- PostDoc at Hamburg University (UHH) working with CMS

- **Physics projects and interests**
  - Observation of $t\bar{t}H$ production and $H \to b\bar{b}$ decays (PhD @ RWTH)
  - Search for $HH \to b\bar{b}\tau^+\tau^-$ production, $HH$ combination (research fellow @ CERN)
  - $HH + H$ combination in CMS, $HH$ combination CMS + ATLAS (PostDoc @ UHH)
  - $HH$ BSM interpretations, BSM, heavy resonances, DM, 2HDM (ongoing)
  - Differentiable limit extraction & assumption-free ML optimization (ongoing)

marcel.rieger@cern.ch
riga

- **CMS projects and roles**
  - 2020-2023: CMS ML group, coordinator for production deployment
    - Integration of TensorFlow into core software
    - Ahead-of-time compilation of ML graphs
    - Automated performance measurement (time & memory) of models
  - Since 2023: CMS CAT (Common Analysis Tools), coordinator for workflow orchestration & preservation
    - Definition of common meta data format for all CMS analyses
    - Development of tools & support for automating analyses **end-to-end**
    - Automation of NanoAOD production for users

- scinum  
  - Python package for defining numbers, subject to **one or multiple uncertainties**
    (single file, no dependencies, all Python versions)
  - Automatic gaussian error propagation (≈ eager autograd)
  - Support for arrays, rounding according to PDG rules, neat formatting, export to HEPData format, ...

```
[11]  1  Number(2.5, {
      2      "stat": 0.5j,         # relative 0.5
      3      "syst": (1.0, 1.5),   # absolute 1.0 up, 1.5 down
      4  })
```

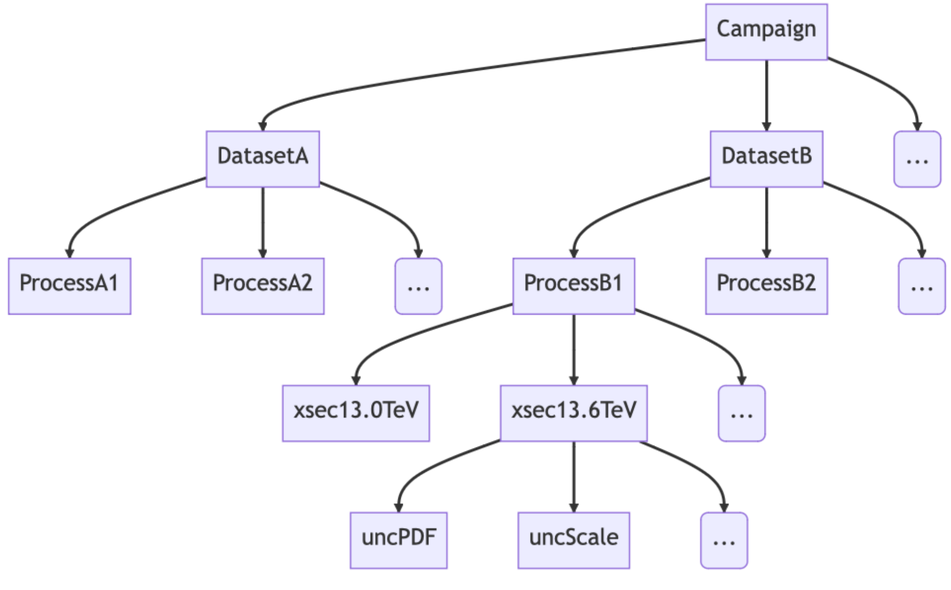
$2.5 \pm 1.25\,(\text{stat})\ ^{+1.0}_{-1.5}\,(\text{syst})$

```
[13]  1  n1 = Number(2.5, {"stat": 0.5j, "syst": (1.0, 1.5)})
      2  n2 = Number(1.8, {"stat": 0.2j, "syst": 1.0})
      3
      4  (n1 @ Correlation(stat=0)) + n2
```

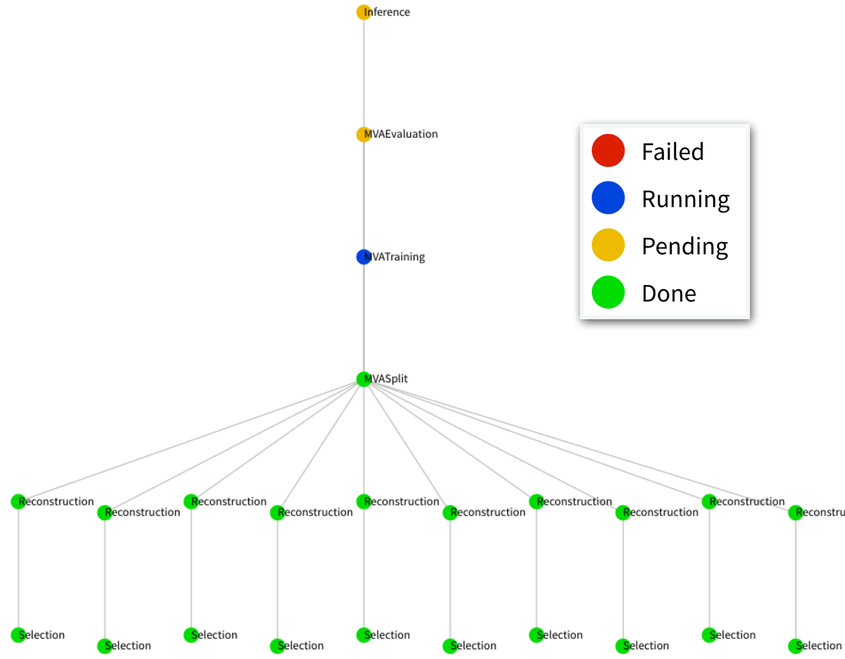
$4.3\ ^{+2.0}_{-2.5}\,(\text{syst}) \pm 1.30080744155313\,(\text{stat})$

Treat "stat" errors as uncorrelated

- order  
  - Pythonic class collection to structure meta data for LHC experiments
  - Relational structure covering
    - Production campaigns, datasets
    - Physics processes, cross sections, uncertainties
    - Variables, categories, channels, ...
  - Seed project for CMS-wide meta data format

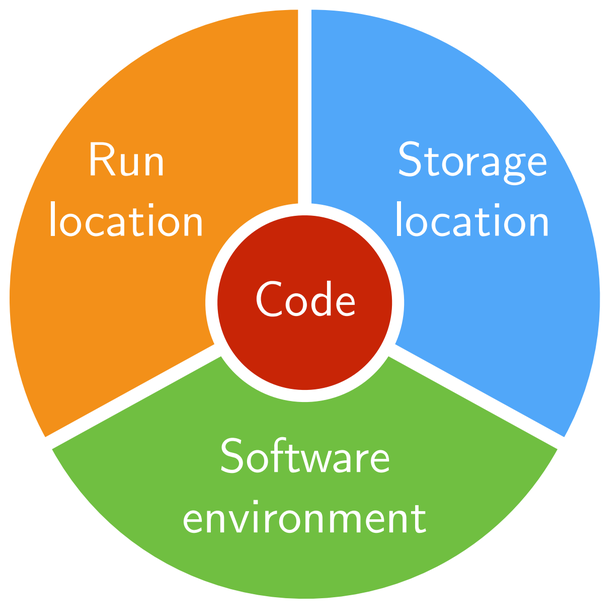


- luigi
  - Python package for creating large & scalable task pipelines
  - Development started at Spotify, now fully open source
  - Extremely lightweight & flexible core, can be explained in 5 minutes
  - Enables **macroscopic** workflows, connecting **all parts** of an analysis
    - ▻ Not *just* heavy lifting from (nano) input files to n-dim. histograms
    - ▻ No constraint on language (popen), data formats, ...



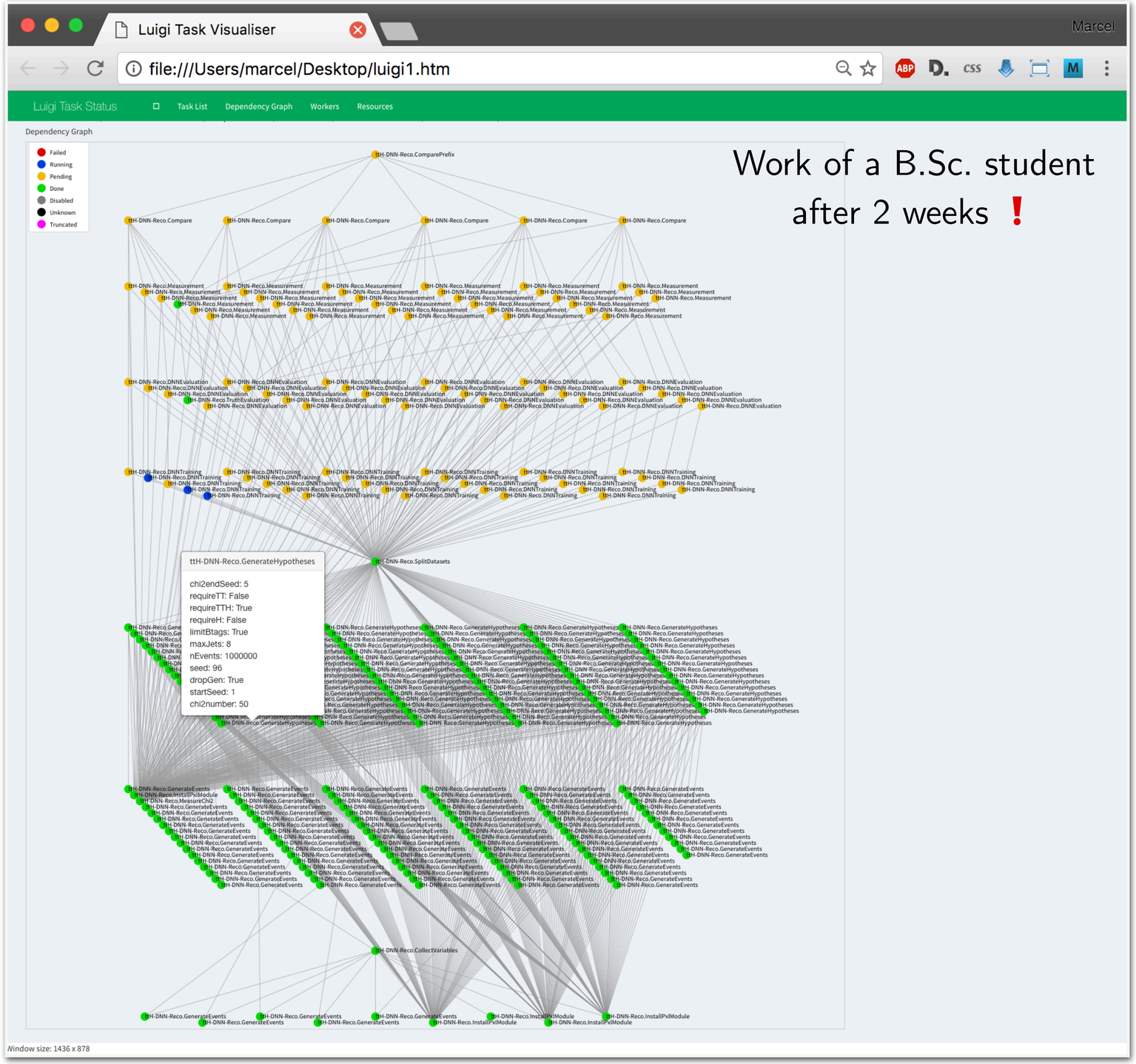

- law
  - Extension on-top of luigi, providing scale-out of HEP infrastructure
  - **Full decoupling** of
    - ▻ **run locations** (local, various batch systems, WLCG)
    - ▻ **storage locations** (local, all WLCG protocols, cloud)
    - ▻ **environments** (subshells, docker/singularity, venv, conda)
  - Everything executable by a **single command**
  - Experiment-agnostic, O(120-150) users, most used analysis workflow system at CMS

! Provides a **daily working environment**,
not just a tool to automate an analysis after the fact



# Macroscopic task graph (1)





- **CLI**

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

  - Full auto-completion of tasks and parameters

- **Scripting**

  - Mix task completeness checks, job execution & input/output retrieval with custom scripts
  - Easy interface to existing tasks for prototyping

```python
from analysis.tasks import Selection
import akward as ak

# create the task and ensure it's complete
task = Selection(dataset="ttH_bb", version="v3", shift="nominal")
task.law_run()

# read the selected events (a .parquet file)
events = task.output().load(formatter="awkward")

# get the number of jets per event
n_jets = ak.num(events.Jet, axis=1)
print(n_jets)
```

- **Notebooks**

launch binder

```
In [5]: %law run ShowFrequencies --print-status -1

print task status with max_depth -1 and target_depth 0

0 > ShowFrequencies(slow=False)
└─1 > MergeCounts(slow=False)
        LocalFileTarget(fs=local_fs, path=$DATA_PATH/chars_merged.json)
          existent
    ├─2 > CountChars(file_index=1, slow=False)
    │       LocalFileTarget(fs=local_fs, path=$DATA_PATH/chars_1.json)
    │         existent
    │   └─3 > FetchLoremIpsum(file_index=1, slow=False)
    │           LocalFileTarget(fs=local_fs, path=$DATA_PATH/loremipsum_1.txt)
    │             existent
```

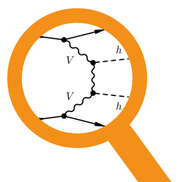
- **Making the Analysis Grand Challenge (AGC) even more realistic**
  - Machine learning training
  - k-fold cross validation, creates realistic points in the analysis where per-event information is needed
  - More complex statistical model
  - Plotting with systematic shifts, pre-fit and post-fit
  - ...

# Backup

# Orchestration software stack







workflow engine
(originally by Spotify)

layer for HEP & scale-out
(experiment independent)

"framework"
(experiment independent*)

analysis

* soon

law & luigi

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

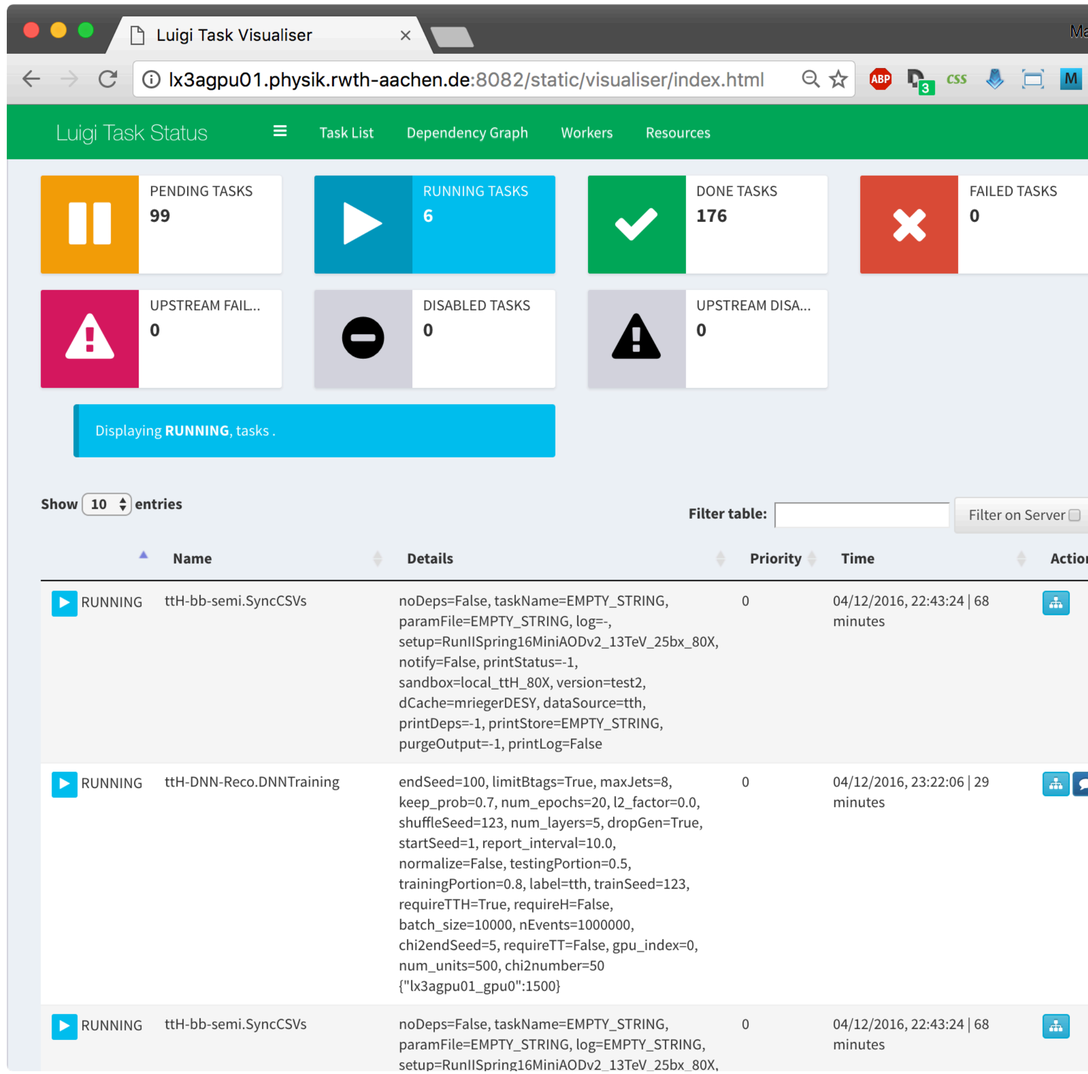
## Building blocks

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

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

github.com/spotify/luigi

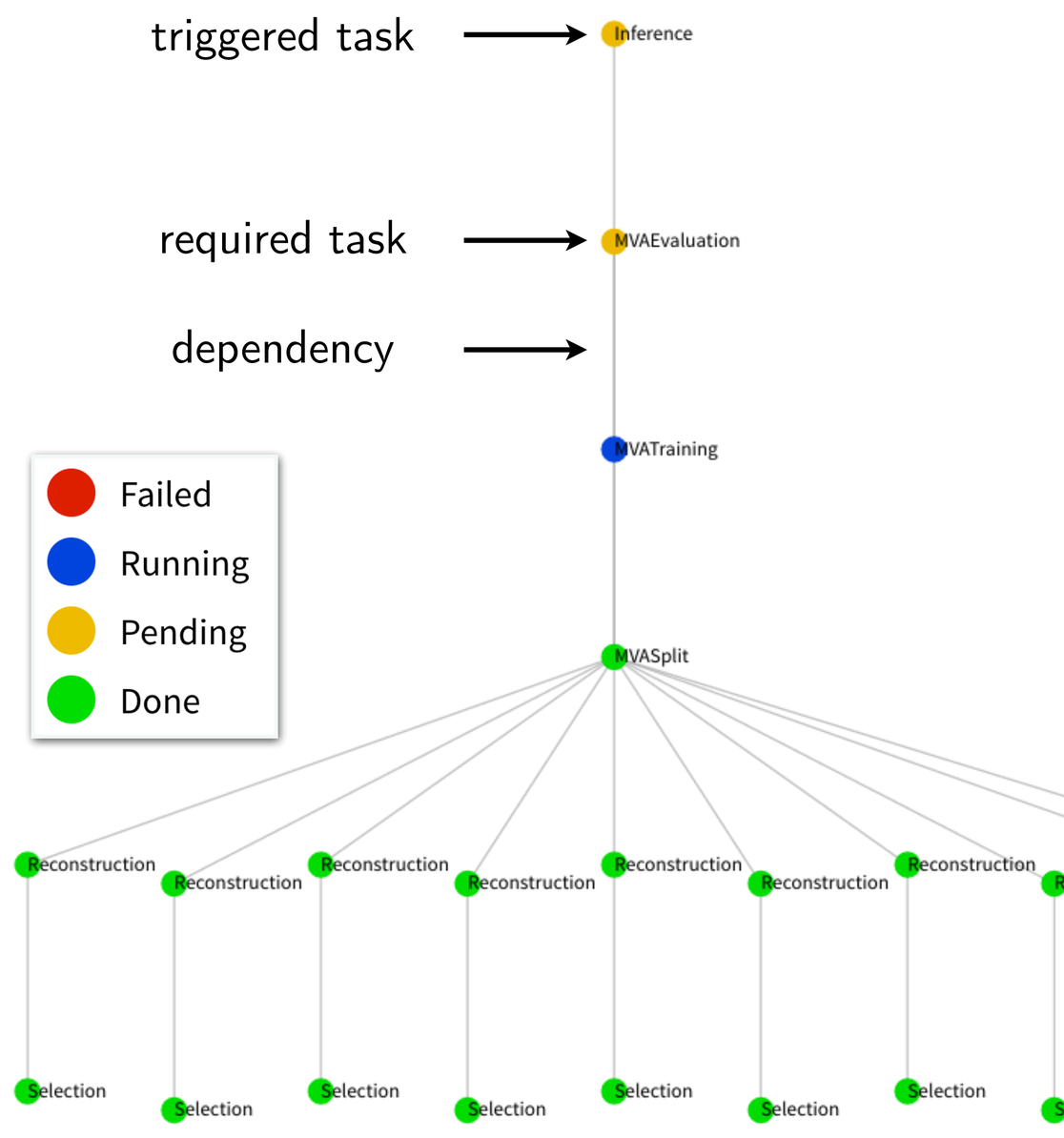
Watch 493 | Unstar 15.2k | Fork 2.3k



- 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`

```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
```

```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

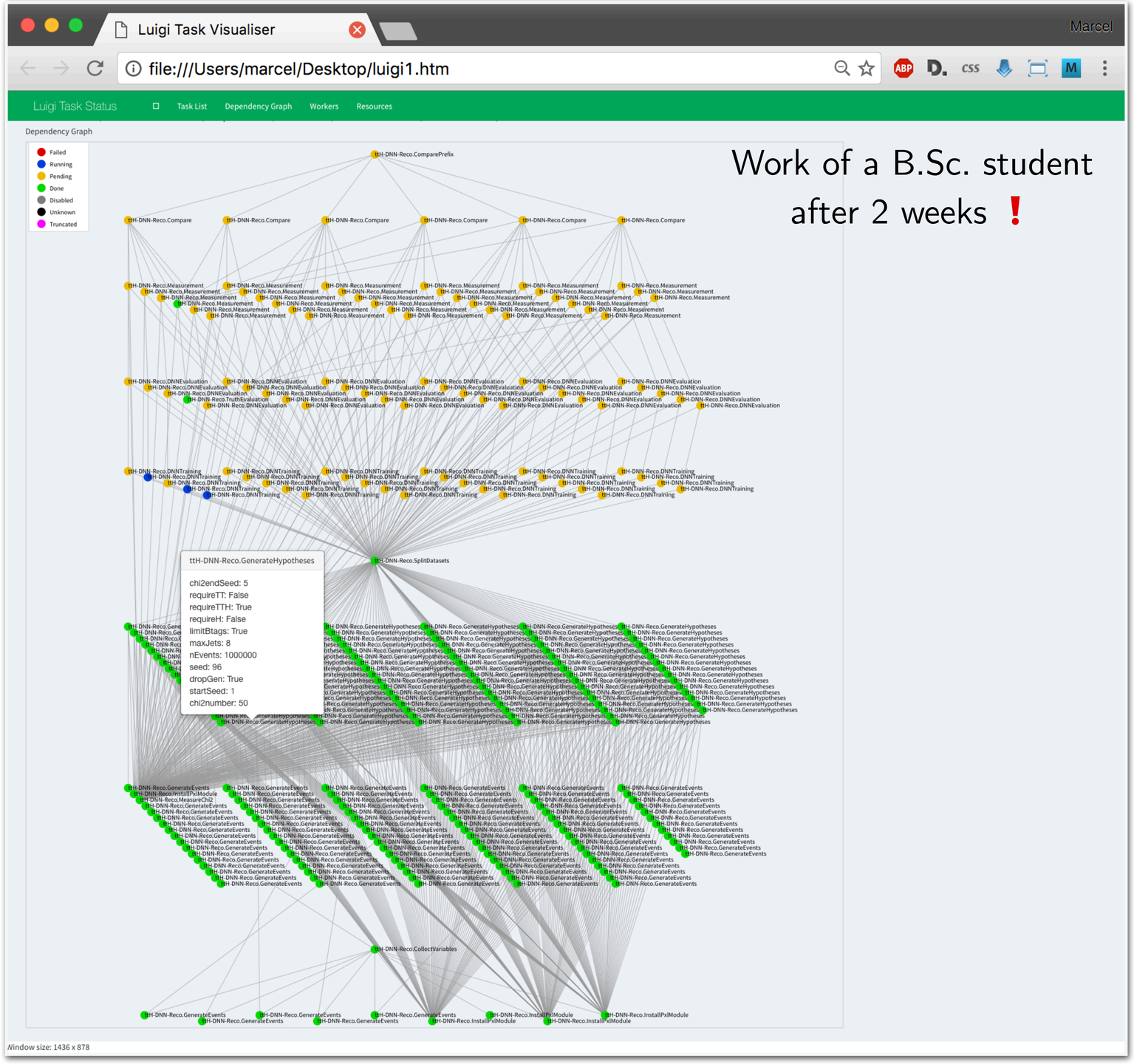
luigi's local file target:
- path: string
- exists(): bool
- remove()
- open(): fd
- ...

Encoding parameters into output target path

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

# Example dependency trees





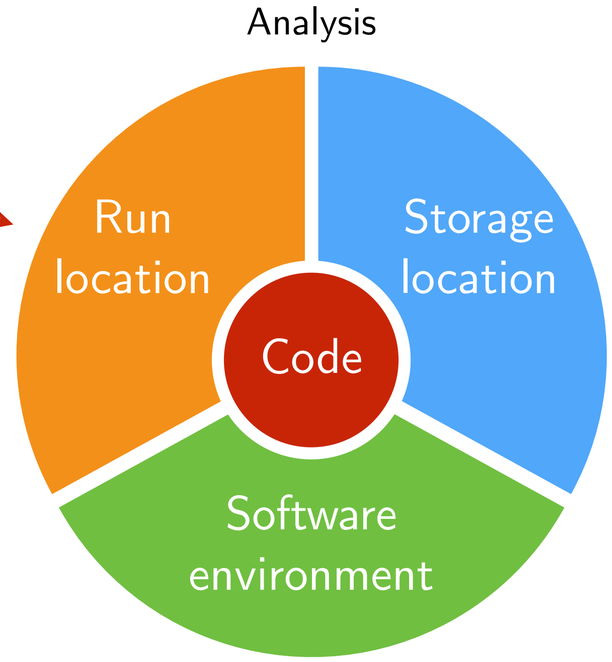

Work of a B.Sc. student after 2 weeks !

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

- Software design follows 3 primary goals:

  1. Experiment-agnostic core (in fact, not even related to physics)
  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

law
luigi analysis workflow




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

  → Not a *framework*

- **Most used** workflow system for analyses in CMS
  - O(20) analyses, O(60-80) people
  - Used at all german CMS sites
  - Central CMS groups, e.g. HIG, TAU, BTV

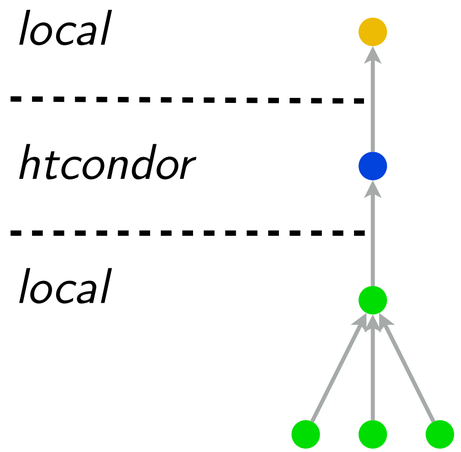
law
luigi analysis workflow

## 1. Job submission

- Idea: submission built into tasks, **no need to write extra code**
- Currently supported job systems: HTCondor, LSF, gLite, ARC, Slurm (+ CRAB ~next month)
- Mandatory features such as automatic resubmission, flexible task ↔ job matching,

  job files fully configurable at submission time, internal job staging when queues are saturated, ...
- 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 >
```



## 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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, fsspec integration easily possible)
- Mandatory features: automatic retries, **local caching** (backup), 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

## 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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, fsspec integration easily possible)
- Mandatory features: automatic retries, **local caching** (backup), configurable protocols, round-robin, ...

Conveniently reading remote files

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

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

## 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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, fsspec integration easily possible)
- Mandatory features: automatic retries, **local caching** (backup), configurable protocols, round-robin, ...

Conveniently reading remote files

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

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

## 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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, fsspec integration easily possible)
- Mandatory features: automatic retries, **local caching** (backup), 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()
```

## 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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ API **identical** to local targets
  - ! Actual remote interface **interchangeable** (GFAL2 is just a good default, fsspec integration easily possible)
- Mandatory features: automatic retries, **local caching** (backup), configurable protocols, round-robin, ...

Conveniently reading remote files

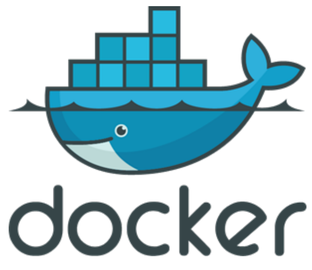
```python
# multiple other "formatters" available
target = law.WLCGFileTarget("/model.pb", fs="wlcg_fs")

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

## 3. Environment sandboxing

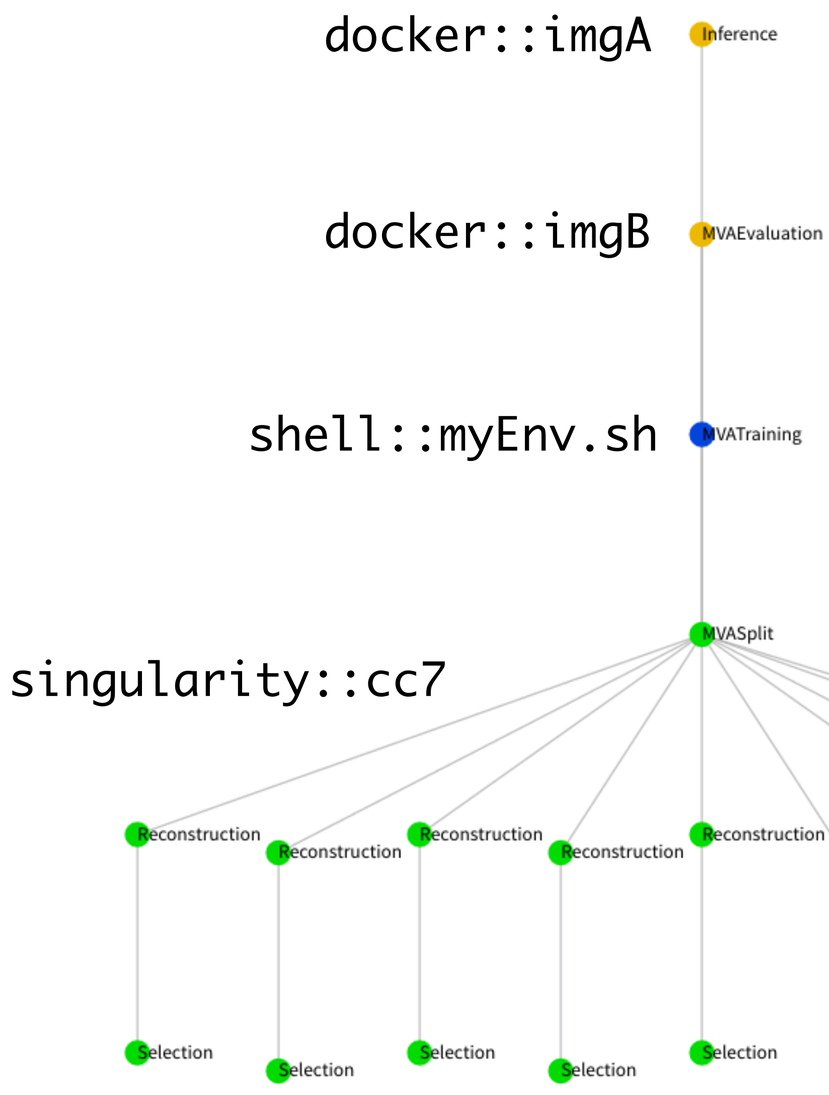
- Diverging software requirements between typical workloads is a great feature / challenge / problem

- Introduce sandboxing:
  - ▻ Run entire task in **different environment**

- Existing sandbox implementations:
  - ▻ Sub-shell with init file
  - ▻ Virtual envs
  - ▻ Docker images
  - ▻ Singularity images


law
luigi analysis workflow



```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"
        ...
```

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

Example ☞

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

```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
```

```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
```

```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
```

```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
```