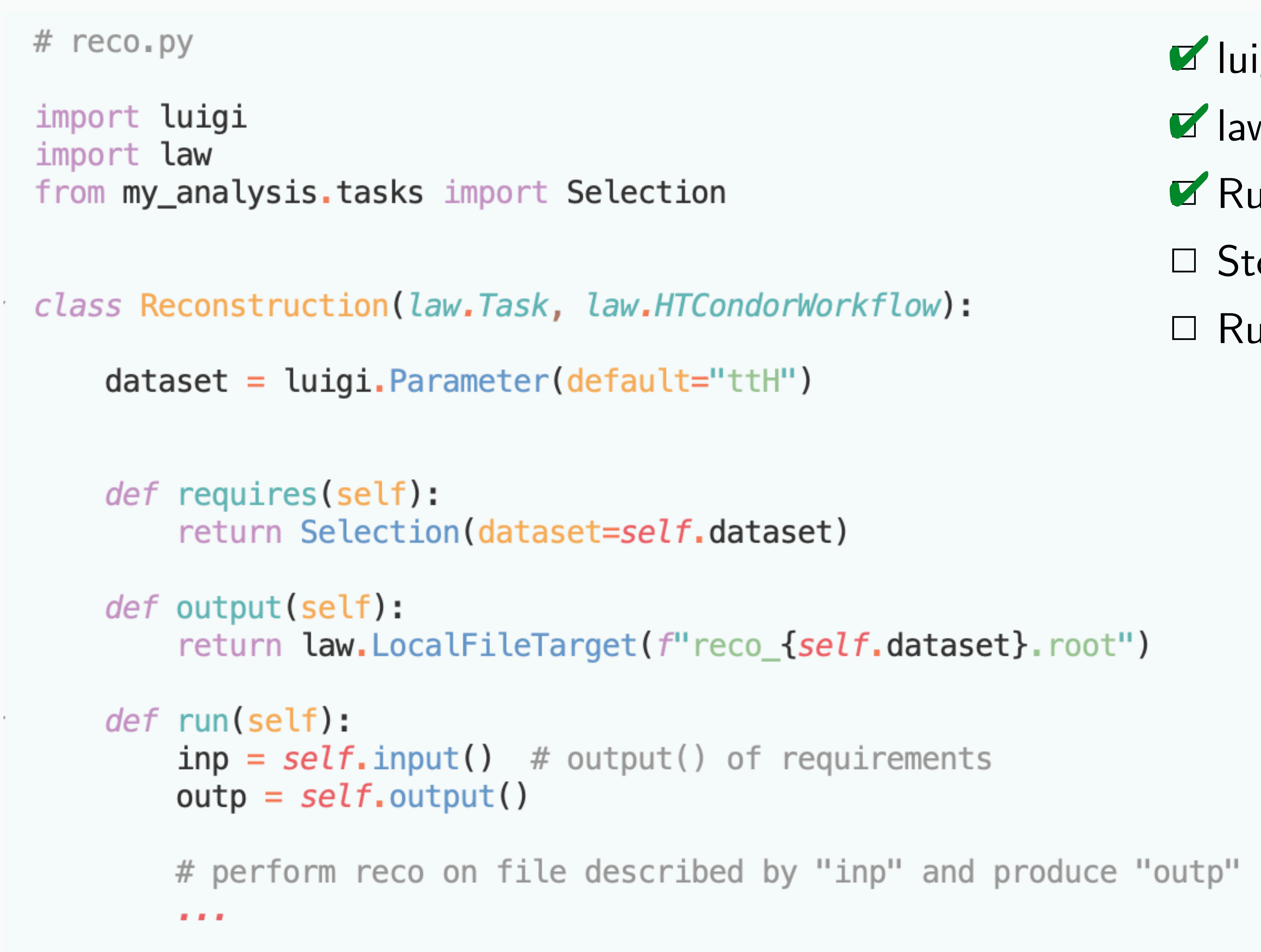
```python
# reco.py

import luigi
import law
from my_analysis.tasks import Selection


class Reconstruction(law.Task):

    dataset = luigi.Parameter(default="ttH")


    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return law.LocalFileTarget(f"reco_{self.dataset}.root")

    def run(self):
        inp = self.input()  # output() of requirements
        outp = self.output()

        # perform reco on file described by "inp" and produce "outp"
        ...
```

- ☑ luigi task
- ☑ law task
- ☐ Run on HTCondor
- ☐ Store on EOS
- ☐ Run in docker

Example ☞

```
> law run Reconstruction --dataset ttbar
```

16 Law in action



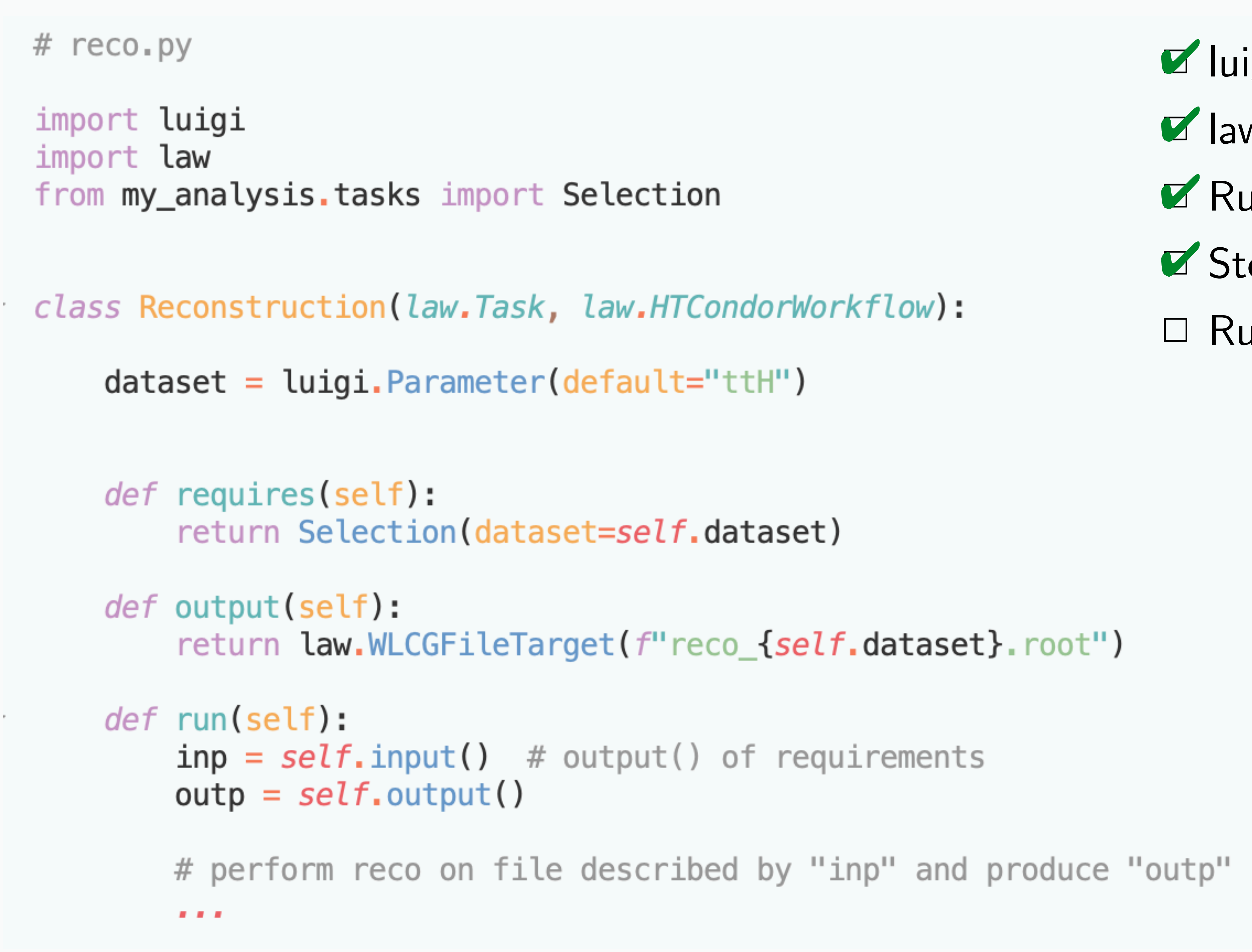
```python
# reco.py

import luigi
import law
from my_analysis.tasks import Selection


class Reconstruction(law.Task, law.HTCondorWorkflow):

    dataset = luigi.Parameter(default="ttH")


    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return law.LocalFileTarget(f"reco_{self.dataset}.root")

    def run(self):
        inp = self.input()  # output() of requirements
        outp = self.output()

        # perform reco on file described by "inp" and produce "outp"
        ...
```

- ☑ luigi task
- ☑ law task
- ☑ Run on HTCondor
- ☐ Store on EOS
- ☐ Run in docker

Example ☞

```
> law run Reconstruction --dataset ttbar --workflow htcondor
```

Law in action



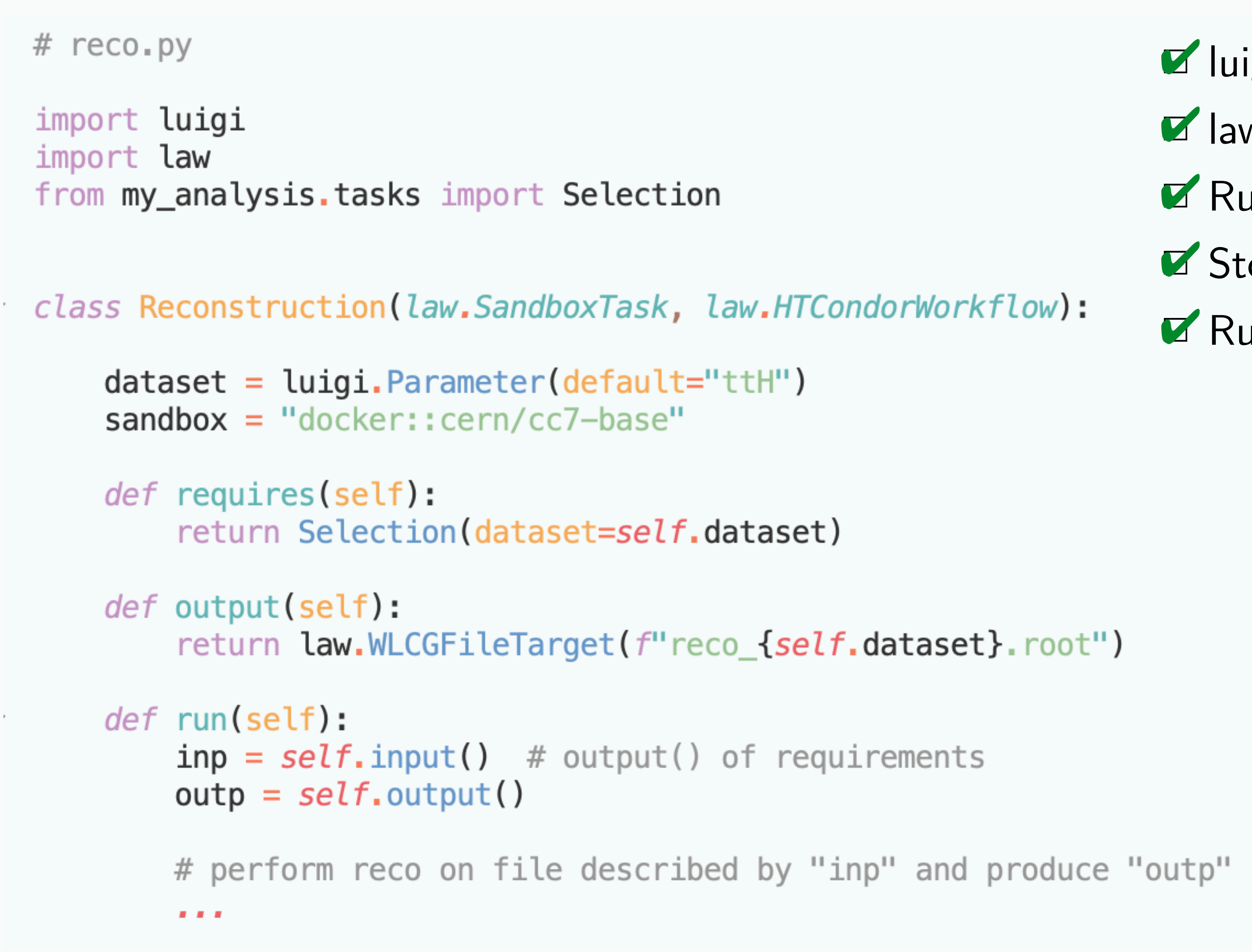
```python
# reco.py

import luigi
import law
from my_analysis.tasks import Selection


class Reconstruction(law.Task, law.HTCondorWorkflow):

    dataset = luigi.Parameter(default="ttH")


    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return law.WLCGFileTarget(f"reco_{self.dataset}.root")

    def run(self):
        inp = self.input()  # output() of requirements
        outp = self.output()

        # perform reco on file described by "inp" and produce "outp"
        ...
```

- ☑ luigi task
- ☑ law task
- ☑ Run on HTCondor
- ☑ Store on EOS
- ☐ Run in docker

Example ☞

```
> law run Reconstruction --dataset ttbar --workflow htcondor
```

16 Law in action



```python
# reco.py

import luigi
import law
from my_analysis.tasks import Selection


class Reconstruction(law.SandboxTask, law.HTCondorWorkflow):

    dataset = luigi.Parameter(default="ttH")
    sandbox = "docker::cern/cc7-base"

    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return law.WLCGFileTarget(f"reco_{self.dataset}.root")

    def run(self):
        inp = self.input()  # output() of requirements
        outp = self.output()

        # perform reco on file described by "inp" and produce "outp"
        ...
```

- ☑ luigi task
- ☑ law task
- ☑ Run on HTCondor
- ☑ Store on EOS
- ☑ Run in docker

Example ☞

```
> law run Reconstruction --dataset ttbar --workflow htcondor
```

Summary

- Resource-agnostic workflow management **essential** for large & complex analyses
- Need for a flexible **analysis design pattern**, not another framework

- Luigi is able to model complex workflows in Pythonic way
- Law extends Luigi in experiment-agnostic way and provides
  - scalability on interchangeable remote resources (file access & job submission)
  - full decoupling of **run locations**, **storage locations** & **software environments**

→ **All** information transparently encoded via tasks, targets & requirements

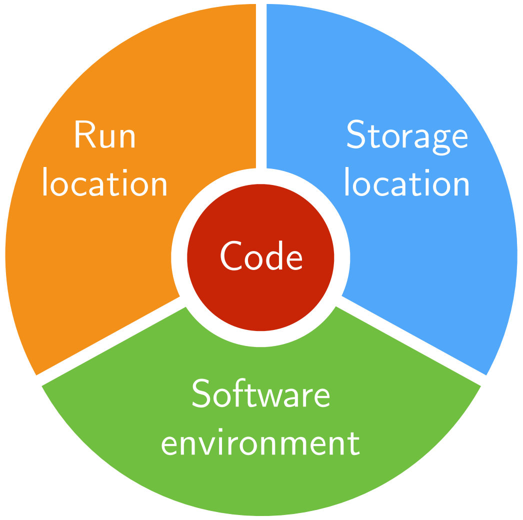
→ **End-to-end automation** of analyses over distributed resources

→ Allows to **interface with existing tasks & code** on any scale (team, group, collaboration, ...)

- Links, documentation & examples
  (e.g. "Hello world", HTCondor example, single top example, CMS HH tools)

law
luigi analysis workflow

github.com/riga/law
law.readthedocs.io

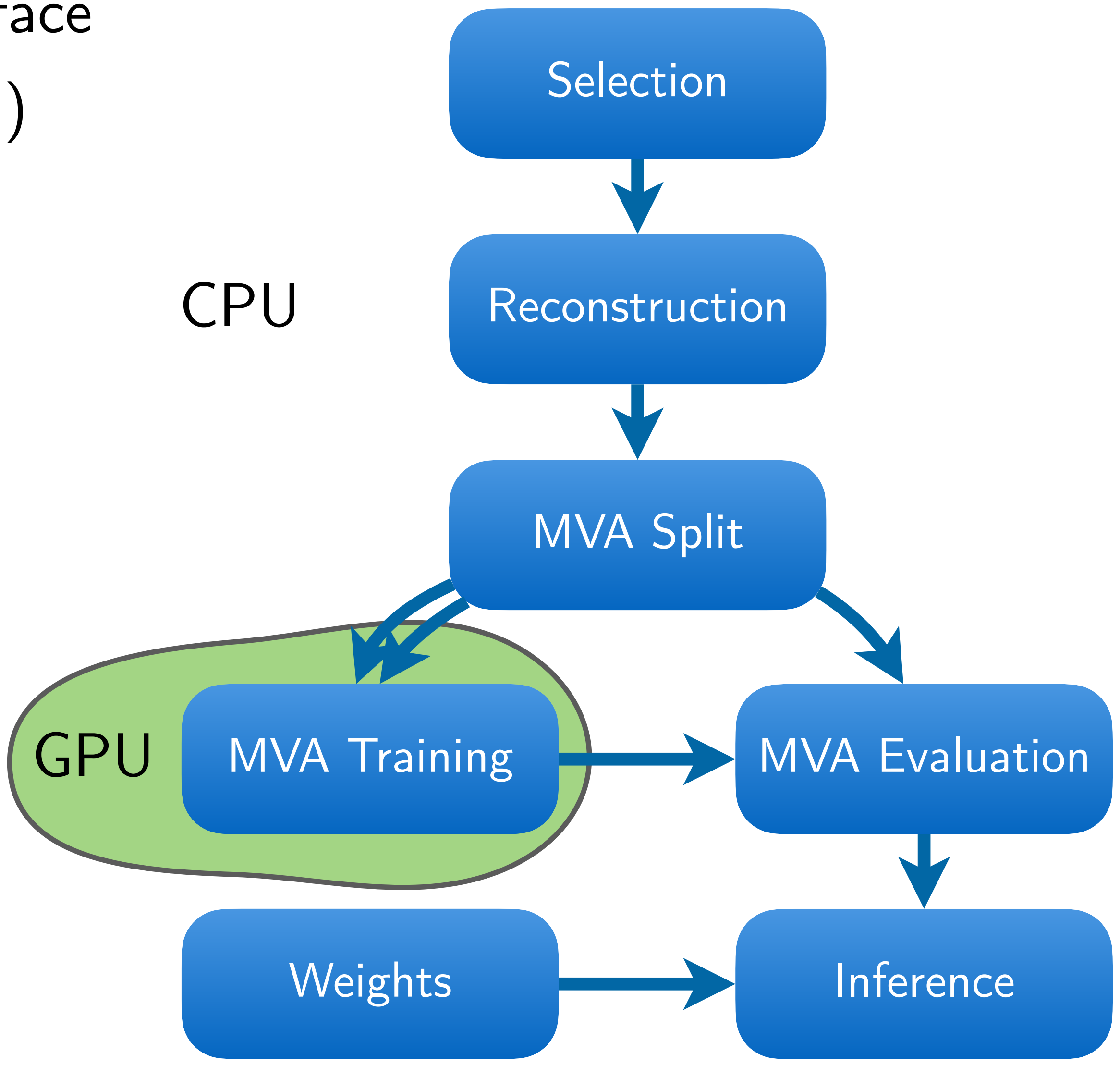


Backup

Abstraction: analysis workflows

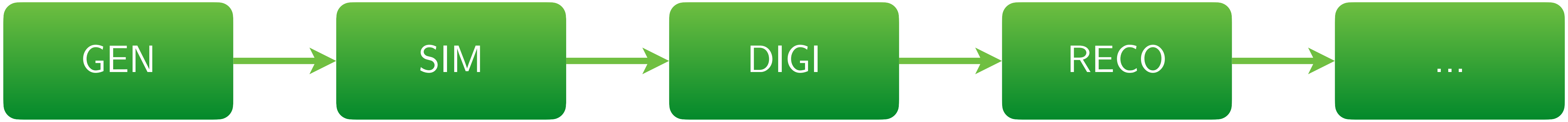
- Workflow, decomposable into particular workloads
- Workloads related to each other by common interface
  - In/outputs define directed acyclic graph (DAG)
- Alter default behavior via parameters
- Computing resources
  - Run location (CPU, GPU, WLCG, ...)
  - Storage location (local, dCache, EOS, ...)
- Software environment
- Collaborative development and processing
- Reproducible intermediate and final results

Example



→ Reads like a checklist for analysis workflow management

Existing WMS: MC production



Tailored systems

- Structure known in advance
- Workflows static & recurring
- One-dimensional design
- Special infrastructure for config and running
- Homogeneous software requirements

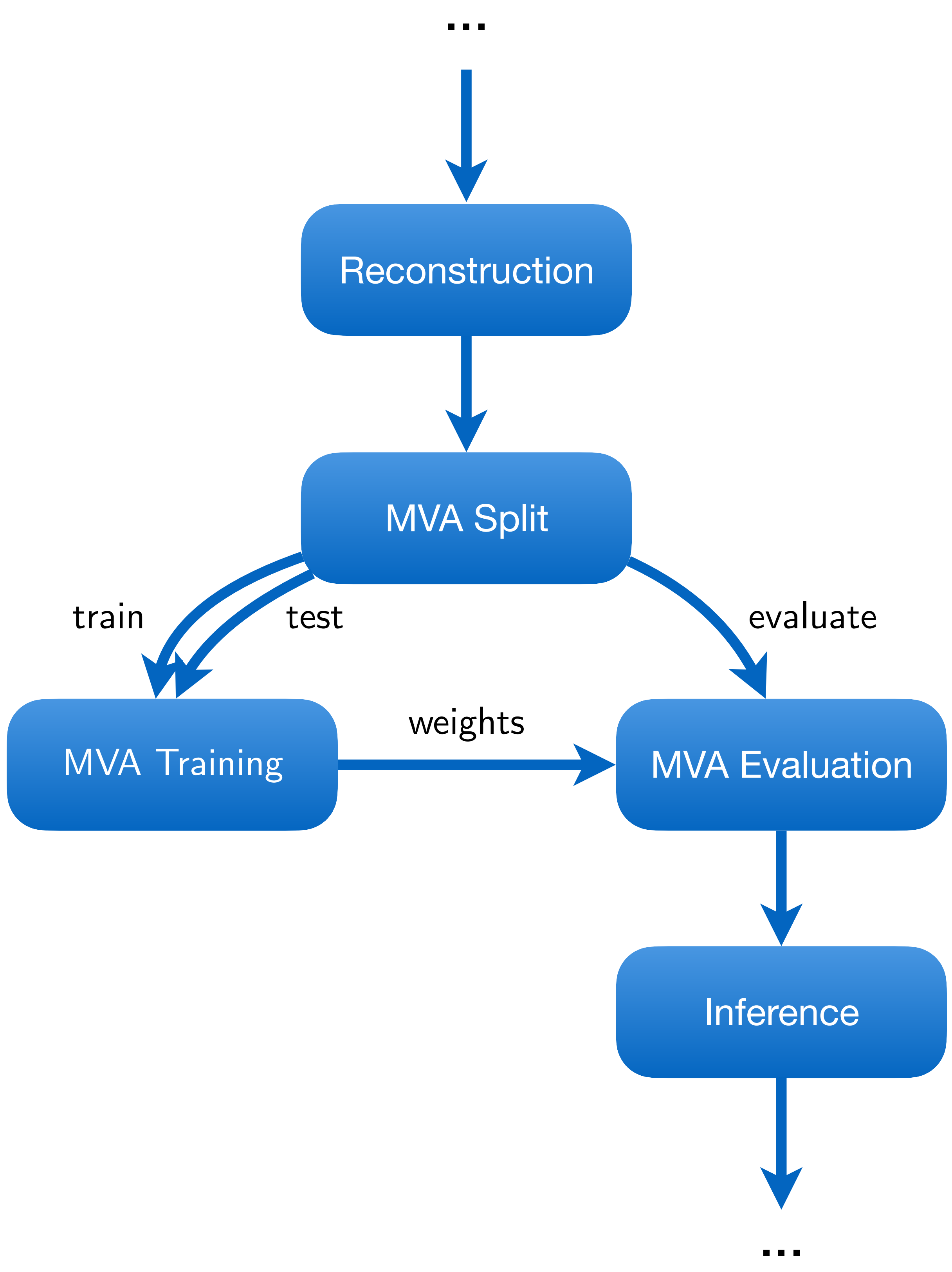
Wishlist for end-user analyses

- Structure "iterative", a-priori unknown
- Dynamic workflows, fast R&D cycles
- Tree design, arbitrary dependencies
- Incorporate existing infrastructure
- Use custom software, everywhere

→ Requirements for HEP analyses mostly orthogonal

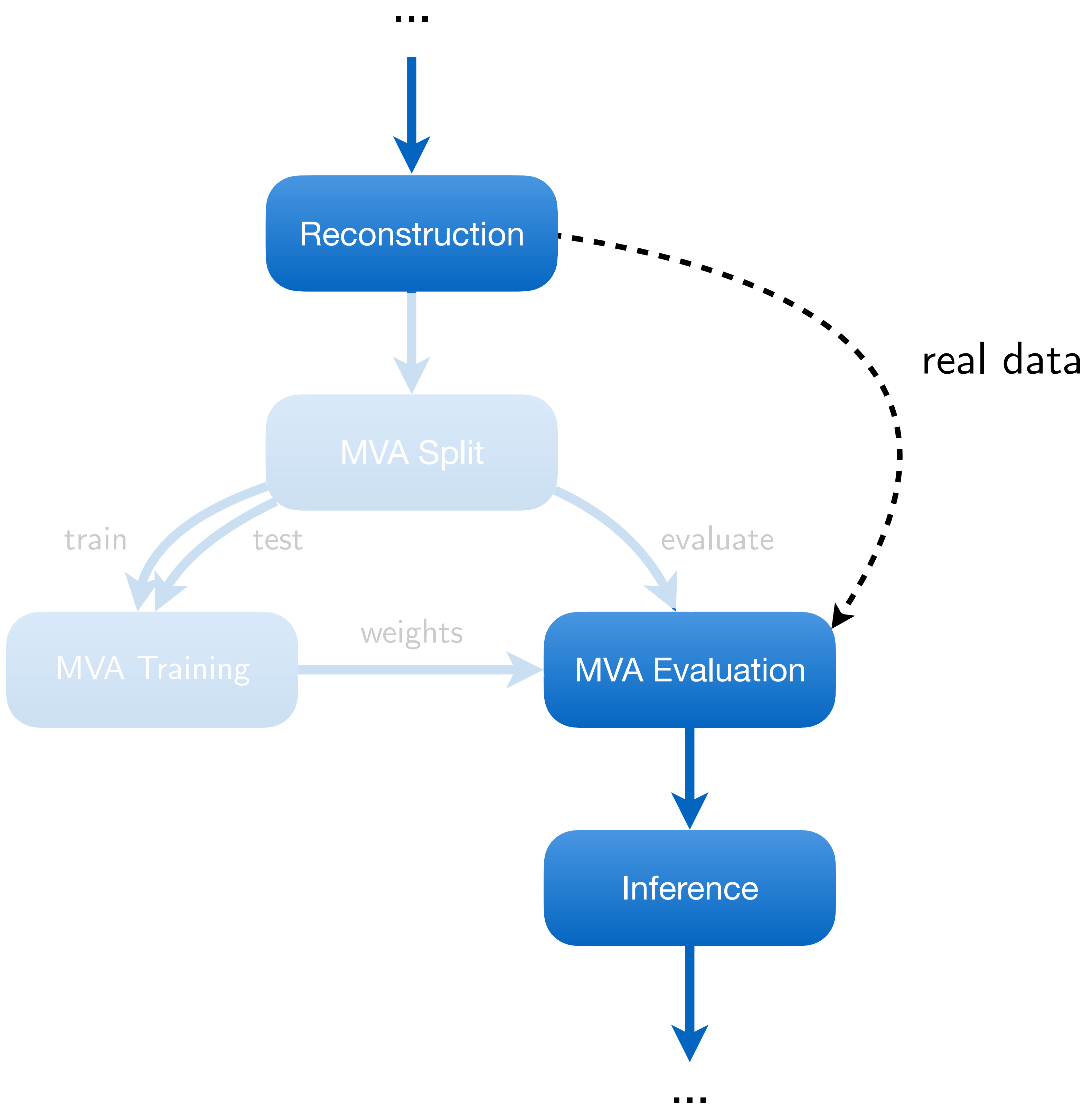
A typical example: ML workflow with uncertainties

Nominal MC



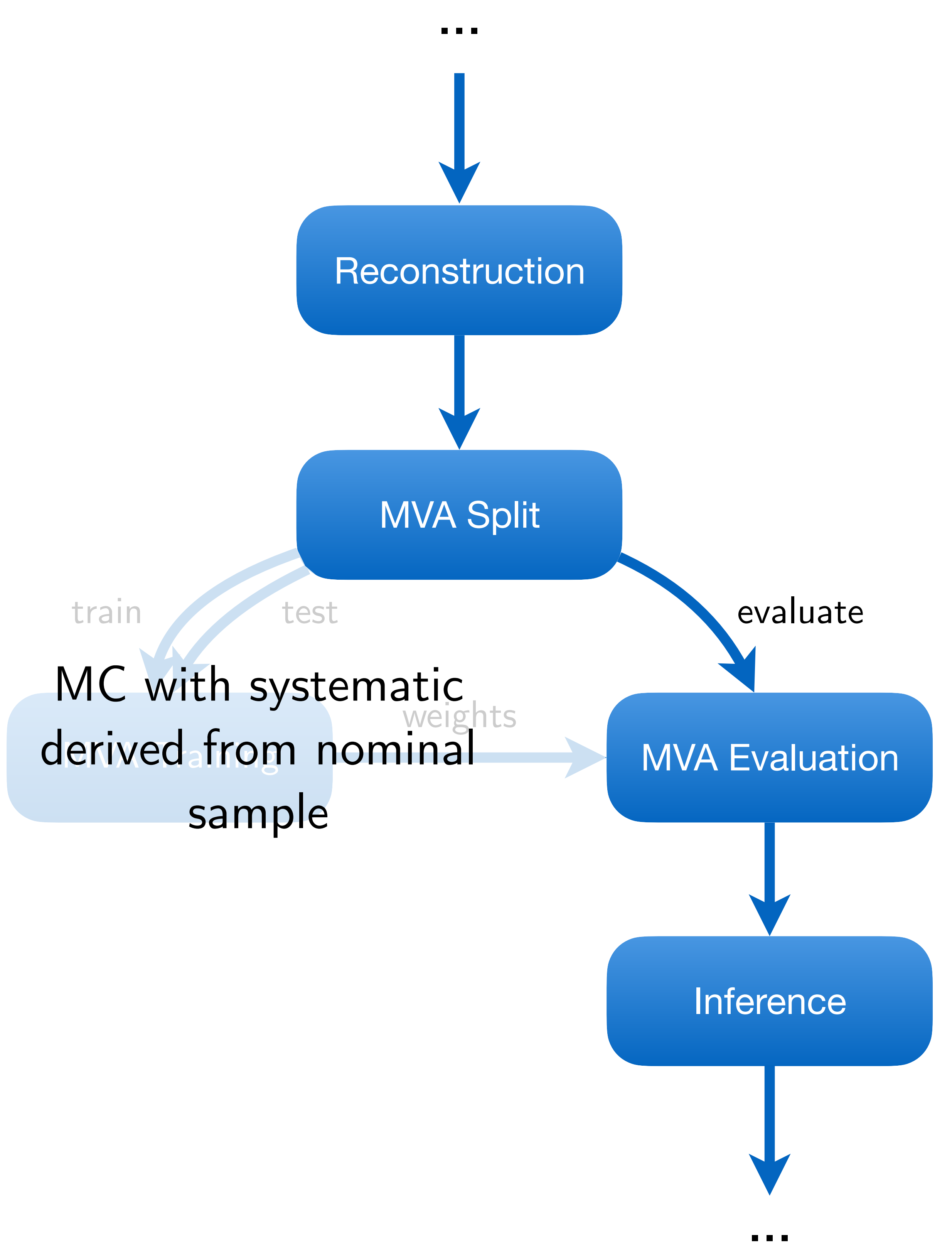
A typical example: ML workflow with uncertainties

Data



A typical example: ML workflow with uncertainties

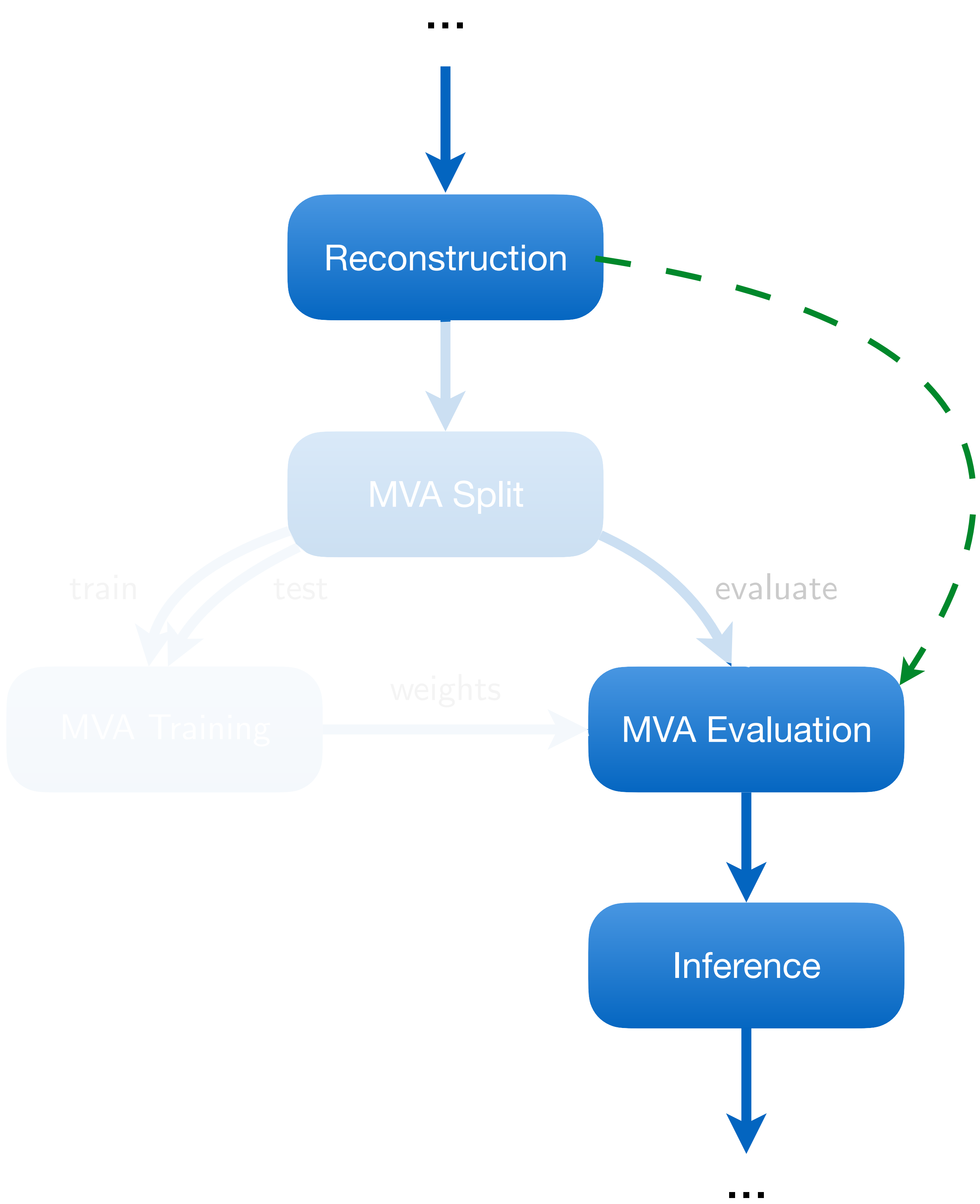
MC, Syst. I



MC with systematic derived from nominal sample

A typical example: ML workflow with uncertainties

MC, Syst. II



MC with systematic generated from new events

Loose collection of thoughts

- **Fast turnaround (*O(2d)*)**
  → Not only a nice-to-have, but can pave the way for repeated tests of new ideas and their impact on the full analysis

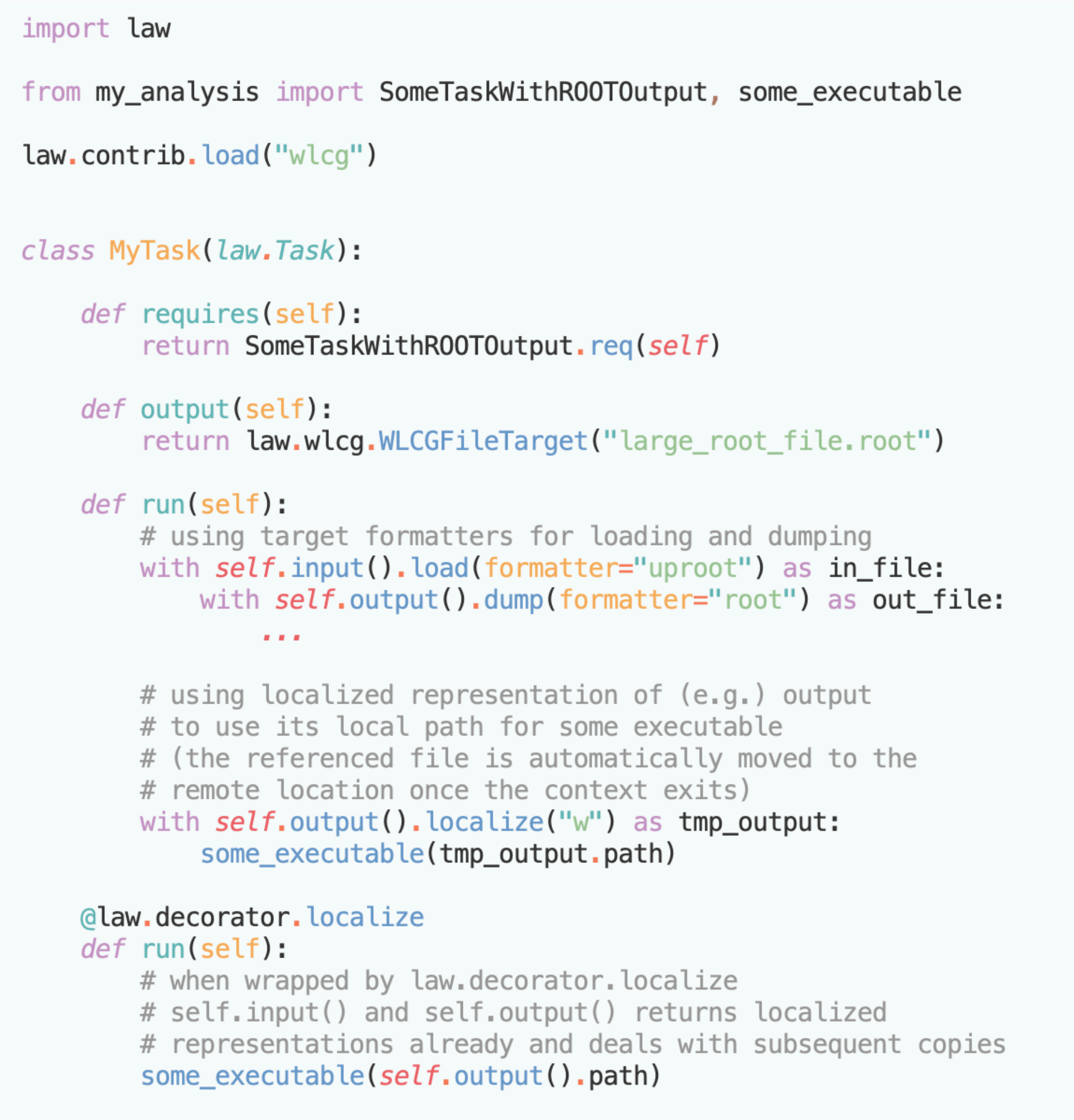
- **IO independence on "framework" software / revisions**
  → Dumping "classdefs" in IO creates "gated communities" (mostly for ROOT IO / C++)
  → Independent IO improves ability to interface with other people, tools, "frameworks"
  → Ability to work with files after O(months / years)

- **No fixation of certain resources**
  → "We have to run at XYZ because the ntuples are there" (can't always be avoided though)
  → Should at least be possible, even if not 100% efficient

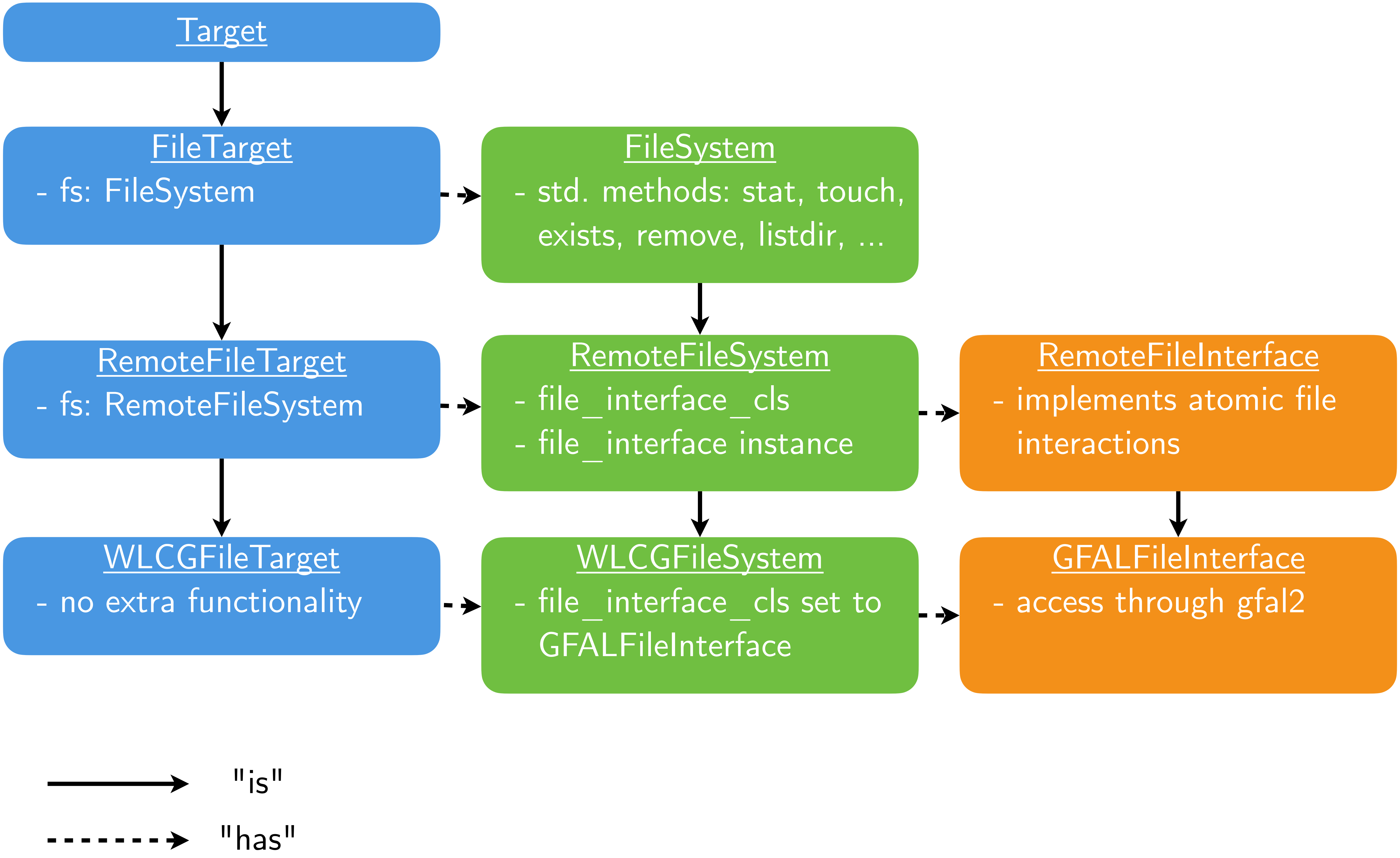
- **Software environment**
  → "Outer" environment should only contain software required to trigger tasks (law, gfal2)
  → Move all software requirements into sandboxes (e.g. a CMSSW sandbox (see later)) which can depend on tasks

(Remote) targets

24 Working with remote targets



```python
import law

from my_analysis import SomeTaskWithROOTOutput, some_executable

law.contrib.load("wlcg")


class MyTask(law.Task):

    def requires(self):
        return SomeTaskWithROOTOutput.req(self)

    def output(self):
        return law.wlcg.WLCGFileTarget("large_root_file.root")

    def run(self):
        # using target formatters for loading and dumping
        with self.input().load(formatter="uproot") as in_file:
            with self.output().dump(formatter="root") as out_file:
                ...

        # using localized representation of (e.g.) output
        # to use its local path for some executable
        # (the referenced file is automatically moved to the
        # remote location once the context exits)
        with self.output().localize("w") as tmp_output:
            some_executable(tmp_output.path)

    @law.decorator.localize
    def run(self):
        # when wrapped by law.decorator.localize
        # self.input() and self.output() returns localized
        # representations already and deals with subsequent copies
        some_executable(self.output().path)
```



Target

FileTarget
- fs: FileSystem

FileSystem
- std. methods: stat, touch, exists, remove, listdir, ...

RemoteFileTarget
- fs: RemoteFileSystem

RemoteFileSystem
- file_interface_cls
- file_interface instance

RemoteFileInterface
- implements atomic file interactions

WLCGFileTarget
- no extra functionality

WLCGFileSystem
- file_interface_cls set to GFALFileInterface

GFALFileInterface
- access through gfal2

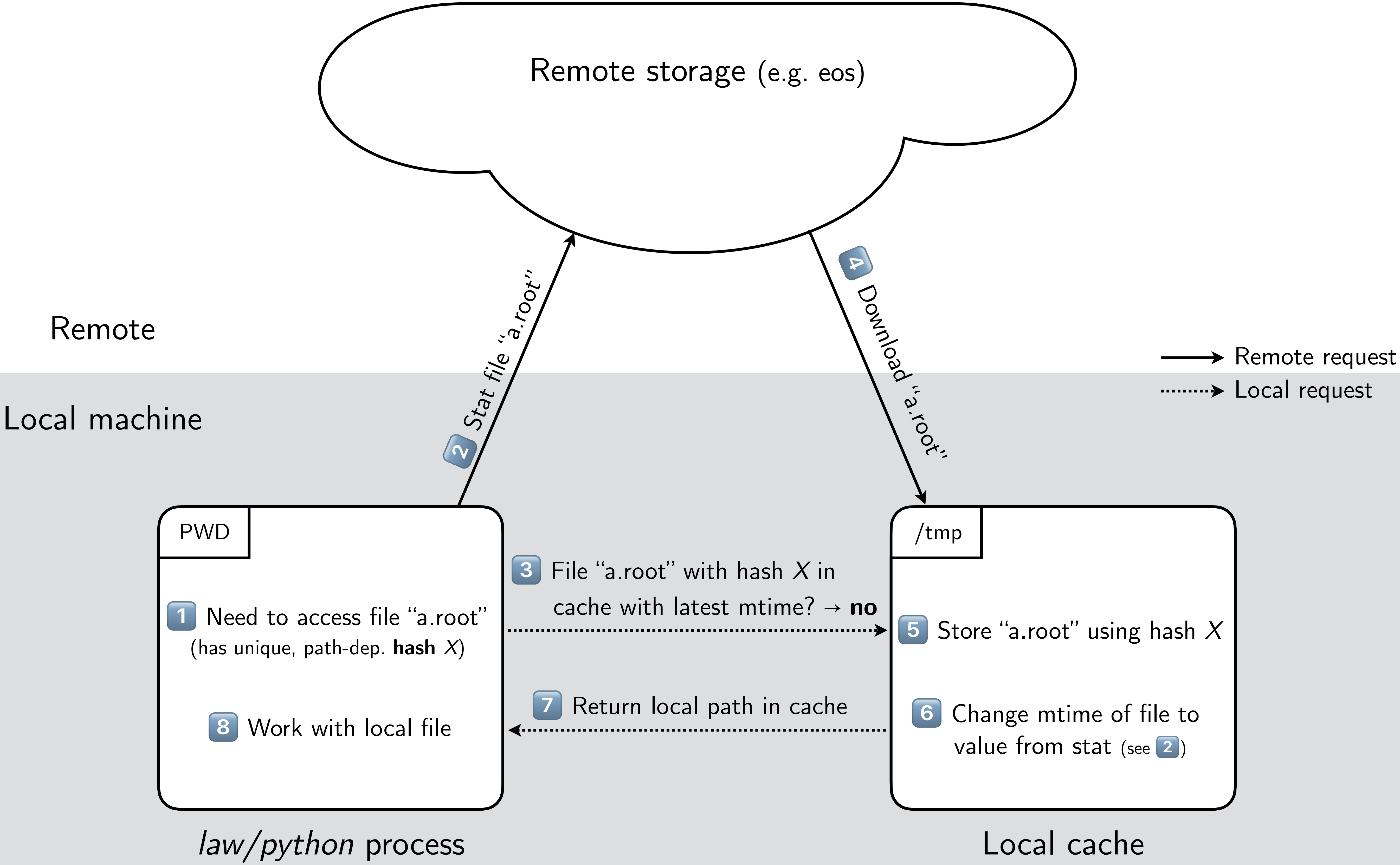
"is"

"has"

Local caching (1)

Scenario A: file not cached *yet*

Configuration ☞



Remote storage (e.g. eos)

Remote

Local machine

→ Remote request

⇢ Local request

PWD

1 Need to access file "a.root" (has unique, path-dep. **hash** *X*)

2 Stat file "a.root"

3 File "a.root" with hash *X* in cache with latest mtime? → **no**

4 Download "a.root"

/tmp

5 Store "a.root" using hash *X*

6 Change mtime of file to value from stat (see 2)

7 Return local path in cache

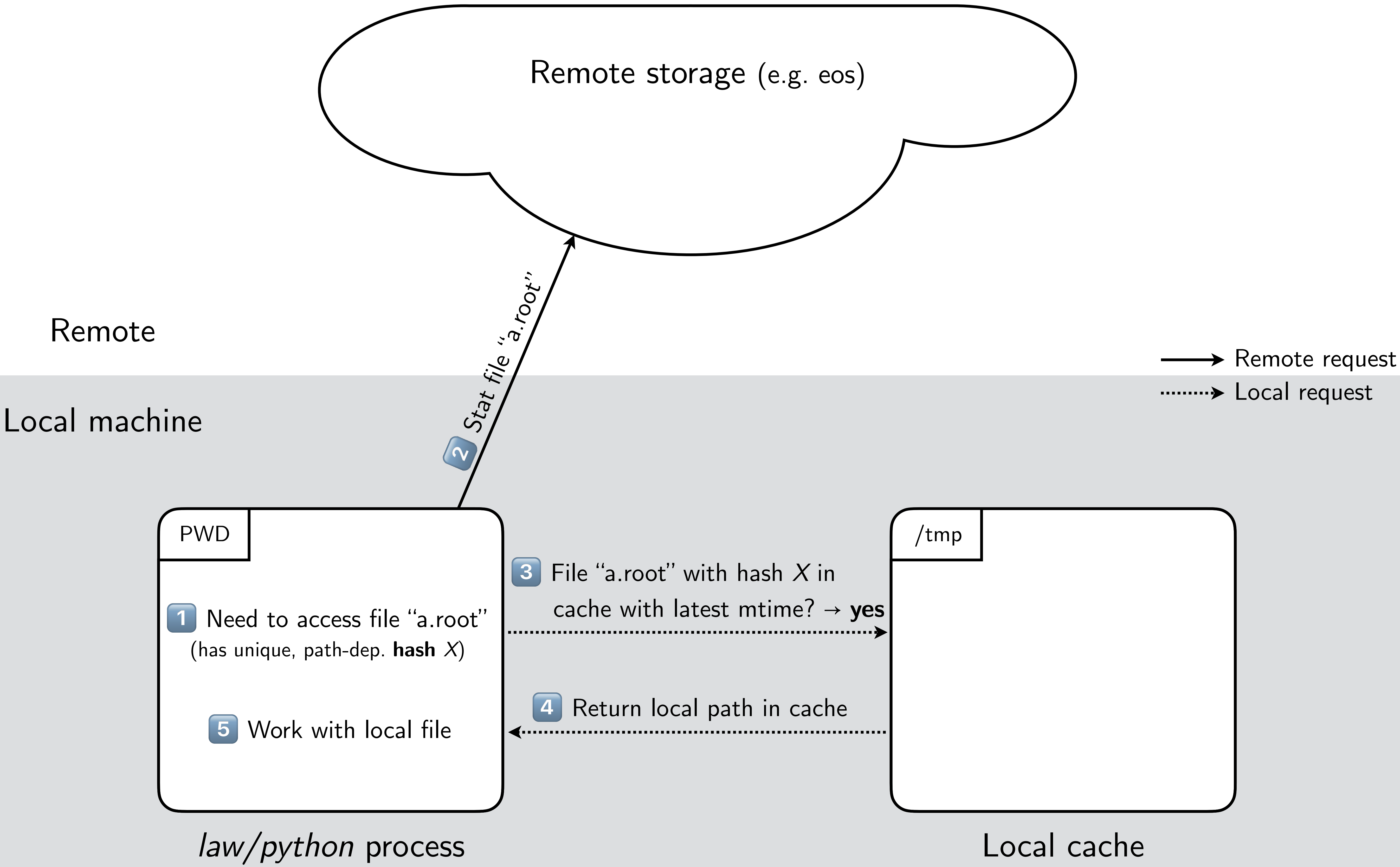
8 Work with local file

law/python process

Local cache

Scenario B: file *already* cached

Configuration ☞


Remote storage (e.g. eos)
Remote
Local machine
2 Stat file "a.root"
→ Remote request
⇢ Local request
PWD
1 Need to access file "a.root"
(has unique, path-dep. **hash** *X*)
5 Work with local file
3 File "a.root" with hash *X* in cache with latest mtime? → **yes**
4 Return local path in cache
/tmp
law/python process
Local cache

Workflows

Workflows: General ideas

- **Many tasks exhibit the same overall structure and/or purpose**
  - *"Run over N existing files" / "Generate N events/toys" / "Merge N into M files"*
  - All these tasks can **profit from the same features**
    - ▻ *"Only process file x and/to y", "Remove outputs of "x, y & z",*
      "Process N files, but consider the task finished once M < N are done", "..."

→ Calls for a generic container object that provides guidance and features for these cases

- **Workflow "containers"**
  - Task that introduces a parameters called `--branch b` (`luigi.IntParameter`)
    - ▻ `b >= 0`: Instantiates particular tasks called "branches"; `run()` will (e.g.) process file `b`
    - ▻ `b = -1`: Instantiates the workflow container itself; `run()` will run* all branch tasks
    - * How branch tasks are run is implemented in different workflow types: local or several remote ones

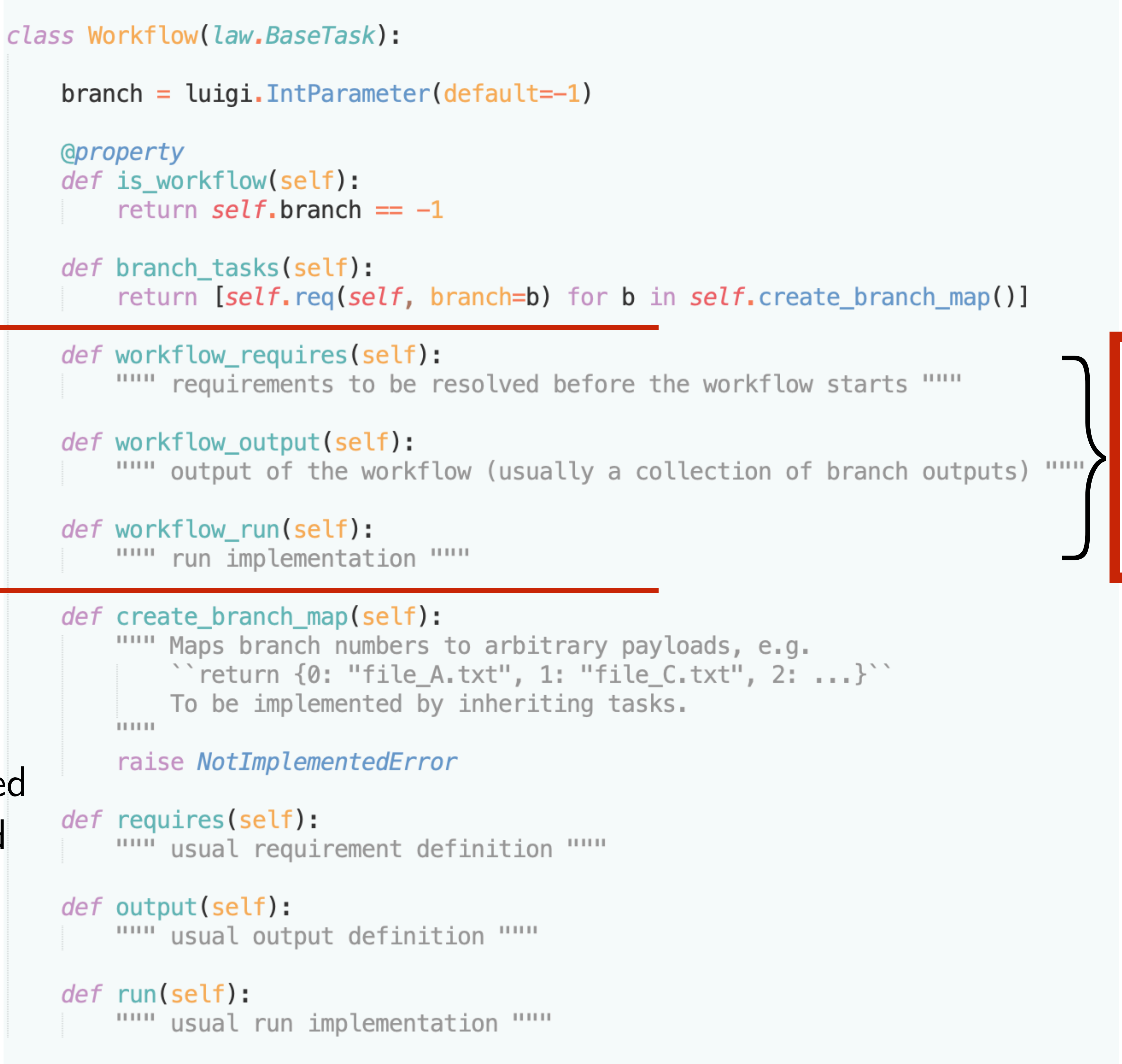
- **Practical advantages**
  - Convenience: same features available in all workflows (see next slides)
  - **Scalability and versatility for remote workflows**
    - ▻ Jobs: Better control of jobs, submission, task-to-job matching ... (see next slides)
    - ▻ Luigi: Central scheduler breaks when pinged by O(10k) tasks every few seconds
    - ▻ Remote storage: allows batched file operations instead of file-by-file requests

Workflows: example implementation

Common

Workflow specific

Implemented when used

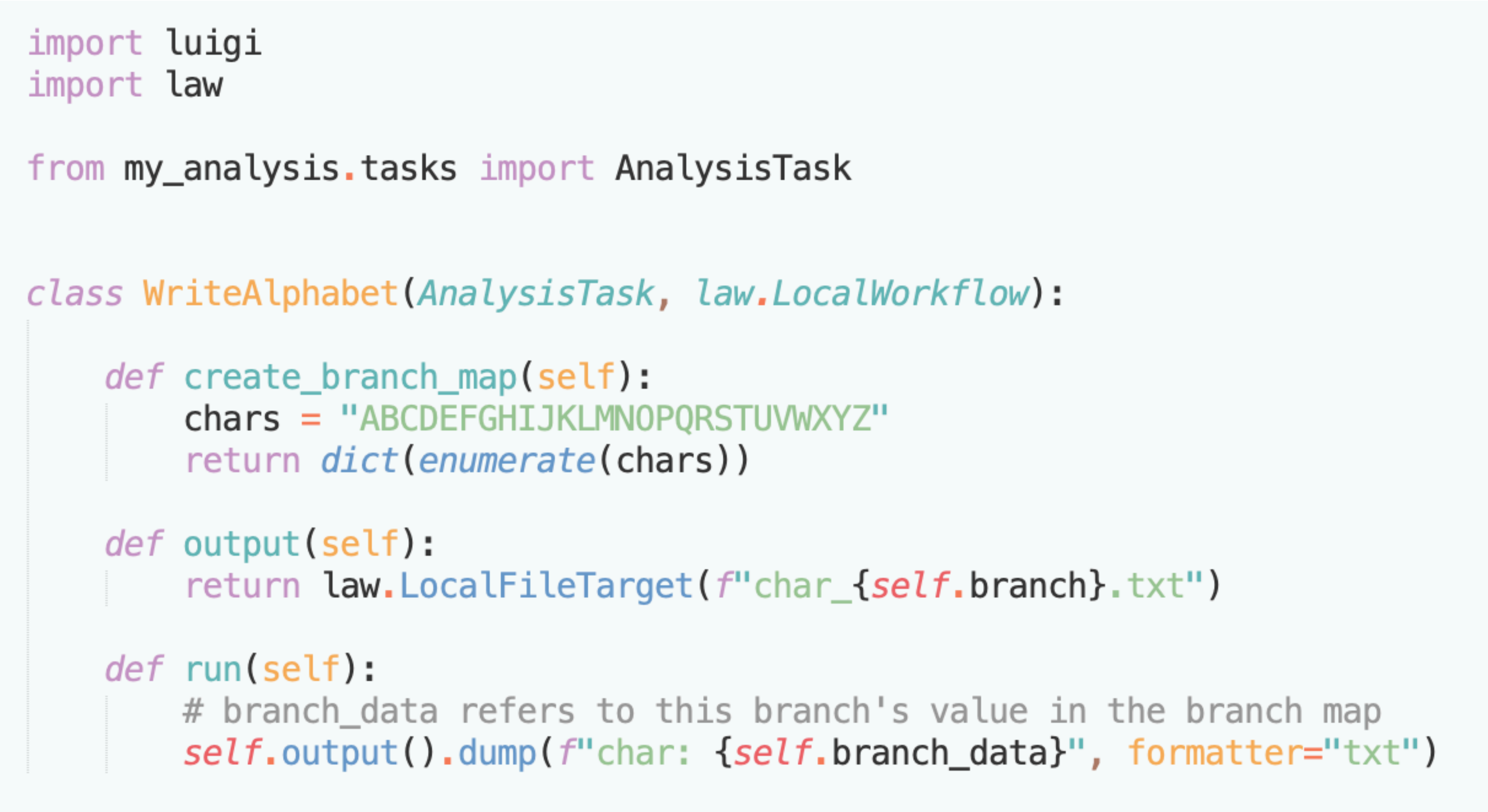


```
class Workflow(law.BaseTask):

    branch = luigi.IntParameter(default=-1)

    @property
    def is_workflow(self):
        return self.branch == -1

    def branch_tasks(self):
        return [self.req(self, branch=b) for b in self.create_branch_map()]

    def workflow_requires(self):
        """ requirements to be resolved before the workflow starts """

    def workflow_output(self):
        """ output of the workflow (usually a collection of branch outputs) """

    def workflow_run(self):
        """ run implementation """

    def create_branch_map(self):
        """ Maps branch numbers to arbitrary payloads, e.g.
            ``return {0: "file_A.txt", 1: "file_C.txt", 2: ...}``
            To be implemented by inheriting tasks.
        """
        raise NotImplementedError

    def requires(self):
        """ usual requirement definition """

    def output(self):
        """ usual output definition """

    def run(self):
        """ usual run implementation """
```

When "is_workflow" seen by luigi as requires(), output() and run()

31 Workflows: example usage

- Tasks that each write a single character into a text file
- Character assigned to them though the branch map as their "branch data"



```python
import luigi
import law

from my_analysis.tasks import AnalysisTask


class WriteAlphabet(AnalysisTask, law.LocalWorkflow):

    def create_branch_map(self):
        chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        return dict(enumerate(chars))

    def output(self):
        return law.LocalFileTarget(f"char_{self.branch}.txt")

    def run(self):
        # branch_data refers to this branch's value in the branch map
        self.output().dump(f"char: {self.branch_data}", formatter="txt")
```

Workflows: remote workflows & jobs

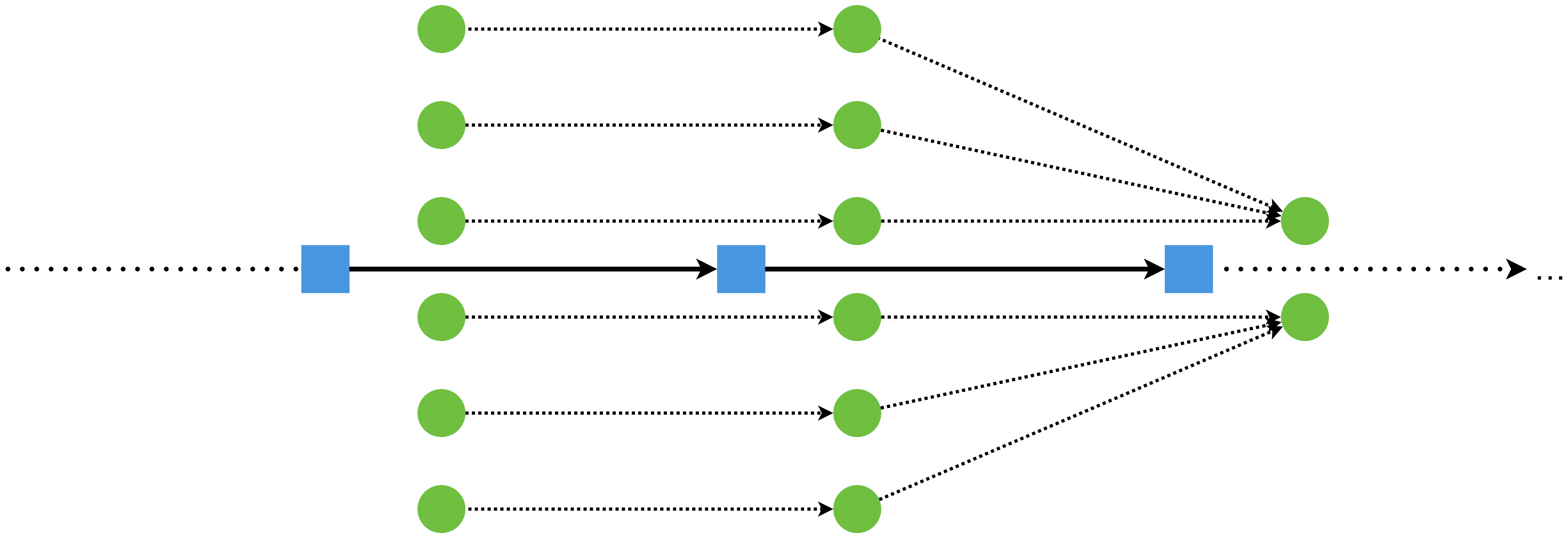
- **4 remote workflow implementations come with law**
  - htcondor, glite, lsf, arc (slurm and cms-crab in development)
  - Based on 4 generic "job manager" implementations in contrib packages

- **Job managers fully decoupled from most law functionality**
  - Simple extensibility
  - No "auto-magic" in submission files, rather minimal and configurable through tasks
  - Usable also without law

- **Most important features**
  - Job submission functionality "declared" via task class inheritance
  - Provision of software and job-specific requirements through `workflow_requires()`
  - Control over remote jobs through parameters:
    - ▻ `--start-branch`, `--end-branch`, `--branches`: granular control of which tasks to process
    - ▻ `--acceptance`, `--tolerance`: defines when a workflow is complete / failed
    - ▻ `--poll-interval`, `--walltime`: controls the job status polling interval and runtime
    - ▻ `--tasks-per-job`, `--parallel-jobs`: control of resource usage at batch systems

Workflows: high-level example

Events → Selection → Reconstruction → Merging → ...

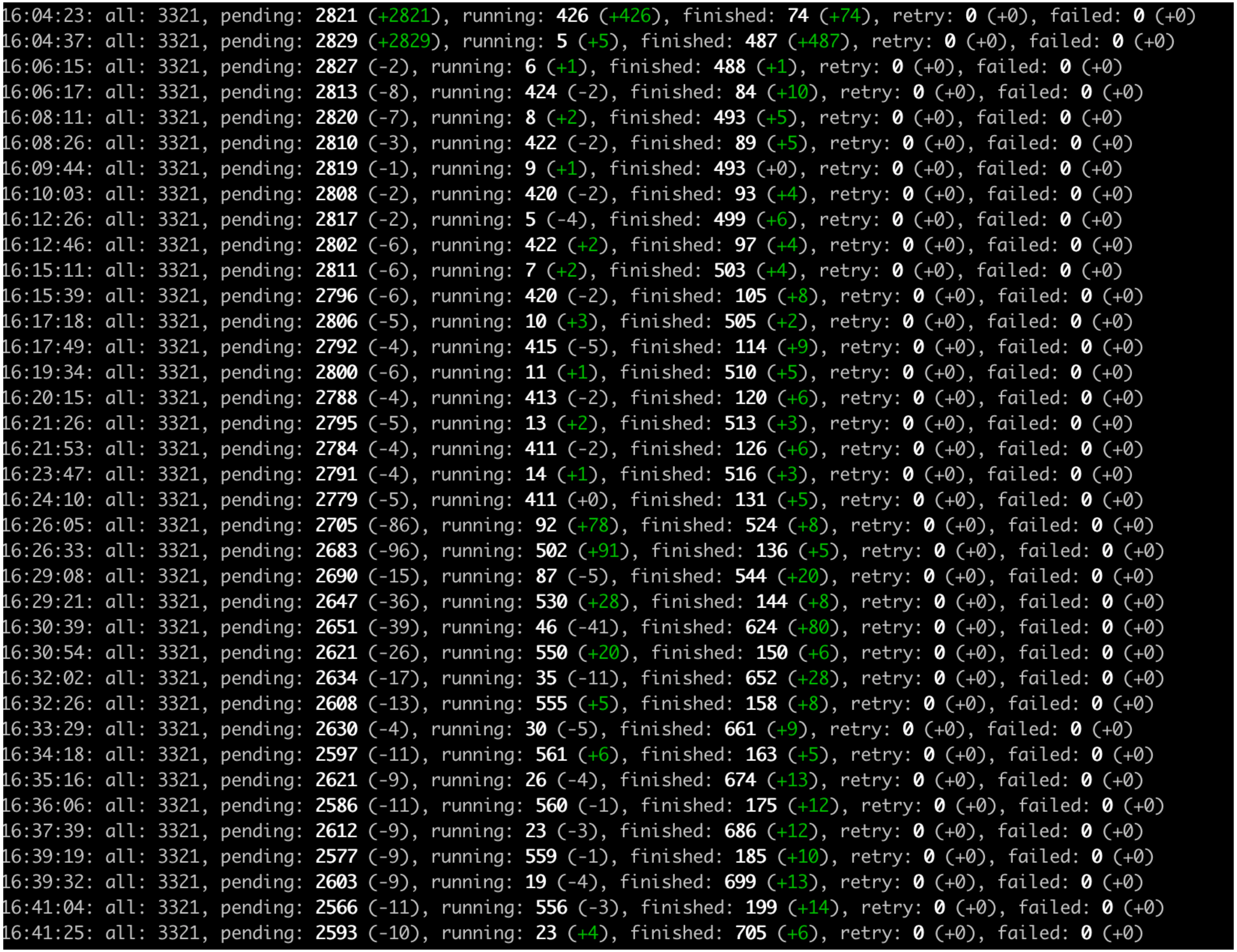


🟦 Workflow

● Branches

* Interactive slide

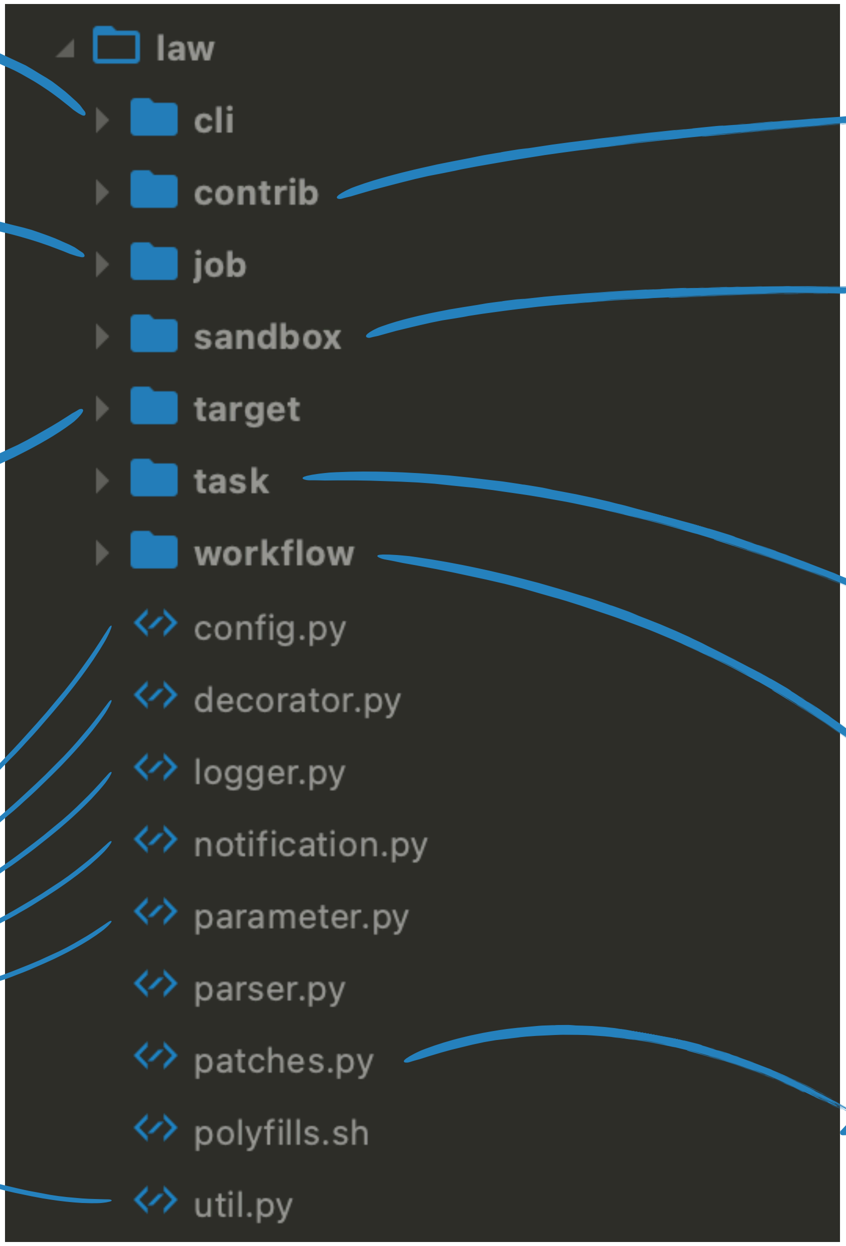
Status polling of remote jobs



```
16:04:23: all: 3321, pending: 2821 (+2821), running: 426 (+426), finished: 74 (+74), retry: 0 (+0), failed: 0 (+0)
16:04:37: all: 3321, pending: 2829 (+2829), running: 5 (+5), finished: 487 (+487), retry: 0 (+0), failed: 0 (+0)
16:06:15: all: 3321, pending: 2827 (-2), running: 6 (+1), finished: 488 (+1), retry: 0 (+0), failed: 0 (+0)
16:06:17: all: 3321, pending: 2813 (-8), running: 424 (-2), finished: 84 (+10), retry: 0 (+0), failed: 0 (+0)
16:08:11: all: 3321, pending: 2820 (-7), running: 8 (+2), finished: 493 (+5), retry: 0 (+0), failed: 0 (+0)
16:08:26: all: 3321, pending: 2810 (-3), running: 422 (-2), finished: 89 (+5), retry: 0 (+0), failed: 0 (+0)
16:09:44: all: 3321, pending: 2819 (-1), running: 9 (+1), finished: 493 (+0), retry: 0 (+0), failed: 0 (+0)
16:10:03: all: 3321, pending: 2808 (-2), running: 420 (-2), finished: 93 (+4), retry: 0 (+0), failed: 0 (+0)
16:12:26: all: 3321, pending: 2817 (-2), running: 5 (-4), finished: 499 (+6), retry: 0 (+0), failed: 0 (+0)
16:12:46: all: 3321, pending: 2802 (-6), running: 422 (+2), finished: 97 (+4), retry: 0 (+0), failed: 0 (+0)
16:15:11: all: 3321, pending: 2811 (-6), running: 7 (+2), finished: 503 (+4), retry: 0 (+0), failed: 0 (+0)
16:15:39: all: 3321, pending: 2796 (-6), running: 420 (-2), finished: 105 (+8), retry: 0 (+0), failed: 0 (+0)
16:17:18: all: 3321, pending: 2806 (-5), running: 10 (+3), finished: 505 (+2), retry: 0 (+0), failed: 0 (+0)
16:17:49: all: 3321, pending: 2792 (-4), running: 415 (-5), finished: 114 (+9), retry: 0 (+0), failed: 0 (+0)
16:19:34: all: 3321, pending: 2800 (-6), running: 11 (+1), finished: 510 (+5), retry: 0 (+0), failed: 0 (+0)
16:20:15: all: 3321, pending: 2788 (-4), running: 413 (-2), finished: 120 (+6), retry: 0 (+0), failed: 0 (+0)
16:21:26: all: 3321, pending: 2795 (-5), running: 13 (+2), finished: 513 (+3), retry: 0 (+0), failed: 0 (+0)
16:21:53: all: 3321, pending: 2784 (-4), running: 411 (-2), finished: 126 (+6), retry: 0 (+0), failed: 0 (+0)
16:23:47: all: 3321, pending: 2791 (-4), running: 14 (+1), finished: 516 (+3), retry: 0 (+0), failed: 0 (+0)
16:24:10: all: 3321, pending: 2779 (-5), running: 411 (+0), finished: 131 (+5), retry: 0 (+0), failed: 0 (+0)
16:26:05: all: 3321, pending: 2705 (-86), running: 92 (+78), finished: 524 (+8), retry: 0 (+0), failed: 0 (+0)
16:26:33: all: 3321, pending: 2683 (-96), running: 502 (+91), finished: 136 (+5), retry: 0 (+0), failed: 0 (+0)
16:29:08: all: 3321, pending: 2690 (-15), running: 87 (-5), finished: 544 (+20), retry: 0 (+0), failed: 0 (+0)
16:29:21: all: 3321, pending: 2647 (-36), running: 530 (+28), finished: 144 (+8), retry: 0 (+0), failed: 0 (+0)
16:30:39: all: 3321, pending: 2651 (-39), running: 46 (-41), finished: 624 (+80), retry: 0 (+0), failed: 0 (+0)
16:30:54: all: 3321, pending: 2621 (-26), running: 550 (+20), finished: 150 (+6), retry: 0 (+0), failed: 0 (+0)
16:32:02: all: 3321, pending: 2634 (-17), running: 35 (-11), finished: 652 (+28), retry: 0 (+0), failed: 0 (+0)
16:32:26: all: 3321, pending: 2608 (-13), running: 555 (+5), finished: 158 (+8), retry: 0 (+0), failed: 0 (+0)
16:33:29: all: 3321, pending: 2630 (-4), running: 30 (-5), finished: 661 (+9), retry: 0 (+0), failed: 0 (+0)
16:34:18: all: 3321, pending: 2597 (-11), running: 561 (+6), finished: 163 (+5), retry: 0 (+0), failed: 0 (+0)
16:35:16: all: 3321, pending: 2621 (-9), running: 26 (-4), finished: 674 (+13), retry: 0 (+0), failed: 0 (+0)
16:36:06: all: 3321, pending: 2586 (-11), running: 560 (-1), finished: 175 (+12), retry: 0 (+0), failed: 0 (+0)
16:37:39: all: 3321, pending: 2612 (-9), running: 23 (-3), finished: 686 (+12), retry: 0 (+0), failed: 0 (+0)
16:39:19: all: 3321, pending: 2577 (-9), running: 559 (-1), finished: 185 (+10), retry: 0 (+0), failed: 0 (+0)
16:39:32: all: 3321, pending: 2603 (-9), running: 19 (-4), finished: 699 (+13), retry: 0 (+0), failed: 0 (+0)
16:41:04: all: 3321, pending: 2566 (-11), running: 556 (-3), finished: 199 (+14), retry: 0 (+0), failed: 0 (+0)
16:41:25: all: 3321, pending: 2593 (-10), running: 23 (+4), finished: 705 (+6), retry: 0 (+0), failed: 0 (+0)
```

Miscellaneous

Package structure



```
law
  cli
  contrib
  job
  sandbox
  target
  task
  workflow
  config.py
  decorator.py
  logger.py
  notification.py
  parameter.py
  parser.py
  patches.py
  polyfills.sh
  util.py
```

Command-line interface

Job interface

- Job file factory interface
- Job manager interface
- Generic remote job script

Target definitions

- Generic + file interace
- Local target impl.
- Remote target interfaces

Config parsing & tools
Task decorators
Custom loggers
Notification tools (for e.g. slack/telegram)
Custom parameters
Utilities & helpers

3rd party tools

Sandboxing mechanism

- Sandbox task
- Sandbox interface
- Bash sandbox impl.

Base task definitions

Base workflow definition

- Local workflow impl.
- Remote workflow interface

Lightweight patches of luigi, e.g.:

- Disable dep. checks in sandboxes
- Colorize logs

→ Could be added directly to luigi

Package structure

Command-line interface

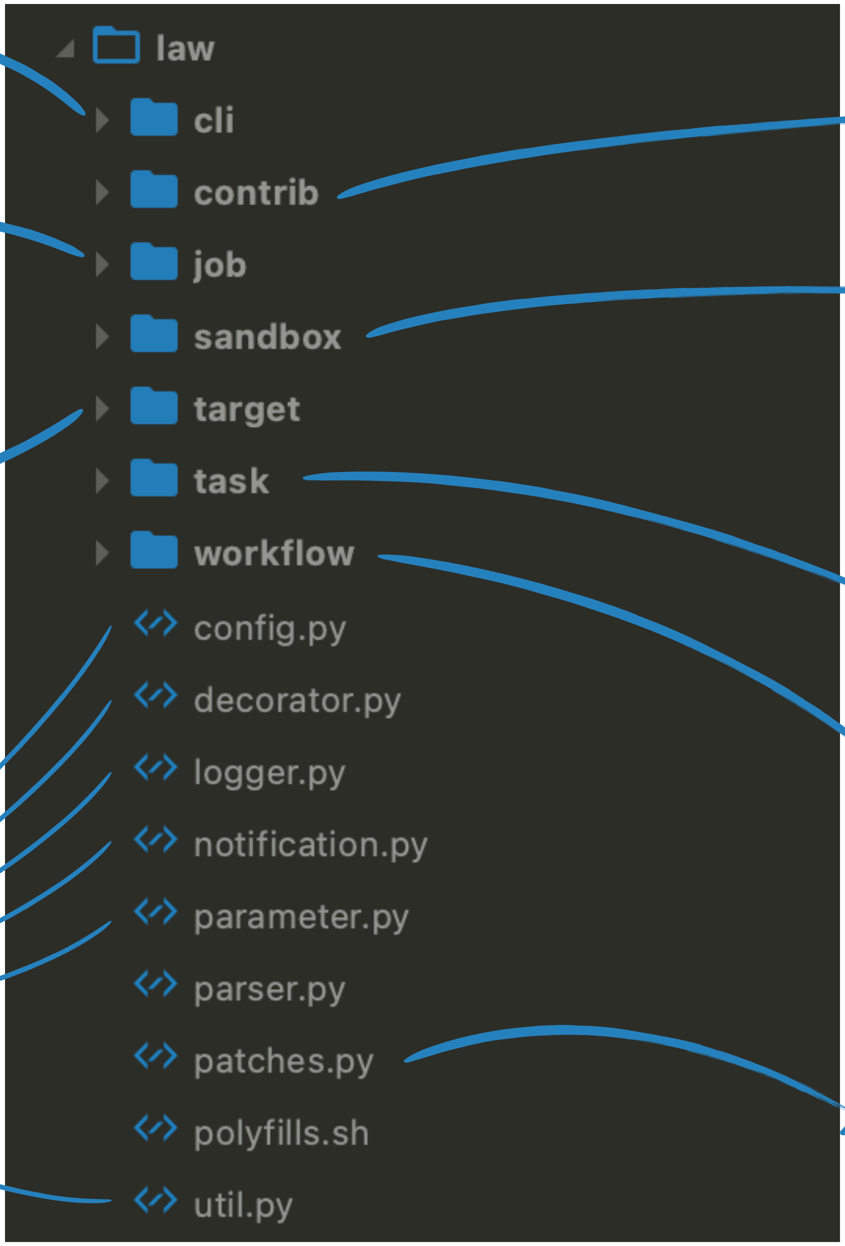
Job interface

- Job file factory interface
- Job manager interface
- Generic remote job script

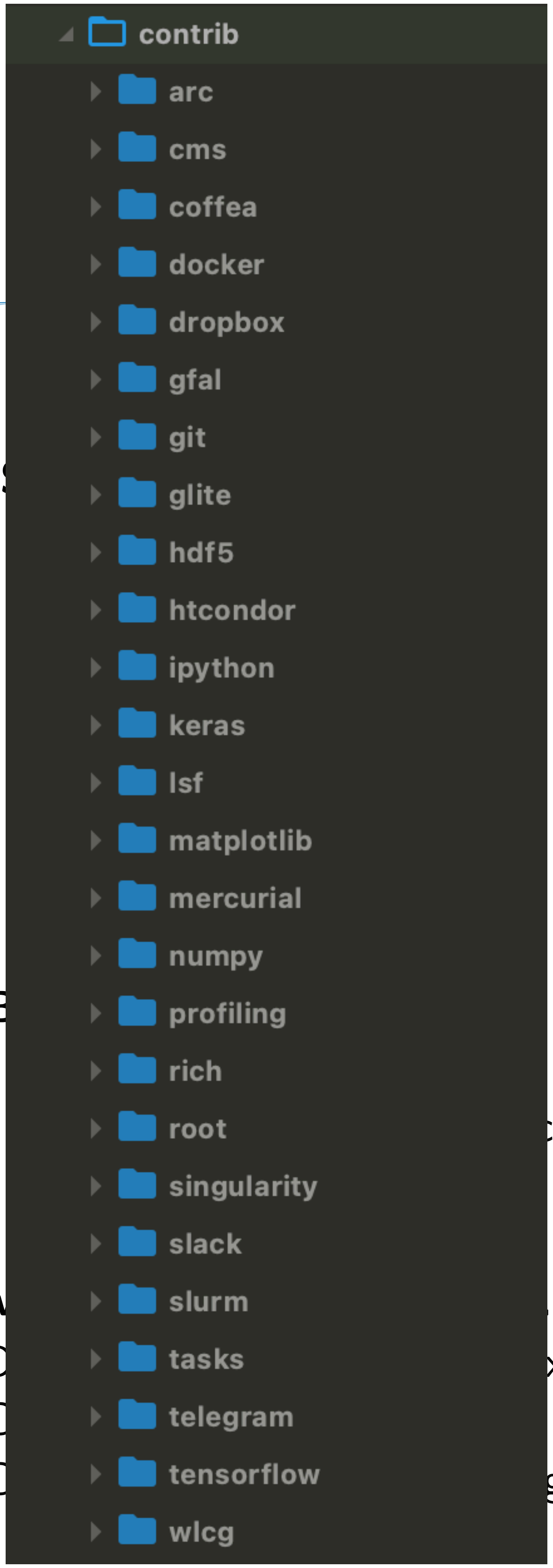
Target definitions

- Generic + file interace
- Local target impl.
- Remote target interfaces

Config parsing & tools
Task decorators
Custom loggers
Notification tools (for e.g. slack/telegram)
Custom parameters
Utilities & helpers



```
law
  cli
  contrib
  job
  sandbox
  target
  task
  workflow
  config.py
  decorator.py
  logger.py
  notification.py
  parameter.py
  parser.py
  patches.py
  polyfills.sh
  util.py
```



```
contrib
  arc
  cms
  coffea
  docker
  dropbox
  gfal
  git
  glite
  hdf5
  htcondor
  ipython
  keras
  lsf
  matplotlib
  mercurial
  numpy
  profiling
  rich
  root
  singularity
  slack
  slurm
  tasks
  telegram
  tensorflow
  wlcg
```

S

E

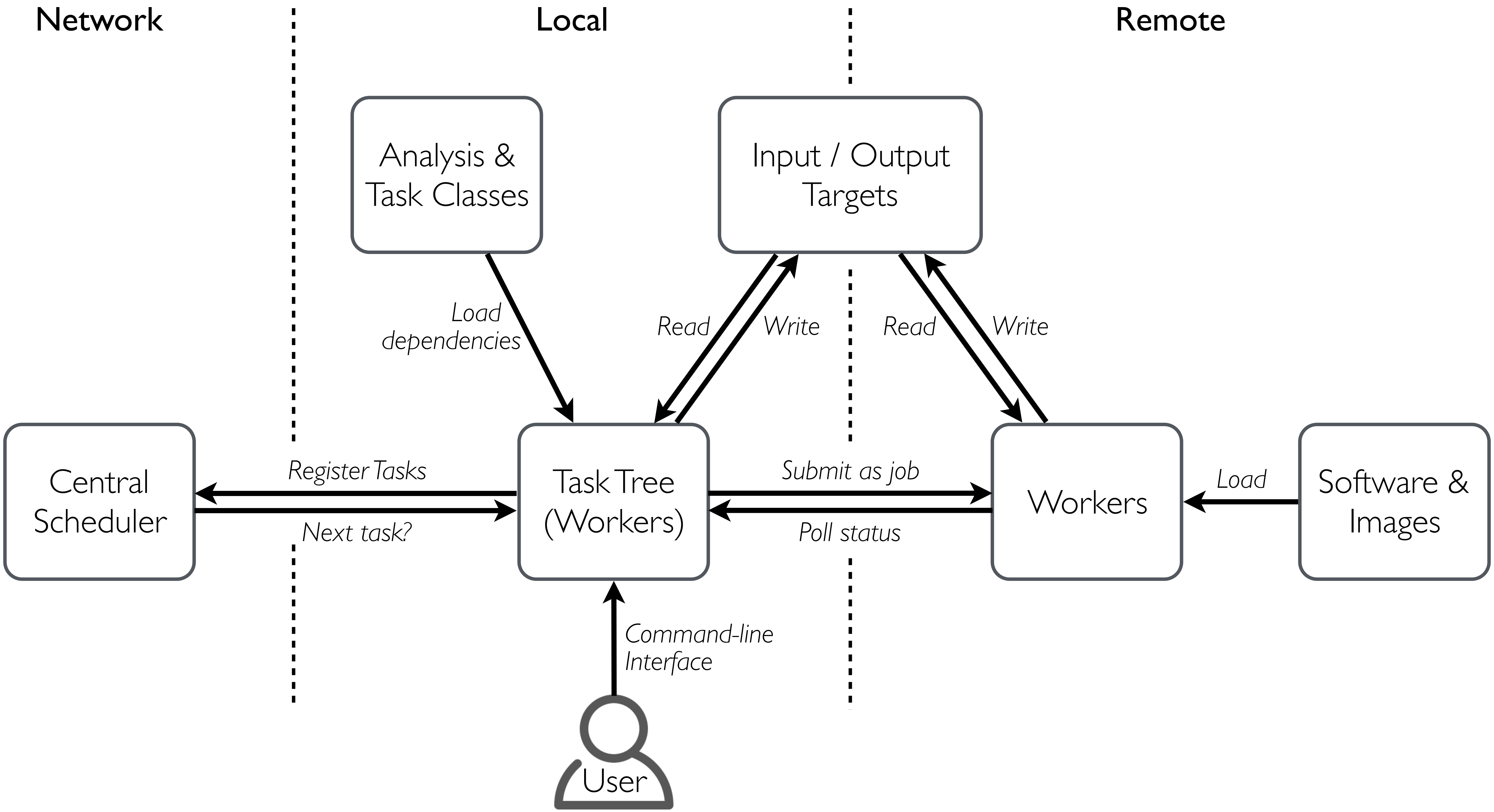
ce

Lightw :

- D xes
- C

→ C gi

luigi/law architecture



Misc. features

- **Notifications**
  - Send messages to slack / mattermost / telegram when tasks finish or crash
- **Extended configs**
  - law and luigi configs in the same file
  - Support for environment variable expansion
  - Internal section and option referencing
- **Targets**
  - load() / dump() methods for common output formats defined in contrib packages
  - Local caching of remote targets with high degree of configurability
- **TODO**
  - foo bar

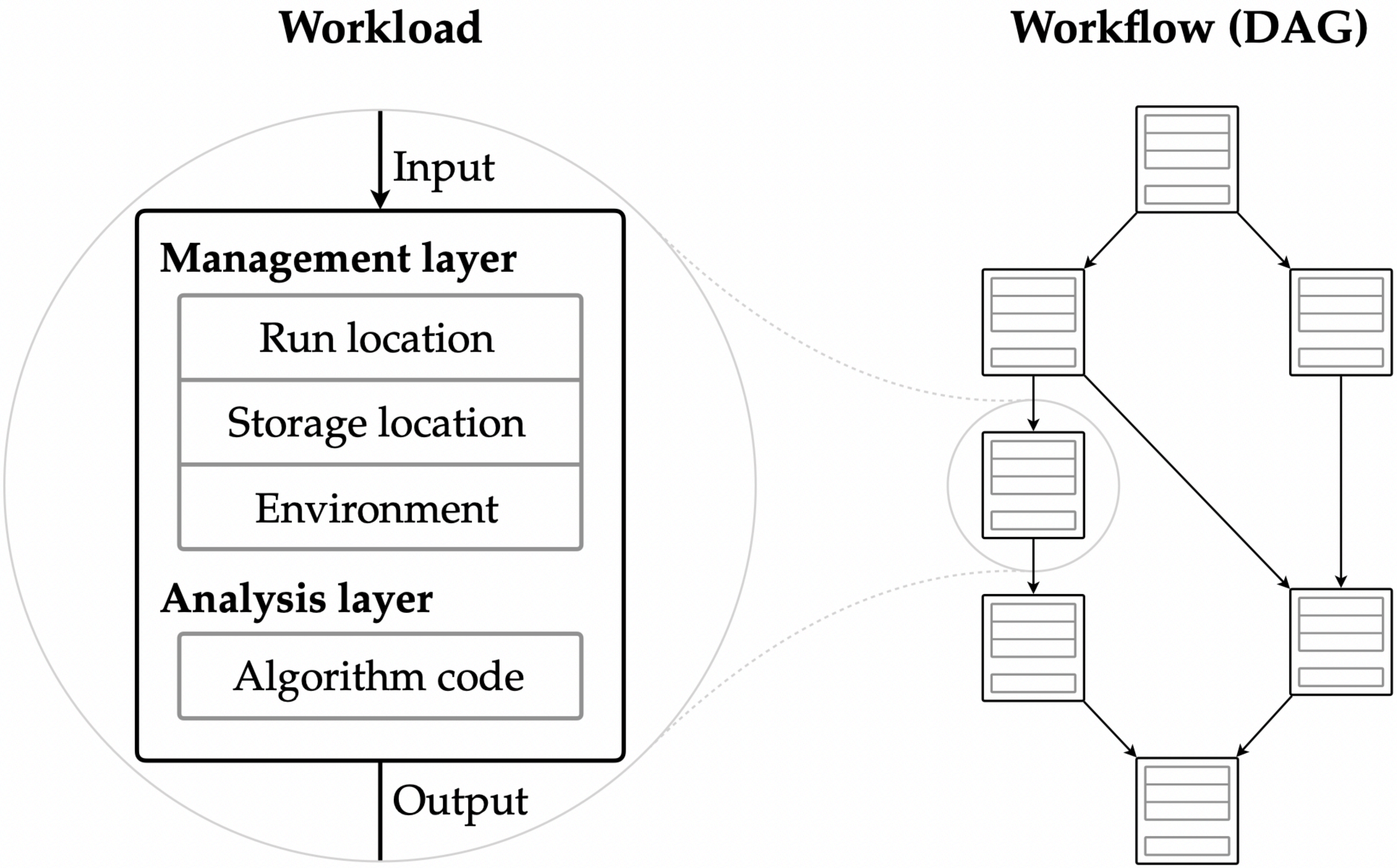
Links

- *law* - *luigi* analysis workflow
  - Repository ☞ github.com/riga/law
  - Paper ☞ arXiv:1706.00955 (CHEP16 proceedings)
  - Documentation ☞ law.readthedocs.io (in preparation)
  - Minimal example ☞ github.com/riga/law/tree/master/examples/loremipsum
  - HTCondor example ☞ github.com/riga/law/tree/master/examples/htcondor_at_cern
  - Contact ☞ Marcel Rieger

- *luigi* - Powerful Python pipelining package (by Spotify)
  - Repository ☞ github.com/spotify/luigi
  - Documentation ☞ luigi.readthedocs.io
  - "Hello world!" ☞ github.com/spotify/luigi/blob/master/examples/hello_world.py

- Technologies
  - GFAL2 ☞ dmc.web.cern.ch/projects/gfal-2/home
  - Docker ☞ docker.com
  - Singularity ☞ singularity.lbl.gov

DAG abstraction



Link (Sec. 5.1)