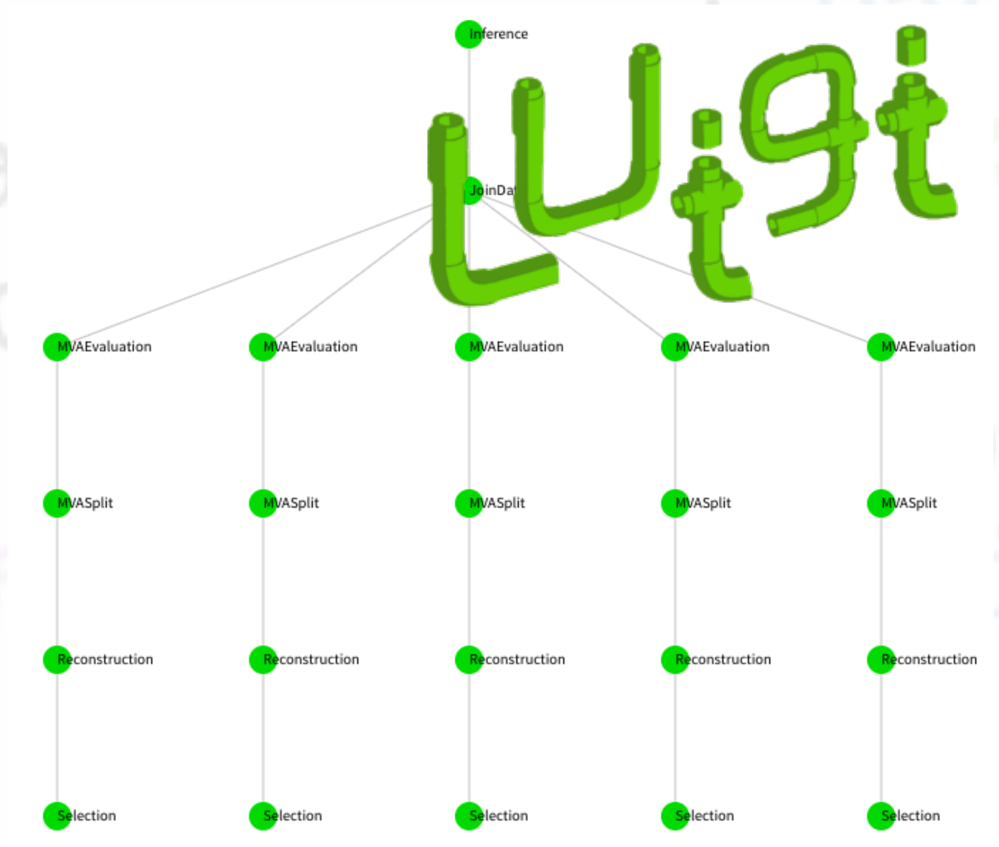


law
luigi analysis workflow

SPONSORED BY THE

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

CLUSTER OF EXCELLENCE
QUANTUM UNIVERSE

FSP CMS
Erforschung von
Universum und Materie

Federal Ministry
of Education
and Research

luigi analysis workflow

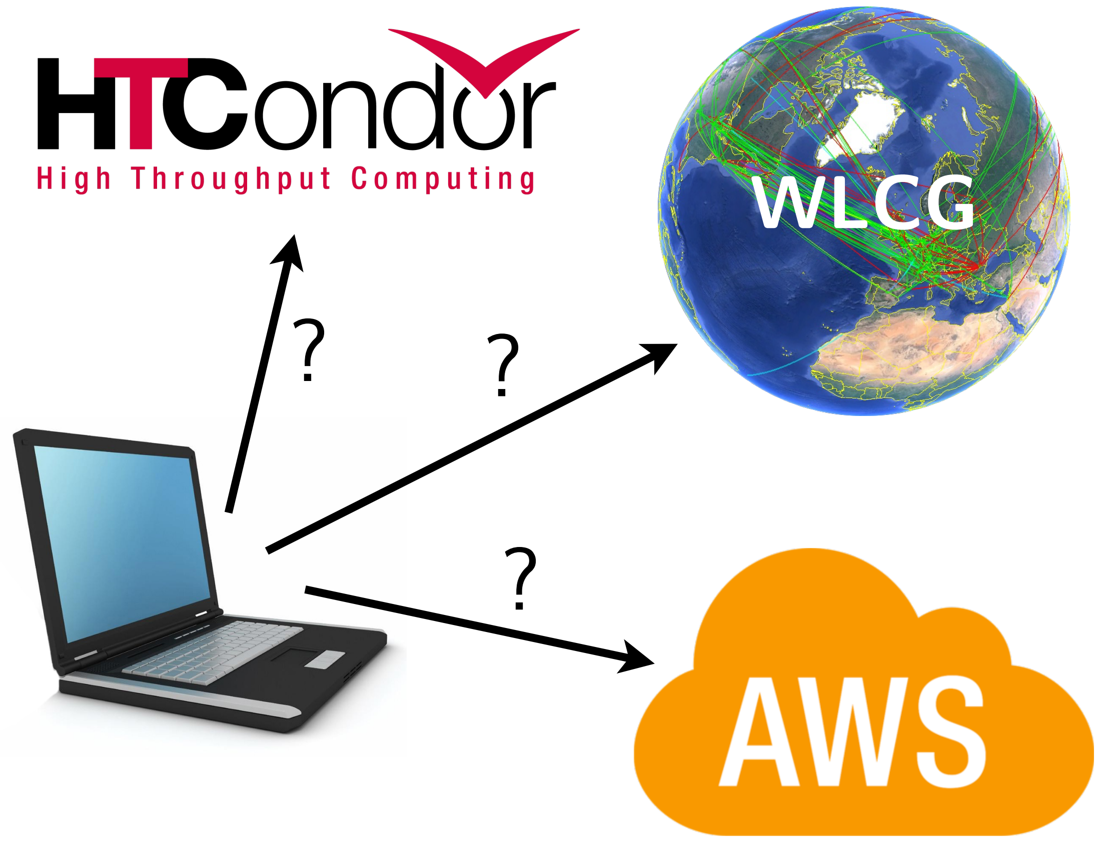
Marcel Rieger

Workshop on Workflow Languages

4.4.2024

Thoughts on analyses ...

- **Questions**
  - *Portability*
    - ▻ Does the analysis depend on **where it runs** or **where it stores data**?
    - ▻ It should **not**
  - *Reproducibility*
    - ▻ A Student / PostDoc is leaving soon ... can someone else run the analysis?
    - ▻ Often **not** the case



- **Familiar situations**
  - "We couldn't produce updates, our local cluster is down for maintenance."
  - "We need to run things again, we forgot to change some paths in script XYZ."
  - "No updates from my side, I had to do job sitting the whole week ..."

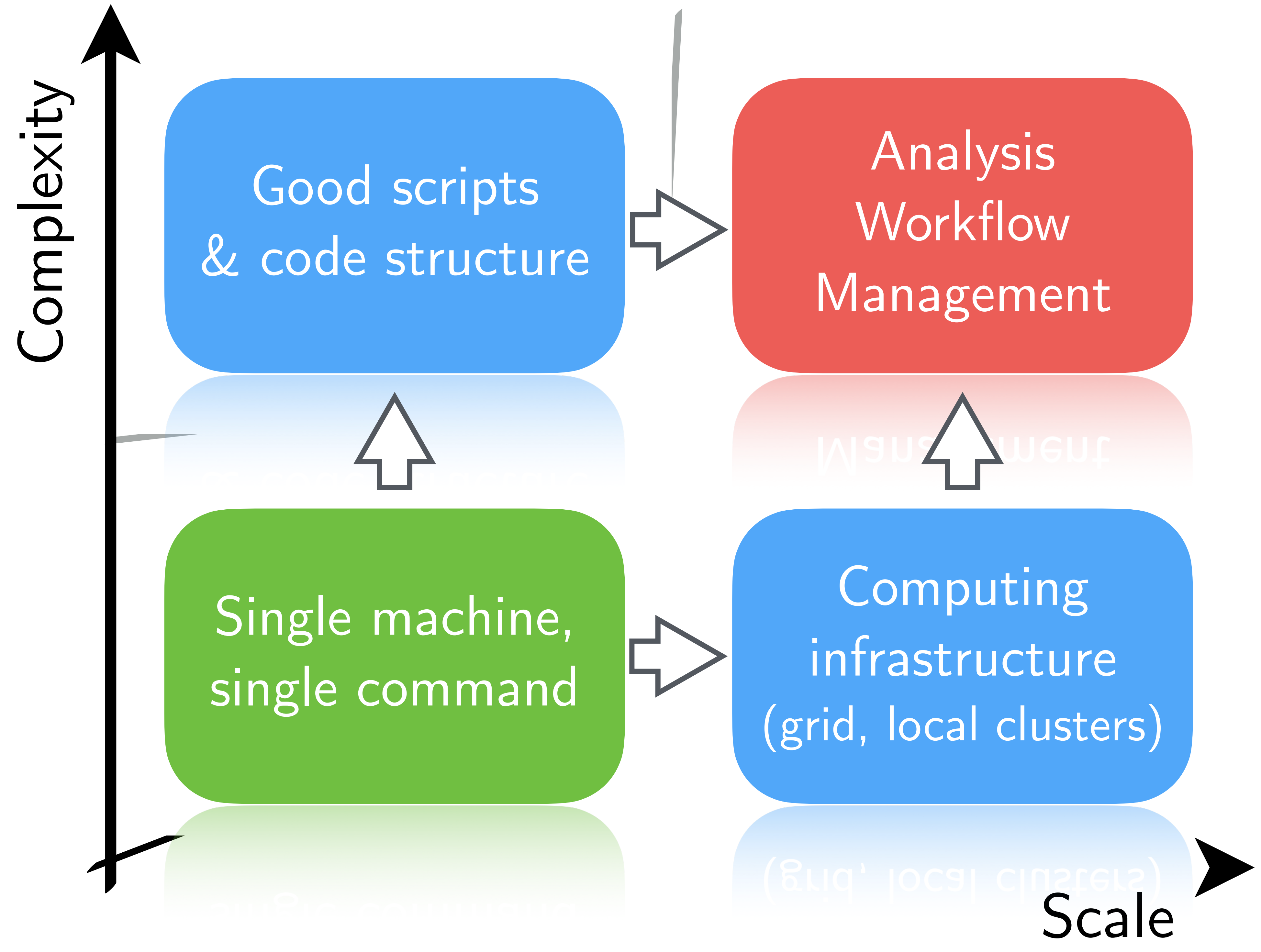
- From personal analysis experience
  - ⅔ of time required for technicalities, ⅓ left for physics

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

Landscape of HEP analyses

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

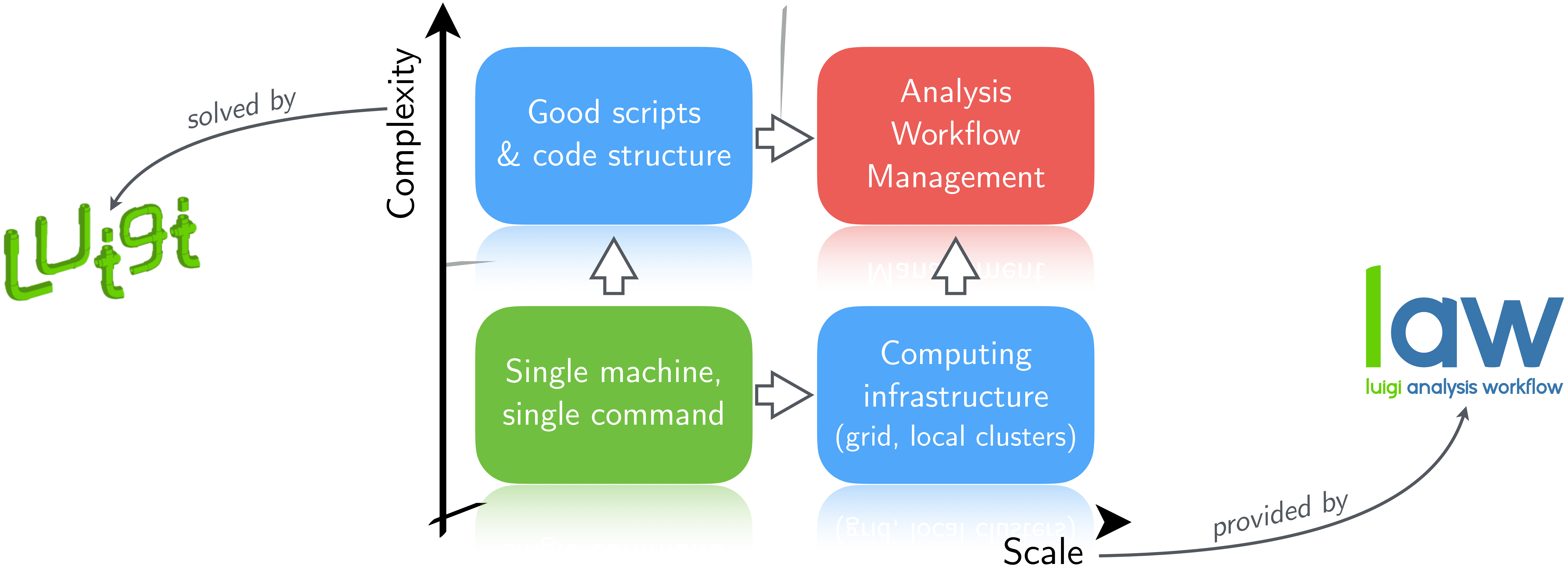
  → **Time-consuming** & **error-prone**


Complexity
Good scripts & code structure
Analysis Workflow Management
Single machine, single command
Computing infrastructure (grid, local clusters)
Scale

- **Workflow management must ...**
  - **provide full automation** → Execution through **a single command**
  - **cover all possible use cases** → Examples on next slides

Landscape of HEP analyses

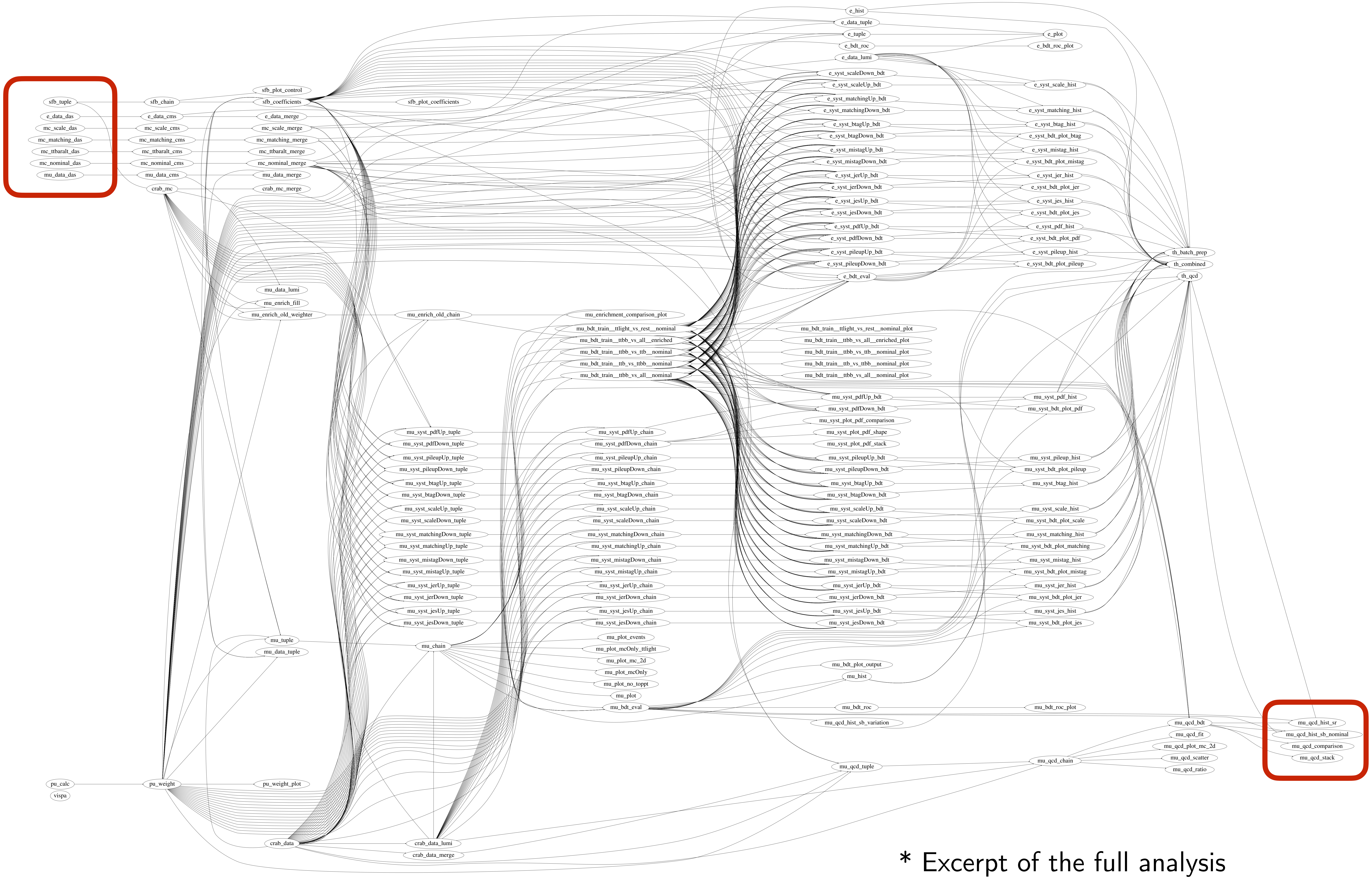
- Most analyses are both **large and complex**
  - Structure & requirements between workloads mostly undocumented
  - Manual execution & steering of jobs, bookkeeping of data across storage elements, different data revisions, ...
  - → **Time-consuming** & **error-prone**


solved by
luigi
Complexity
Good scripts & code structure
Analysis Workflow Management
Single machine, single command
Computing infrastructure (grid, local clusters)
Scale
provided by
law
luigi analysis workflow

- **Workflow management must ...**
  - **provide full automation** → Execution through **a single command**
  - **cover all possible use cases** → Examples on next slides

Example 1: ttbb measurement visualization

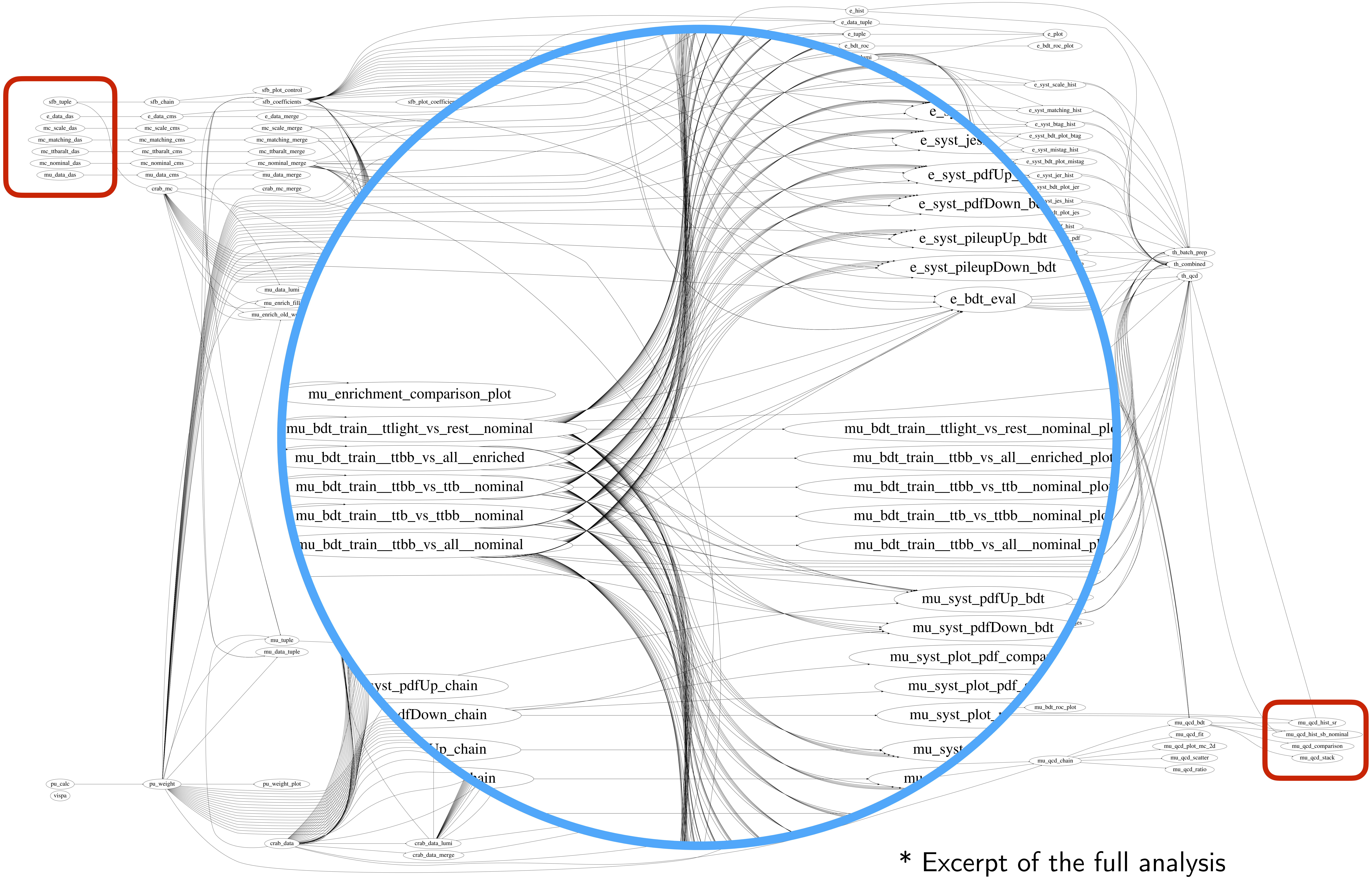
Entry points



Results

* Excerpt of the full analysis

Example 1: ttbb measurement visualization



Entry points

Results

* Excerpt of the full analysis

Example 2: Analysis Grand Challenge (with ML)

Data access

Selection

Separate events into subsamples and calculate training feature set

Model fitting and hyperparameter optimization

Select best models for inference

Entry point

Columnar data → Event selection → Obtain BDT Input Features and Labels → Training → Model Storage

Intermediate result

Dependency

Modularity & interfaces

Declarative configuration

Entry point

Columnar data → Event selection, systematic uncertainties → Obtain BDT Input Features → Machine Learning Inference → Histograms → Statistical model building → Workspace → Statistical analysis → Result, diagnostics

Result

Data access

Selection & systematic uncertainties

Inference as a service

Post-processing, e.g. smoothing

From workspaces to likelihoods

Fit results and diagnostics

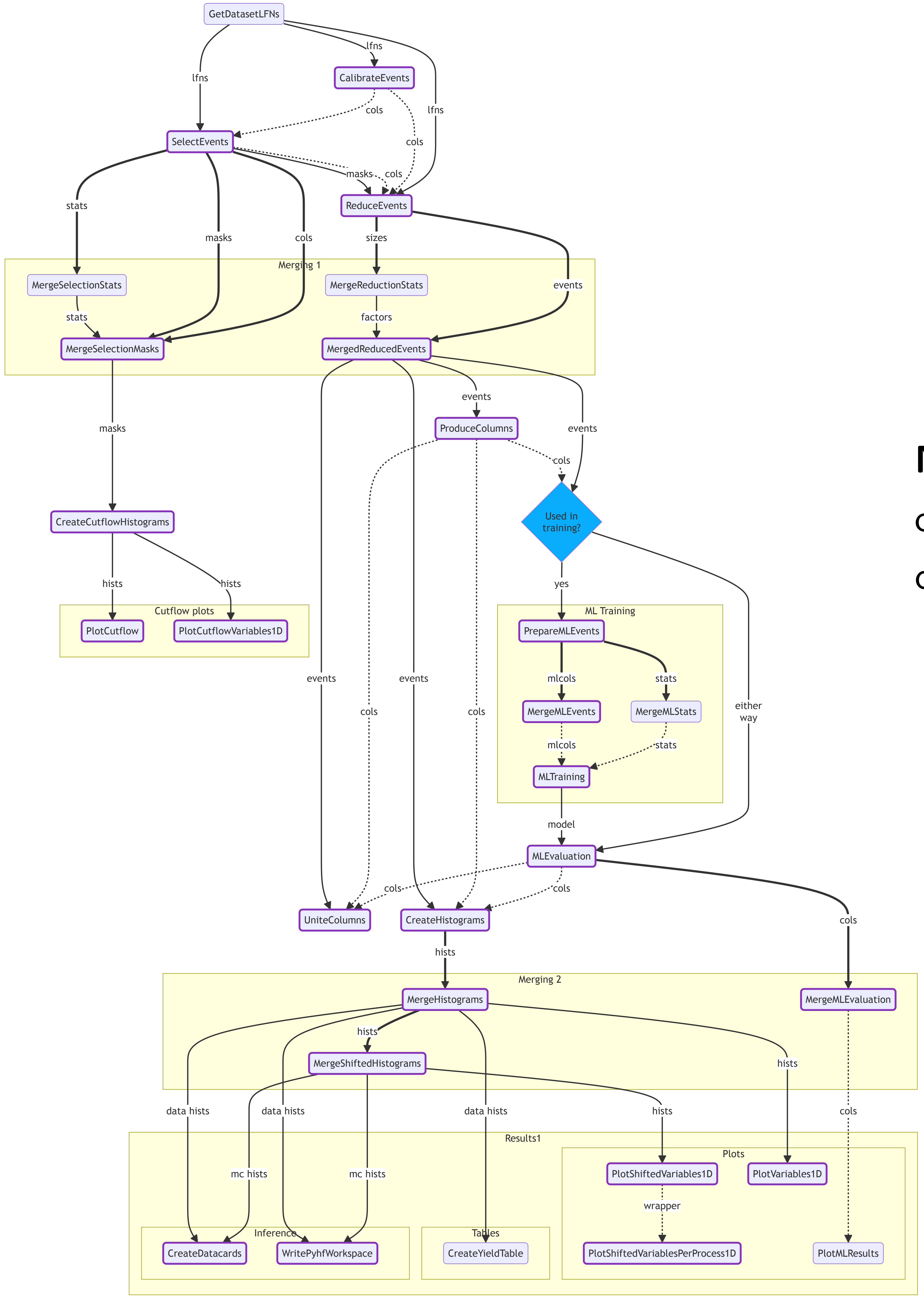
Reusability and preservation

Example 3: Fully orchestrated LHC Run 2 + 3 analysis with columnflow

Event processing

workflow

Plots & inference



Note: this is a simplified, stylized view of the full workflow, which can easily consist of $\mathcal{O}(1\mathrm{M})$ particular *workloads*

Source

Luigi

Luigi

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

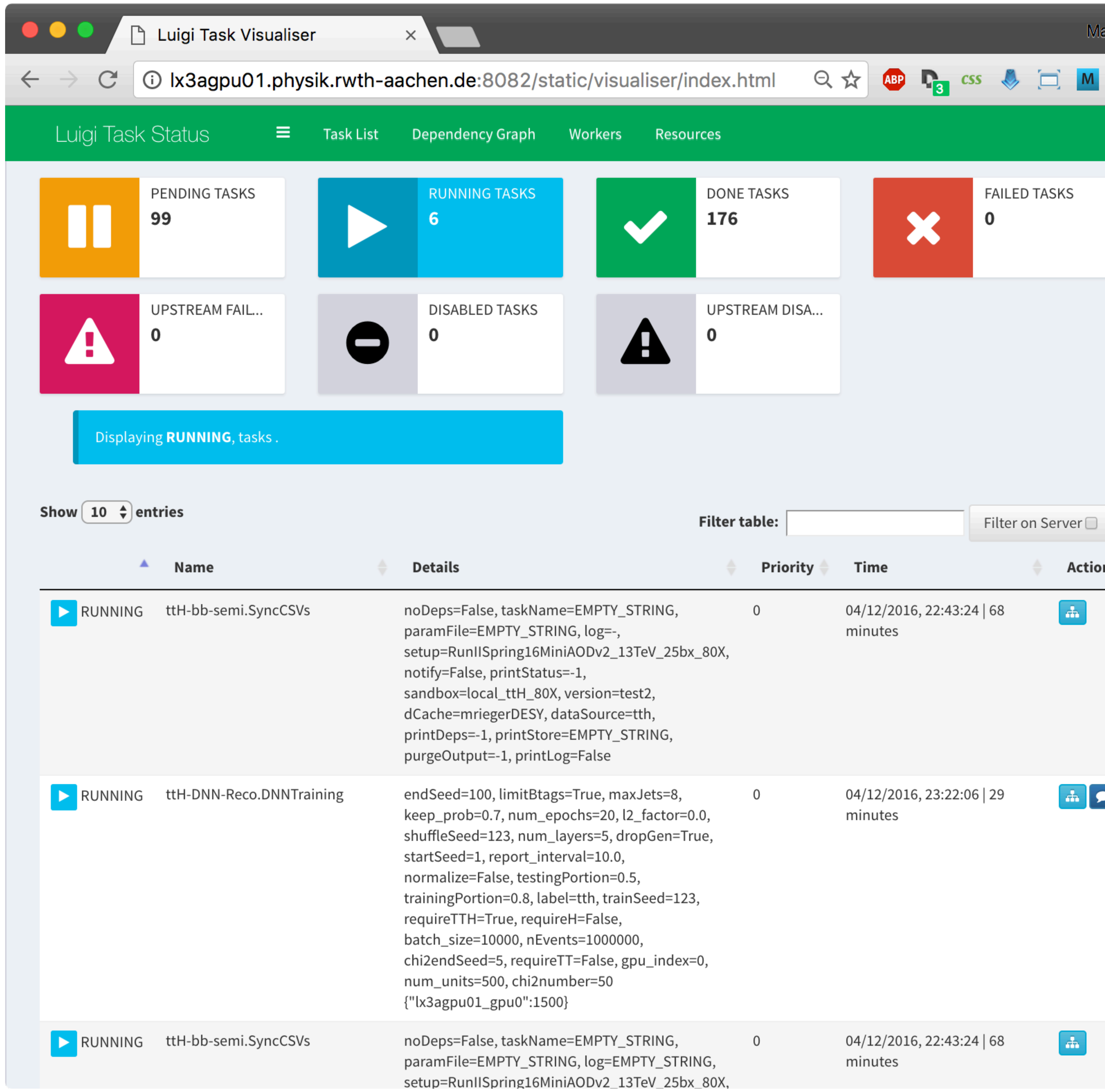
Building blocks

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, …
- Great documentation 📖

github.com/spotify/luigi

Watch 493 | Unstar 15.2k | Fork 2.3k



Luigi Task Visualiser

lx3agpu01.physik.rwth-aachen.de:8082/static/visualiser/index.html

Luigi Task Status ≡ Task List Dependency Graph Workers Resources

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

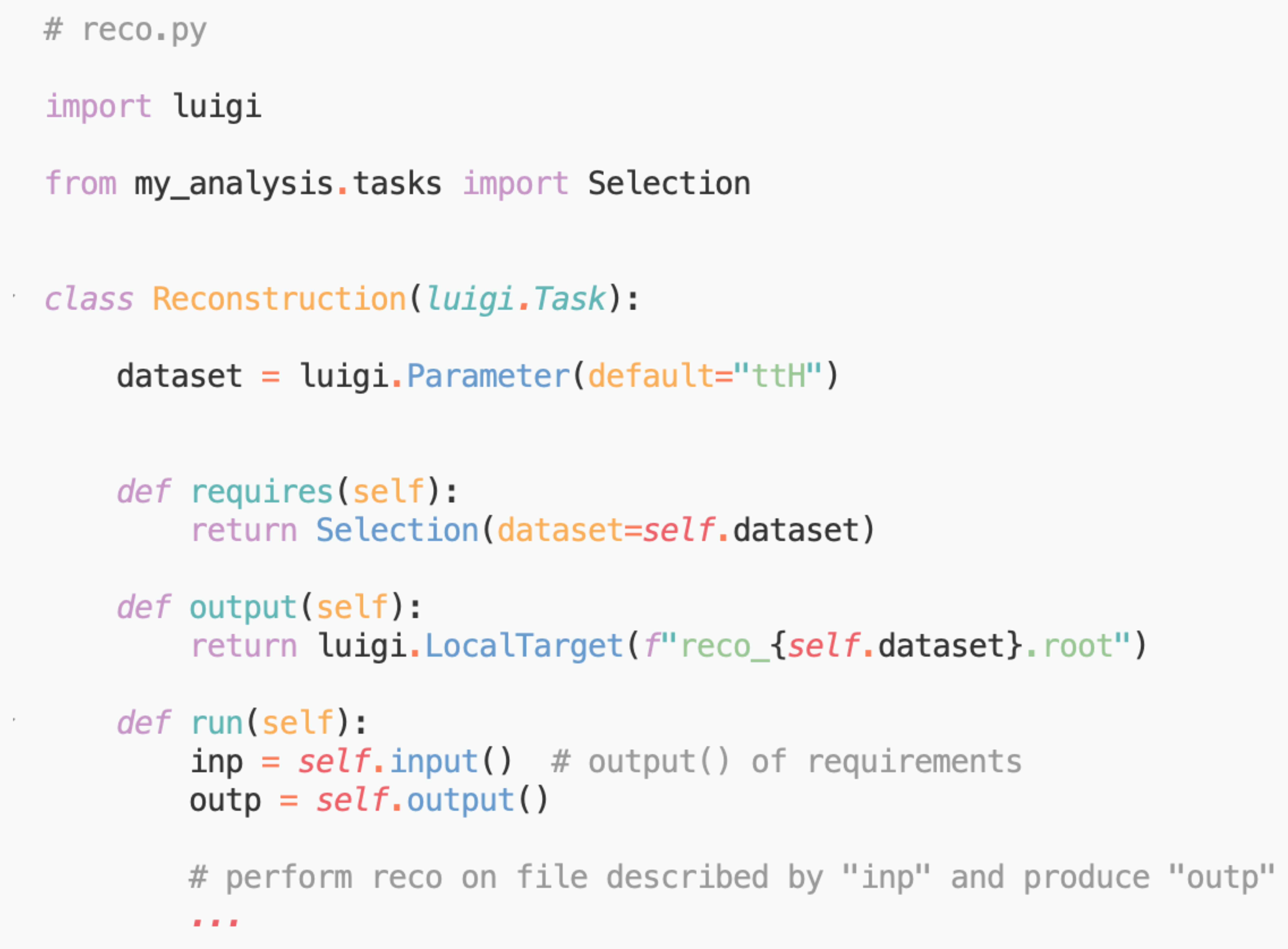
Displaying **RUNNING**, tasks .

Show 10 **entries**

Filter table: Filter on Server □

	Name	Details	Priority	Time	Actio
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	04/12/2016, 22:43:24 | 68 minutes	
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	04/12/2016, 23:22:06 | 29 minutes	
RUNNING	ttH-bb-semi.SyncCSVs	noDeps=False, taskName=EMPTY_STRING, paramFile=EMPTY_STRING, log=EMPTY_STRING, setup=RunIISpring16MiniAODv2_13TeV_25bx_80X,	0	04/12/2016, 22:43:24 | 68 minutes	

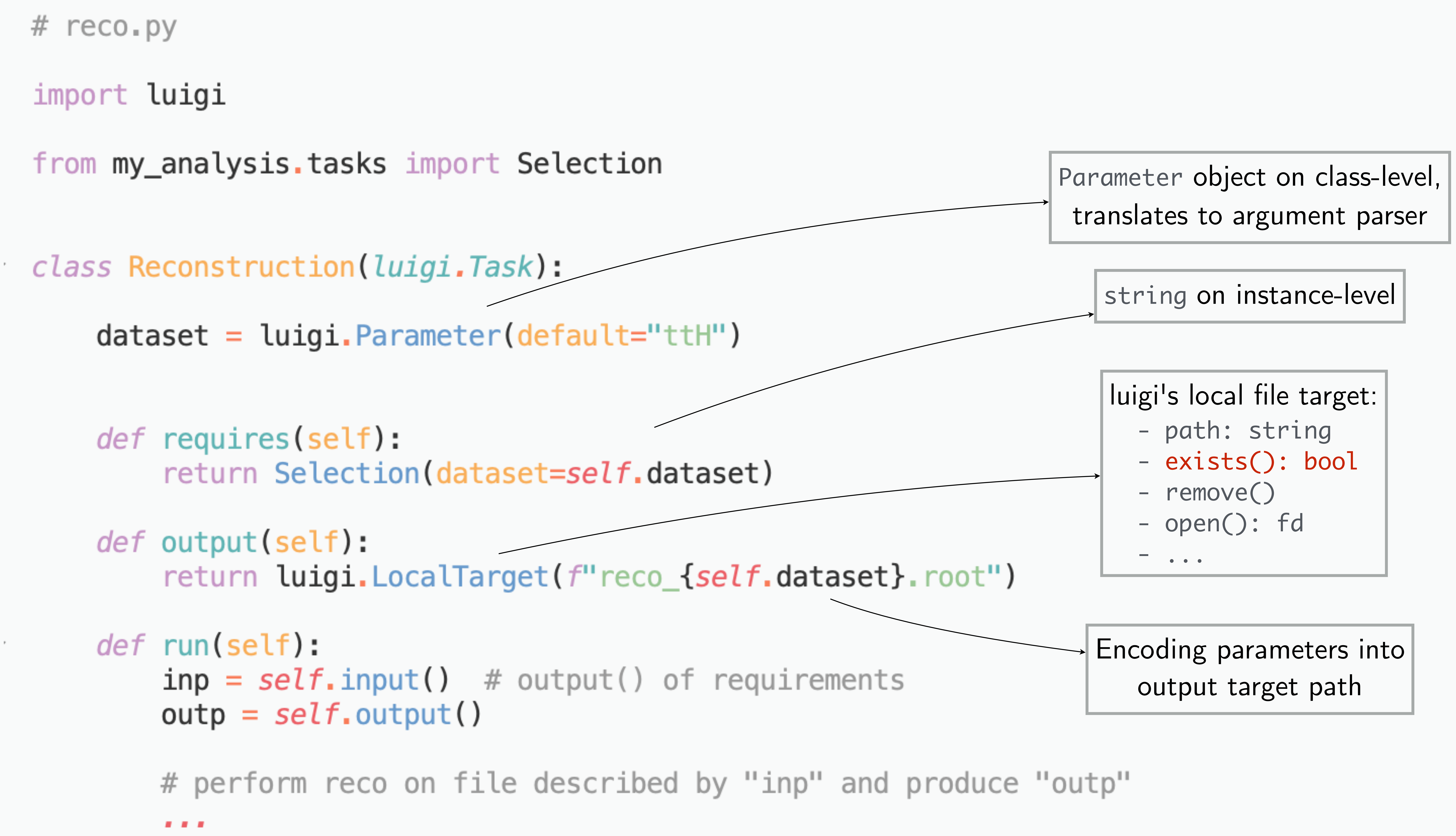
Luigi in a nutshell



```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
```

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, translates to argument parser

`string` on instance-level

luigi's local file target:

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

Encoding parameters into output target path

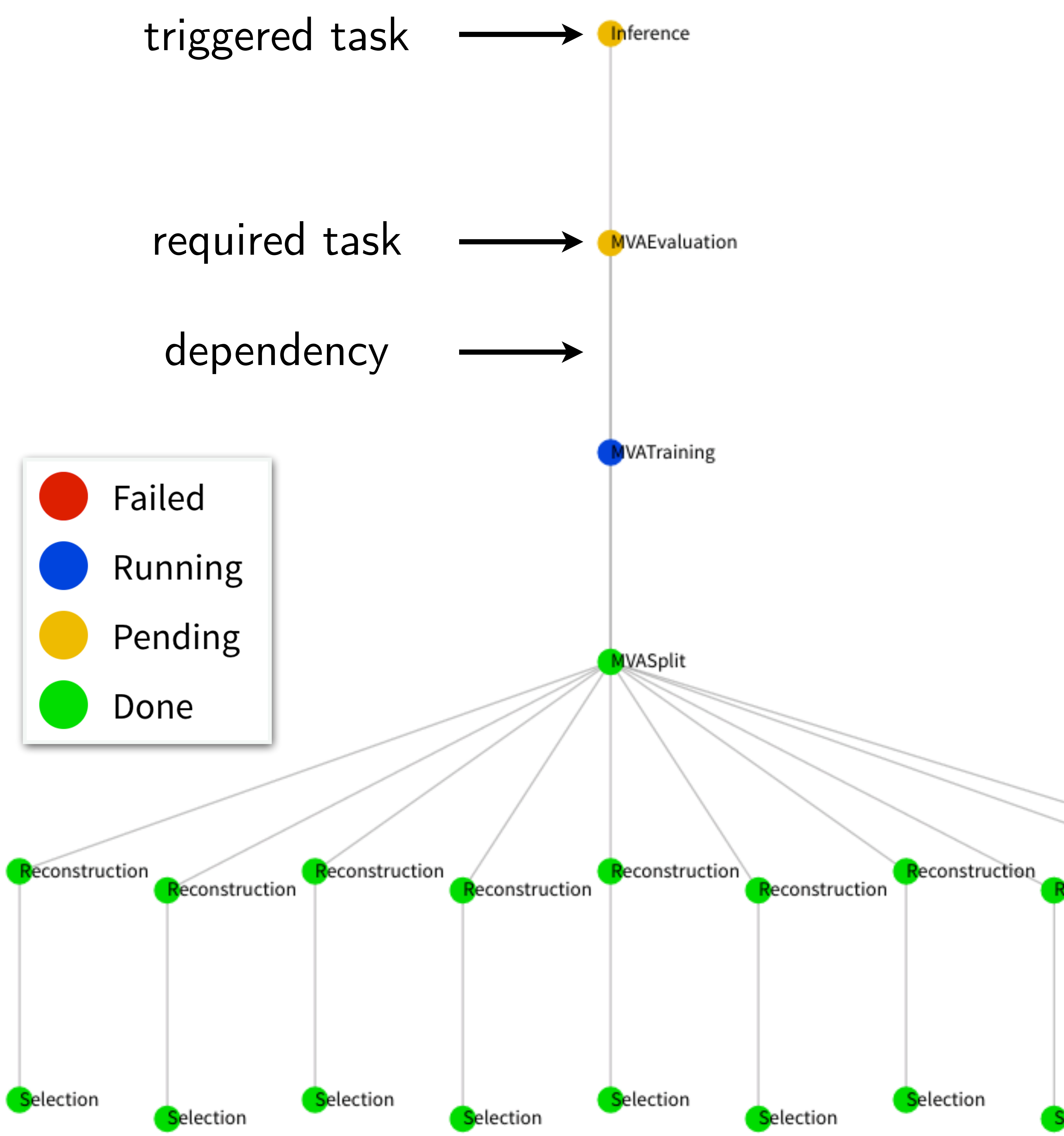
```
> python reco.py Reconstruction --dataset ttbar
```

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



triggered task

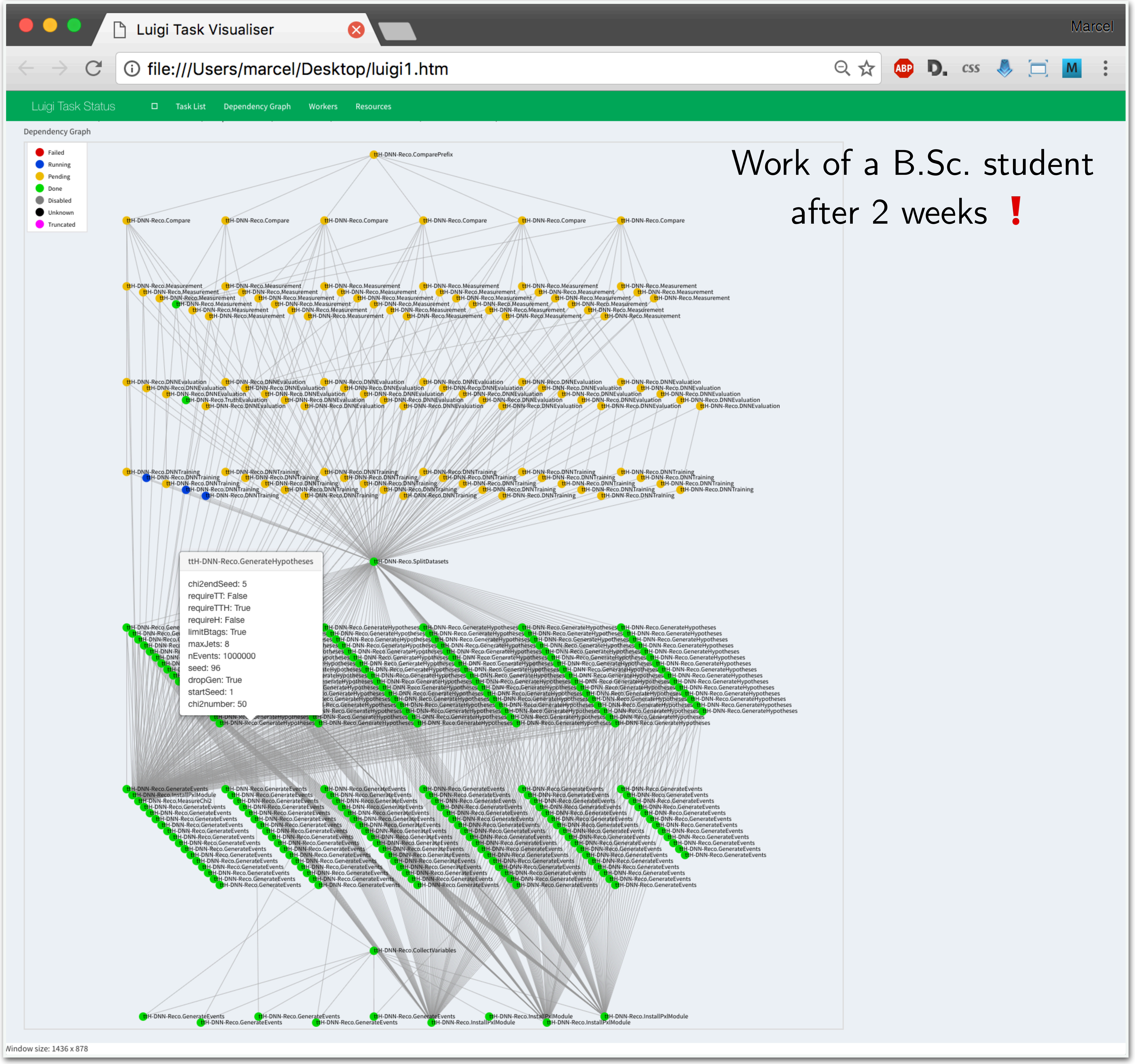
required task

dependency

- ● Failed
- ● Running
- ● Pending
- ● Done

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

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 !

HEP concepts, constraints & peculiarities (aka "reality check")

Design choices

- **Purpose**
  - Analysis workflow system that provides necessary tools to develop an automated analysis **right from the start**
    - ▻ Ability to adapt to **all possible resources** (software stacks, remote file access, submission to batch systems, ...)
    - ▻ Features for **interactive** work
    - ▻ **Collaborative** aspects
    - → More details on next slides

! A system that is designed for **a-posteriori analysis preservation** is not necessarily an appropriate candidate for a "***w**orkflow **d**evelopment **e**nvironment*" for large analyses

Design choices

- **Purpose**
  - Analysis workflow system that provides necessary tools to develop an automated analysis **right from the start**
    - Ability to adapt to **all possible resources** (software stacks, remote file access, submission to batch systems, ...)
    - Features for **interactive** work
    - **Collaborative** aspects
    - → More details on next slides
  - ! A system that is designed for **a-posteriori analysis preservation** is not necessarily an appropriate candidate for a "*__w__orkflow __d__evelopment __e__nvironment*" for large analyses

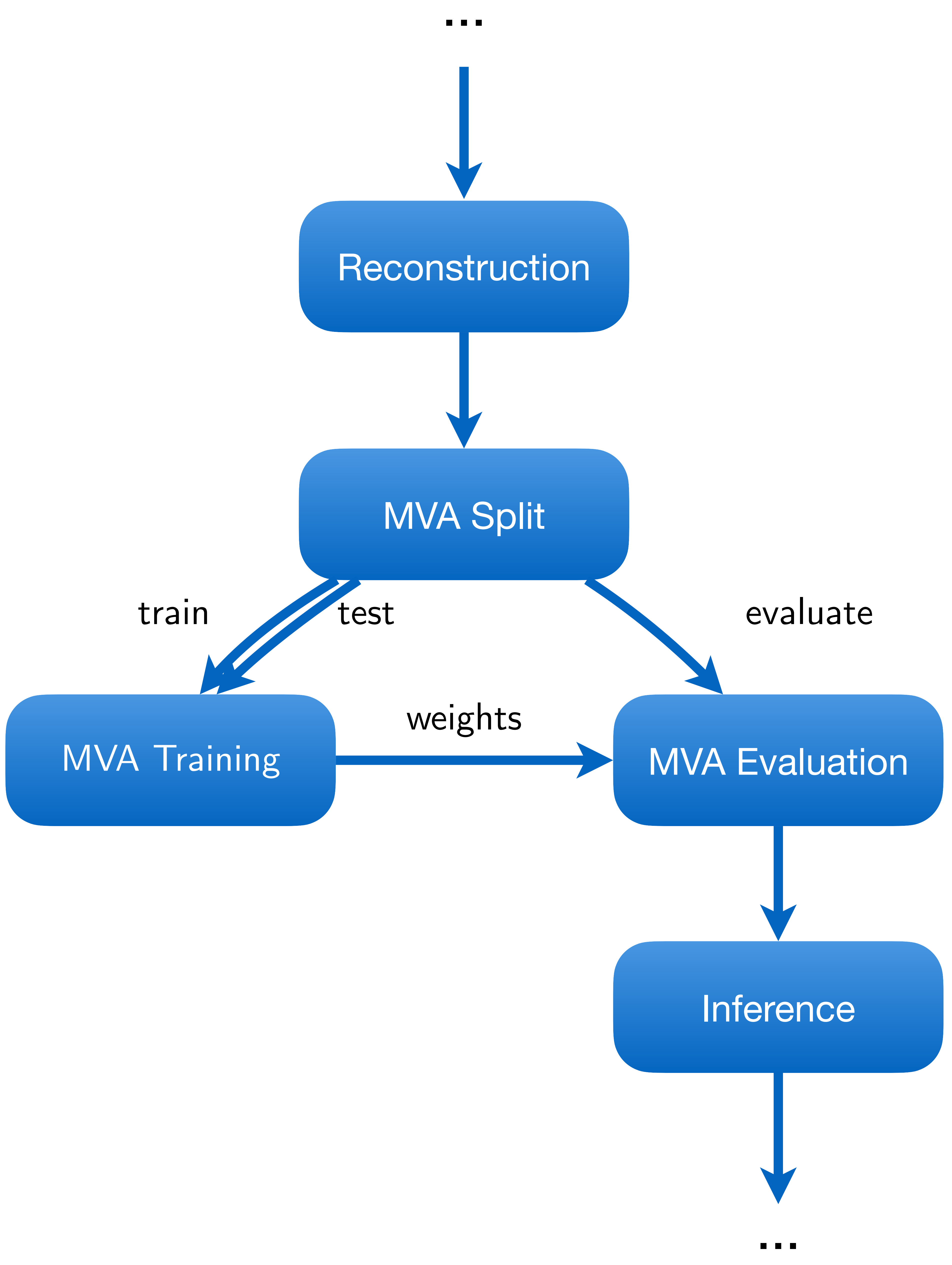
- **Typical usage**
  - Most analysis development is done by **PhD students** and **early PostDocs** (popular exception: "*framework devs*")
    - Structure of an analysis (workflow shape) might not be perfectly clear a-priori
    - Several stages in the course of an analysis that can cause perturbations
      - Commencing collaboration with other groups
      - Internal reviews and suggestions to restructure / repurpose an analysis
  - A *typical* analysis cycle ...
    - Year 1: "Let's start from scratch and plan everything ahead. This is going to be great."
    - Year 2: "Ok, we didn't know we had to consider *XYZ*. But we can still make it happen ..."
    - Year 3+n: " 🔩🪛 it! My contract is ending & I need that paper to apply for a job. Let's do workarounds ..."

Design choices (cont'd)

- **Language & flexibility**
  - An physics analysis workflow is not a simple sequence of steps
    - Being able to model dynamic "paths" is a **mandatory feature**
    - Only parts of the workflow shape are predictable, some are not!

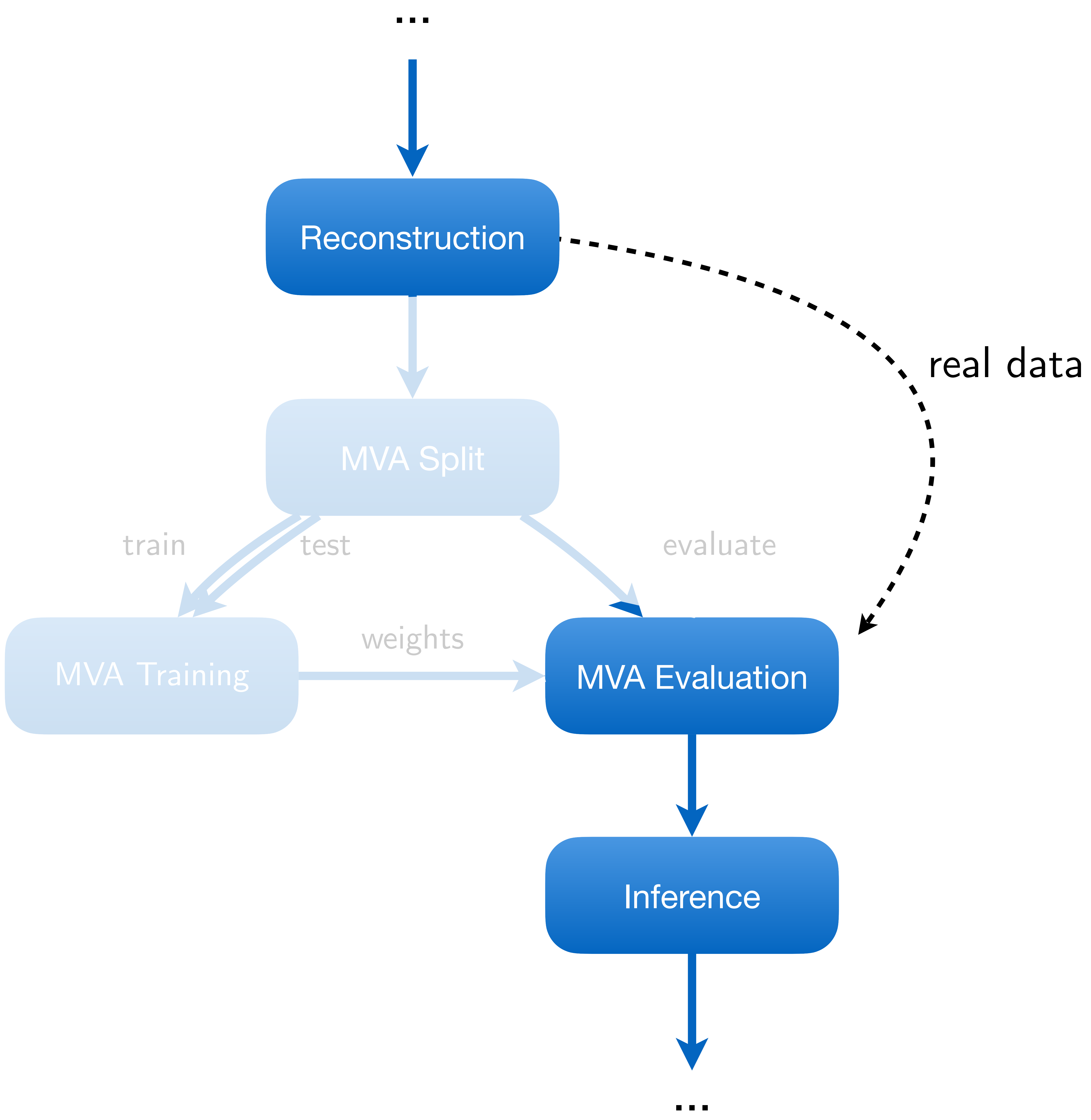
A typical example: ML workflow with uncertainties

Nominal MC



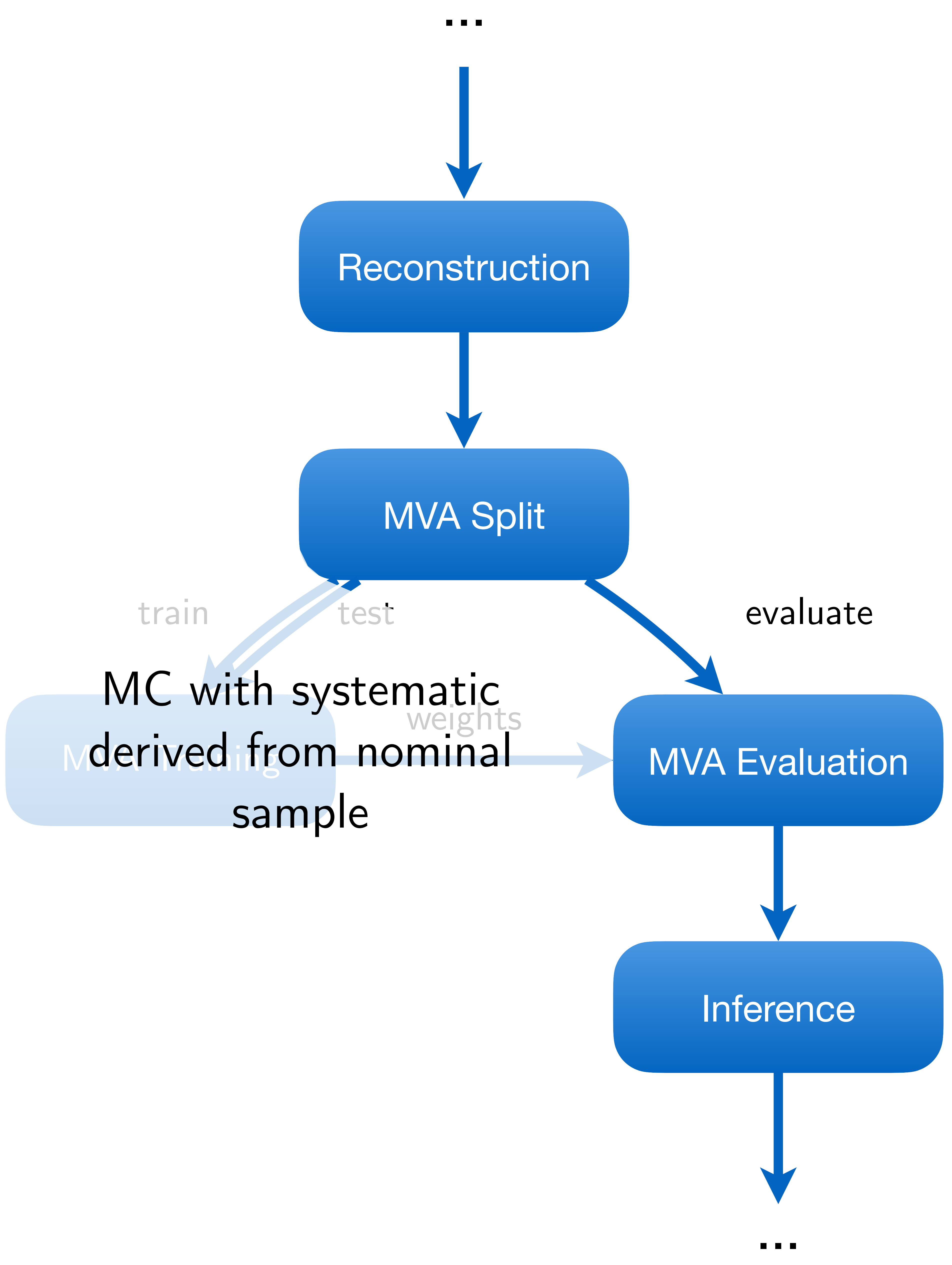
A typical example: ML workflow with uncertainties

Data



A typical example: ML workflow with uncertainties

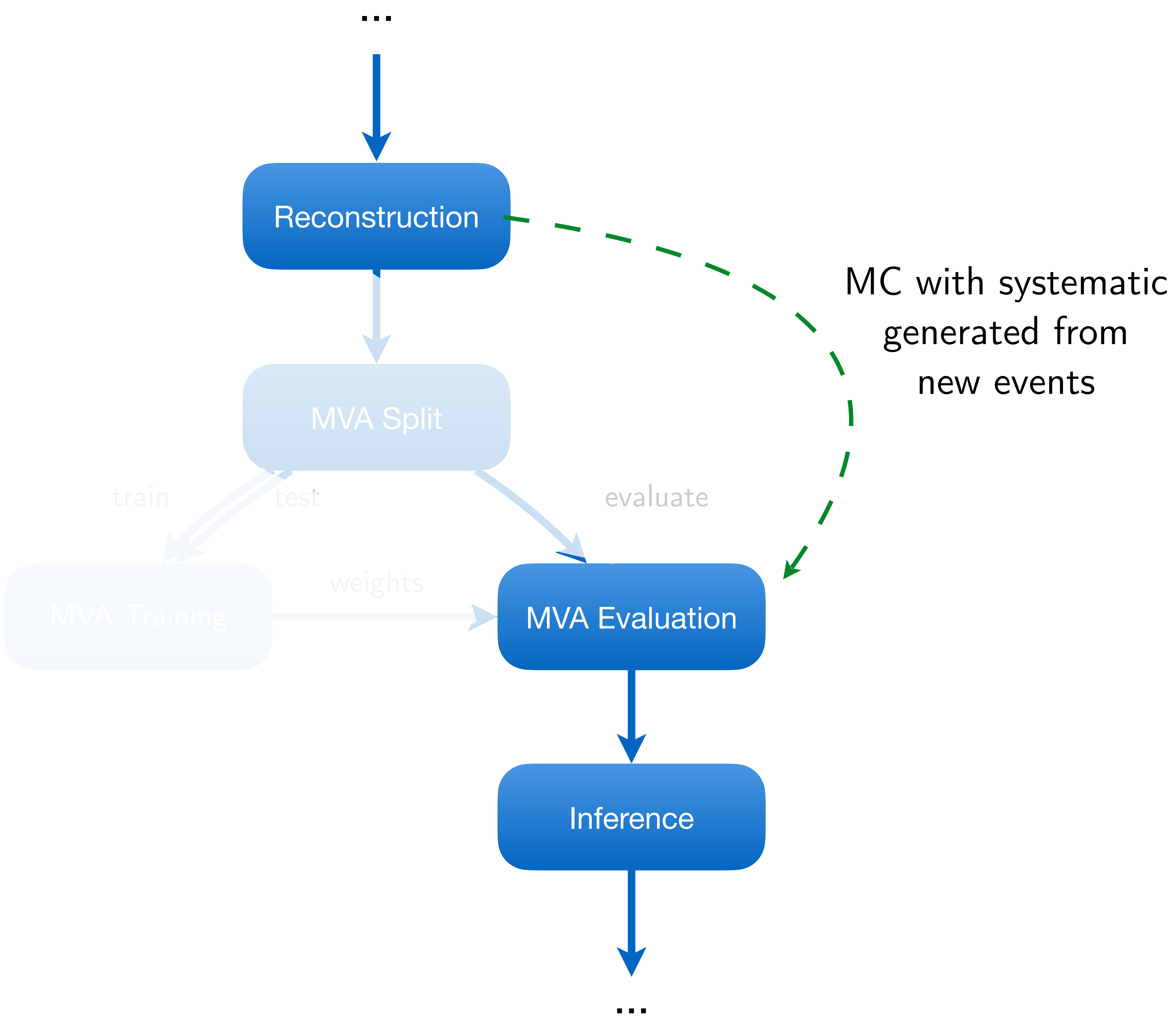
MC, Syst. I



MC with systematic derived from nominal sample

A typical example: ML workflow with uncertainties

MC, Syst. II



Design choices (cont'd)

- **Language & flexibility**
  - An physics analysis workflow is **not a simple sequence of steps**
    - Being able to model dynamic "paths" is a **mandatory feature**
    - Only parts of the workflow shape are predictable, some are not!
  - Dynamic behavior can depend on **many (!)** aspects, categorized into three classes:

a) a-priori known:	easy to consider into analysis design
b) a-priori **unknown:**	potential for severe disruptions, especially in late stages
c) dynamic:	the workflow shape is not fully determined at execution time but can depend on outcomes at runtime

  - People are aware of potential risks and
    - hesitate to use new tools - while solving a **short-term** issue - might constrain them **long-term**
      - collaborative / centralized development and training!
    - avoid straying too far from their current point of expertise
    - **for defining workflows**, want to use a language they know
      - just to be equipped for what might come down the road

"Reality check"

- **Remote storage is mandatory**
  - Local storage (e.g. at lab or institute) not always sufficient
  - Using only local storage constraints you to use only (the only?) local batch system
  - When collaborating with groups, copying files manually between sites is **error prone** & **high-maintenance**!

"Reality check"

- **Remote storage is mandatory**
  - Local storage (e.g. at lab or institute) not always sufficient
  - Using only local storage constraints you to use only (the only?) local batch system
  - When collaborating with groups, copying files manually between sites is **error prone** & **high-maintenance**!

- **Analyses are large**
  - Imagine $\mathcal{O}(1\mathrm{M})$ tasks $\approx \mathcal{O}(1\mathrm{M})$ (file based) outputs and a target-based workflow engine
  - Starting the workflow requires checking the existence of many (remote) files
  - Without doing optimizations, **this will just not work** (and site admins *will* find you 👀)

"Reality check"

- **Remote storage is mandatory**
  - Local storage (e.g. at lab or institute) not always sufficient
  - Using only local storage constraints you to use only (the only?) local batch system
  - When collaborating with groups, copying files manually between sites is **error prone** & **high-maintenance**!

- **Analyses are large**
  - Imagine $\mathcal{O}(1\mathrm{M})$ tasks $\approx \mathcal{O}(1\mathrm{M})$ (file based) outputs and a target-based workflow engine
  - Starting the workflow requires checking the existence of many (remote) files
  - Without doing optimizations, **this will just not work** (and site admins *will* find you 👀)

- **Our IT infrastructure is (very) heterogeneous**
  - Different systems (storage, batch) and exerpise at different sites
  - Random yet typical example
    - Accessing files on site *X* via `webdav://`, and on *Y* via root://
    - Site *X* updates their configuration, and now `mkdir_rec` requests are no longer supported
      - Switch to `root://` on site *X* for `mkdir_rec`
    - Site *Y* updates their caching database to accelerate `stat` requests through `root://`, and now `mtime`'s are gone
      - Your local cache just got invalidated ...
      - Switch to `gsiftp://` on site *Y* for `stat`

law

luigi analysis workflow

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

- **Design follows three primary goals**

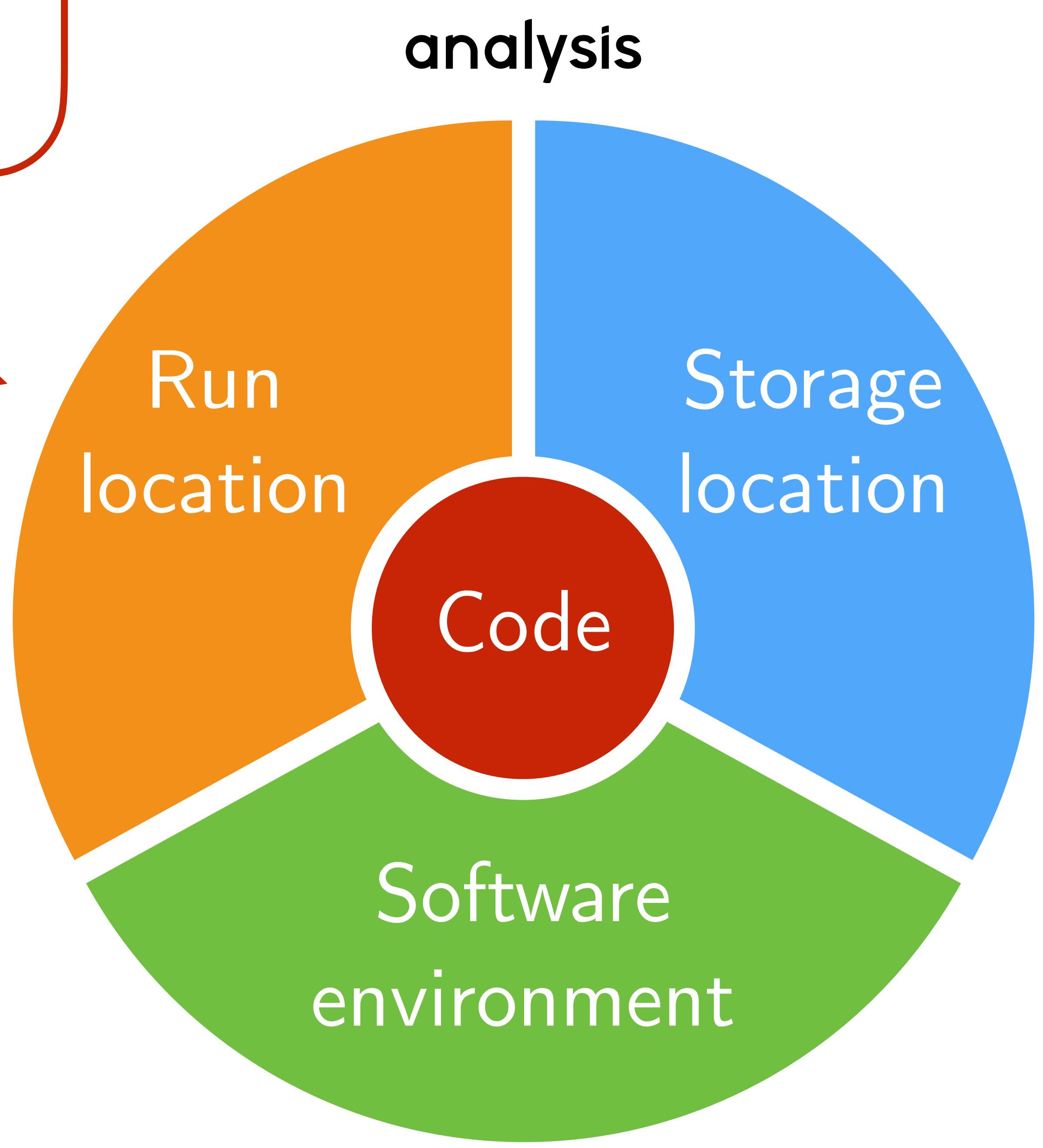
1. Experiment-agnostic core (in fact, not even related to physics)
2. Scalability on HEP infrastructure (but not limited to it*)
3. Decoupling of **run locations**, **storage locations** & **software environments**
   - ▻ Not constrained to specific resources, all components interchangeable

- Toolbox to follow an **analysis design pattern**
  - → Not a *framework* (no language or data format constraints)
  - → Serves as a **day-to-day working environment** allowing to prototype and automatically scale-out for free

- **Most used** workflow system for analyses in CMS
  - O(30) analyses, O(100) people
  - Central groups, e.g. HIG, TAU, BTV

- Also used outside CMS (e.g. LIGO) and outside HEP

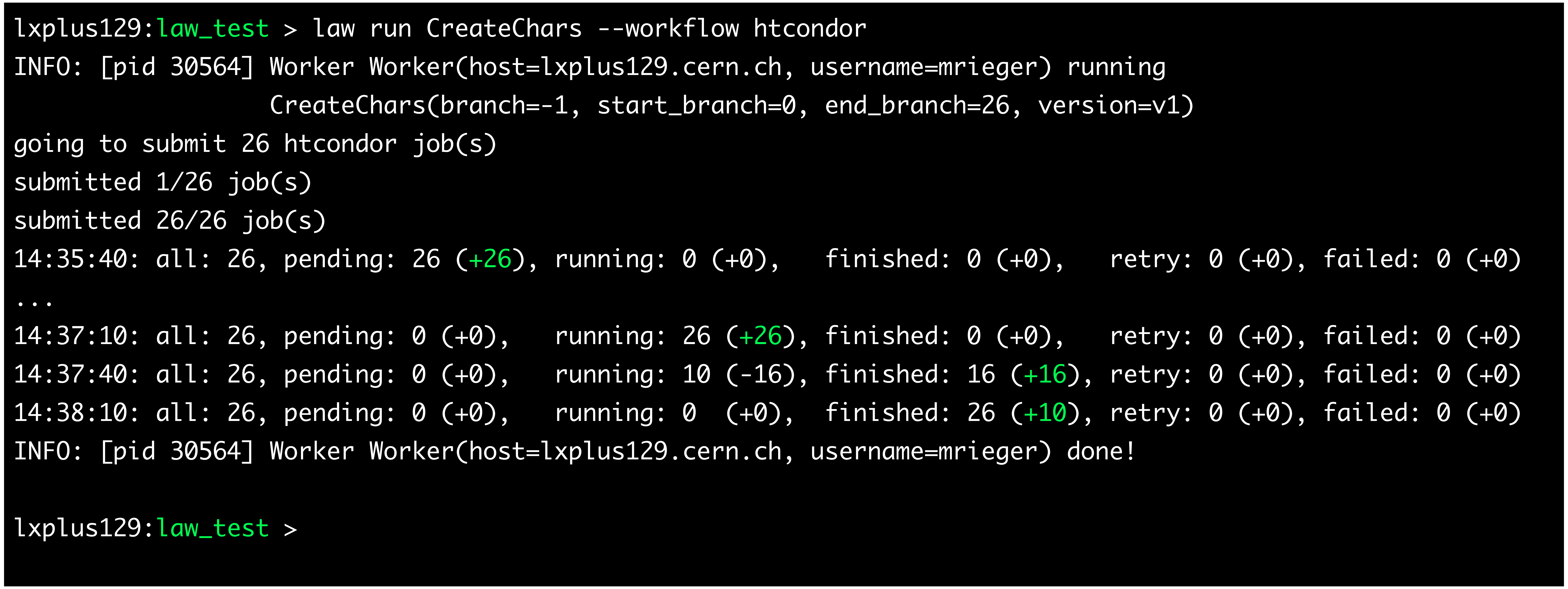
law
luigi analysis workflow



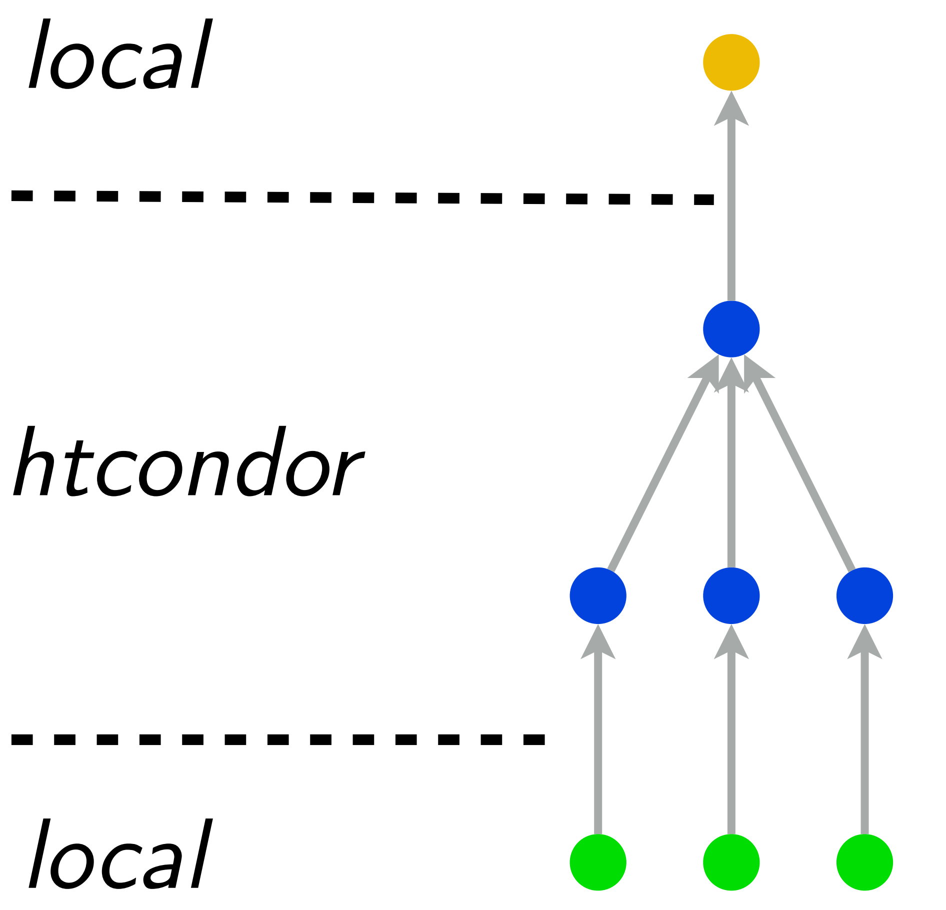
Law features (1)

1. Job submission

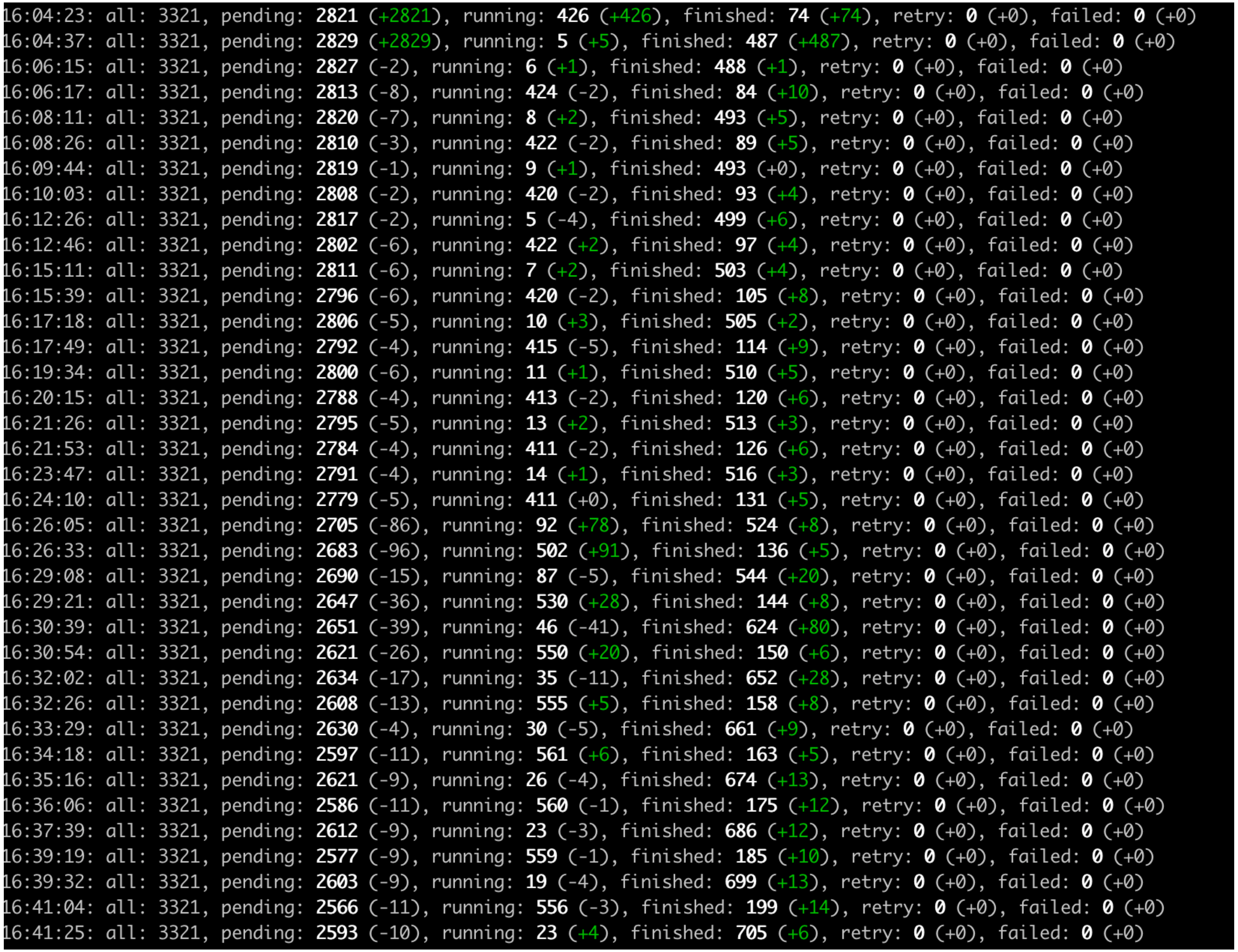
- Idea: submission built into tasks, **no need to write extra code**
- Currently supported job systems: HTCondor, SLURM, LSF, gLite, ARC, CMS-CRAB new
- Mandatory features such as automatic resubmission, flexible task ↔ job matching, job files fully configurable at submission time, internal job staging in case of saturated queues, ...
- 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 >
```



Job status polling from CMS HH combination



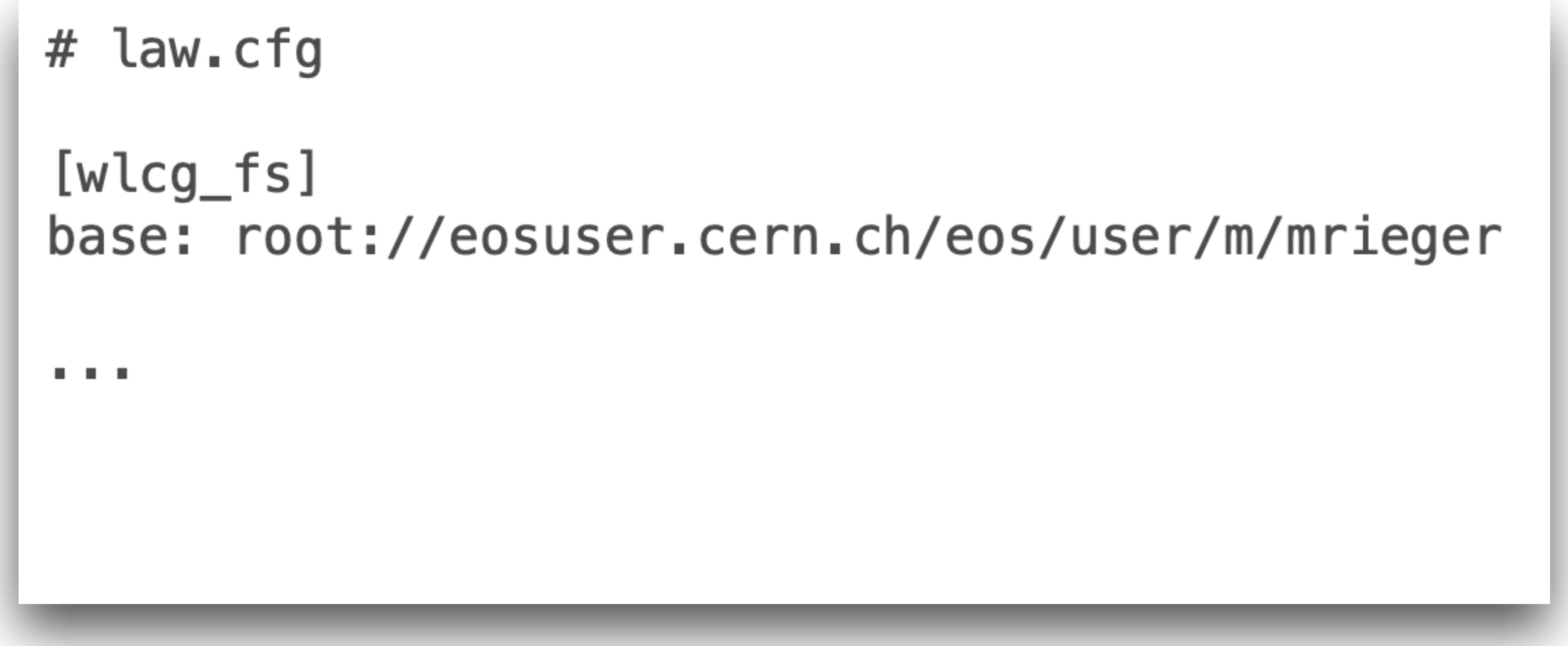
```
16:04:23: all: 3321, pending: 2821 (+2821), running: 426 (+426), finished: 74 (+74), retry: 0 (+0), failed: 0 (+0)
16:04:37: all: 3321, pending: 2829 (+2829), running: 5 (+5), finished: 487 (+487), retry: 0 (+0), failed: 0 (+0)
16:06:15: all: 3321, pending: 2827 (-2), running: 6 (+1), finished: 488 (+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)
```

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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - HDFS under development new
  - 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

...
```

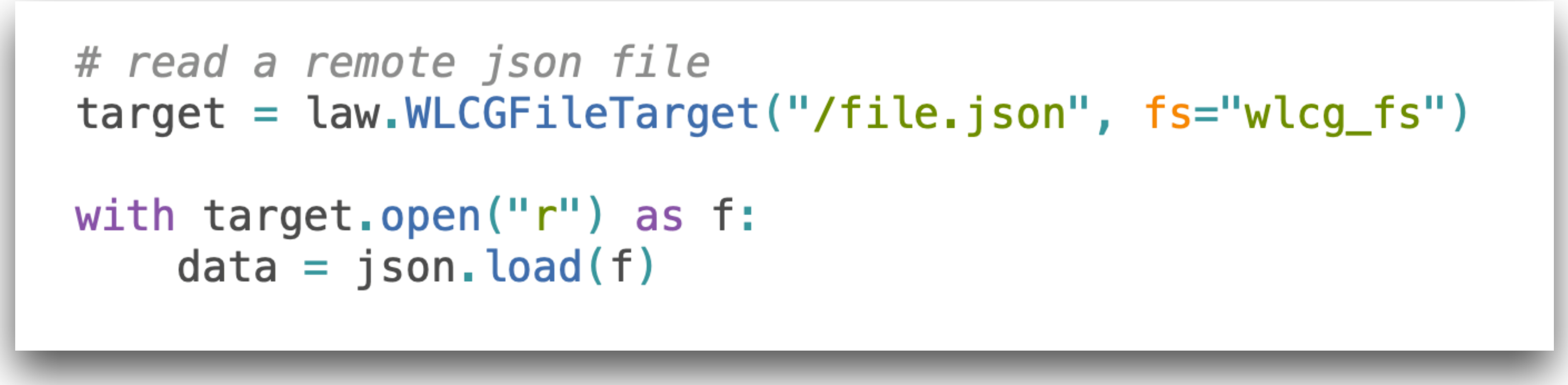
- Base path prefixed to all paths using this "fs"
- Configurable per file operation (stat, listdir, ...)
- Protected against removal of parent directories

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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ HDFS under development *new*
  - ▻ 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



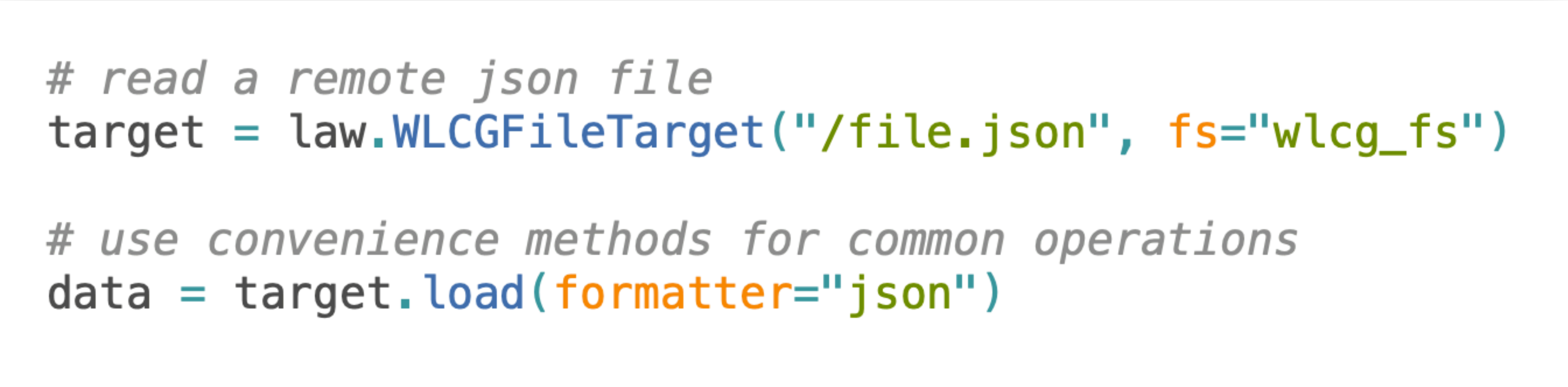
```
# read a remote json file
target = law.WLCGFileTarget("/file.json", fs="wlcg_fs")

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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ HDFS under development new
  - ▻ 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



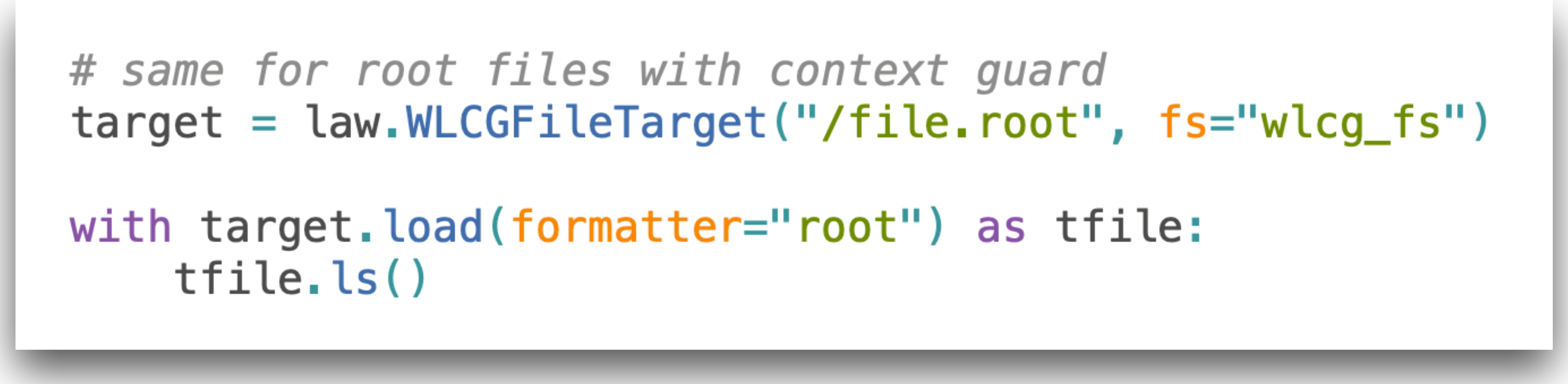
```python
# read a remote json file
target = law.WLCGFileTarget("/file.json", fs="wlcg_fs")

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

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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ HDFS under development *new*
  - ▻ 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



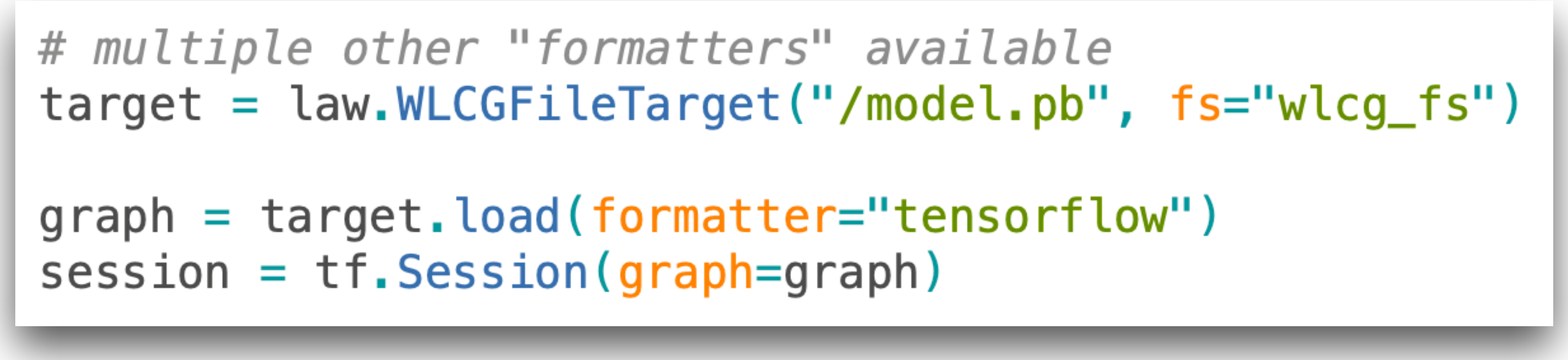
```
# same for root files with context guard
target = law.WLCGFileTarget("/file.root", fs="wlcg_fs")

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

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 (XRootD, WebDAV, GridFTP, dCache, SRM, ...) + DropBox
  - ▻ HDFS under development new
  - ▻ 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)
```

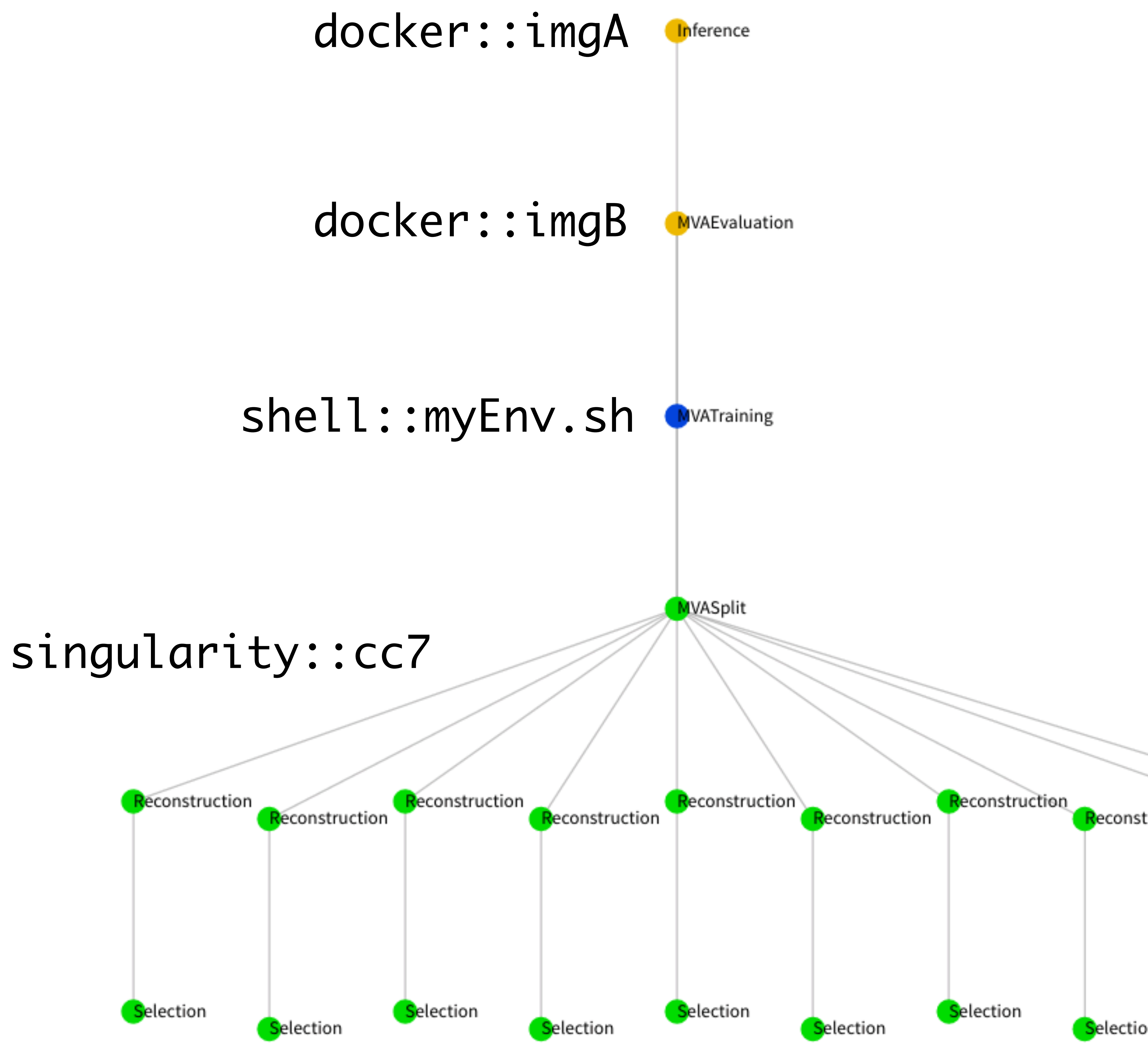
Law features (3)

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 (e.g. for CMSSW)
  - Virtual envs
  - Docker images
  - Singularity images



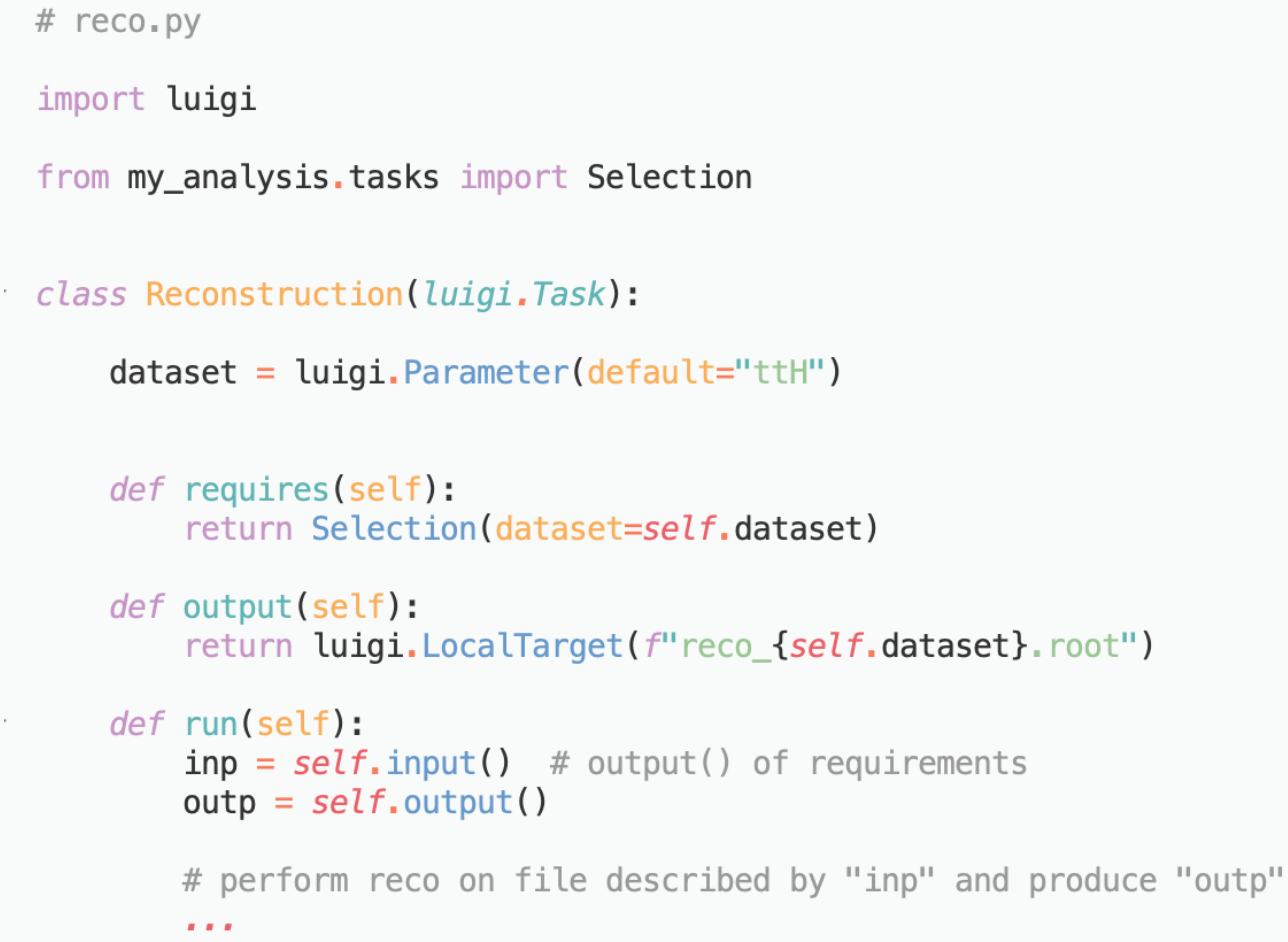
docker::imgA

docker::imgB

shell::myEnv.sh

singularity::cc7

Law in action



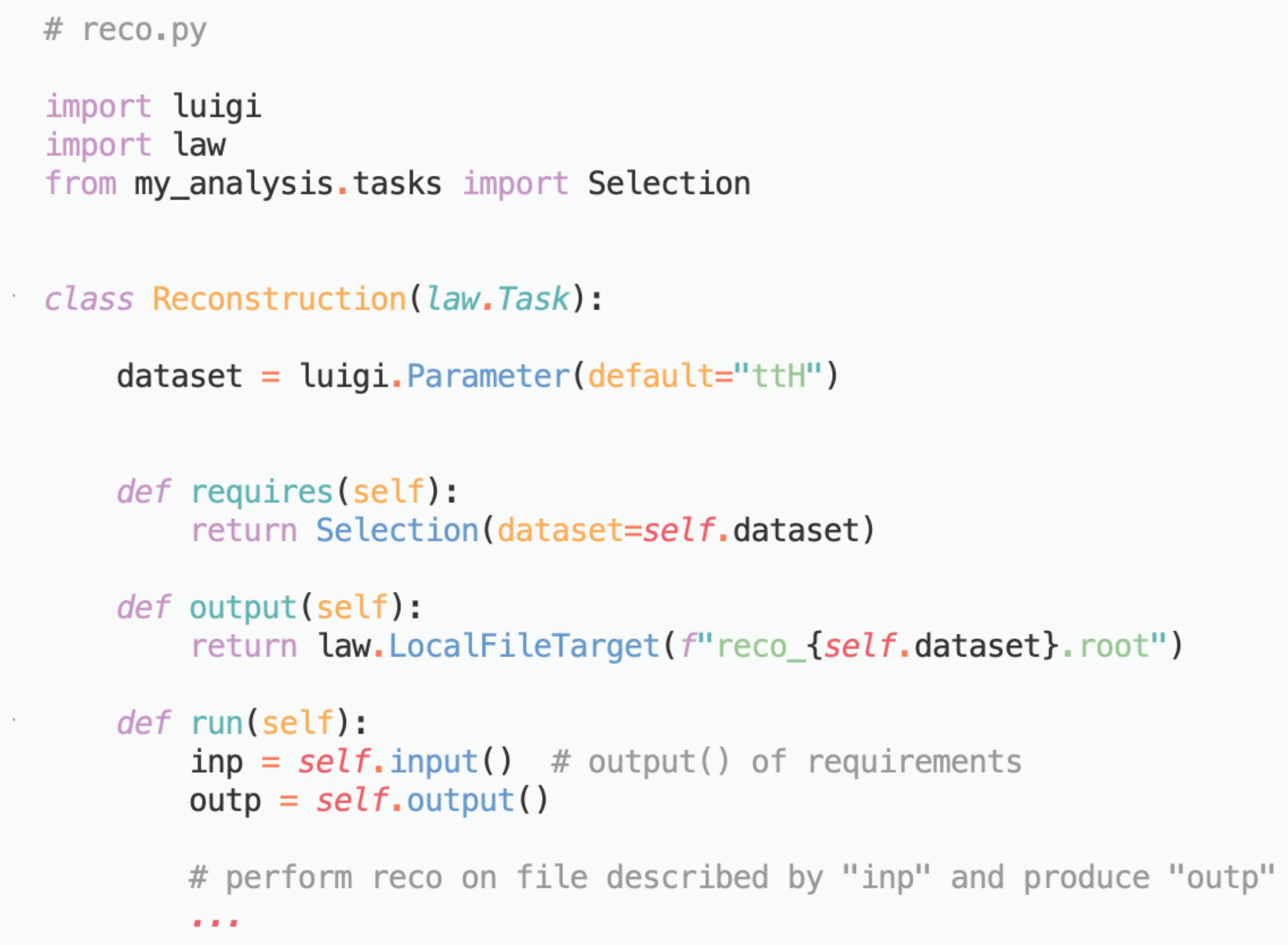
```python
# reco.py

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
```

Law in action



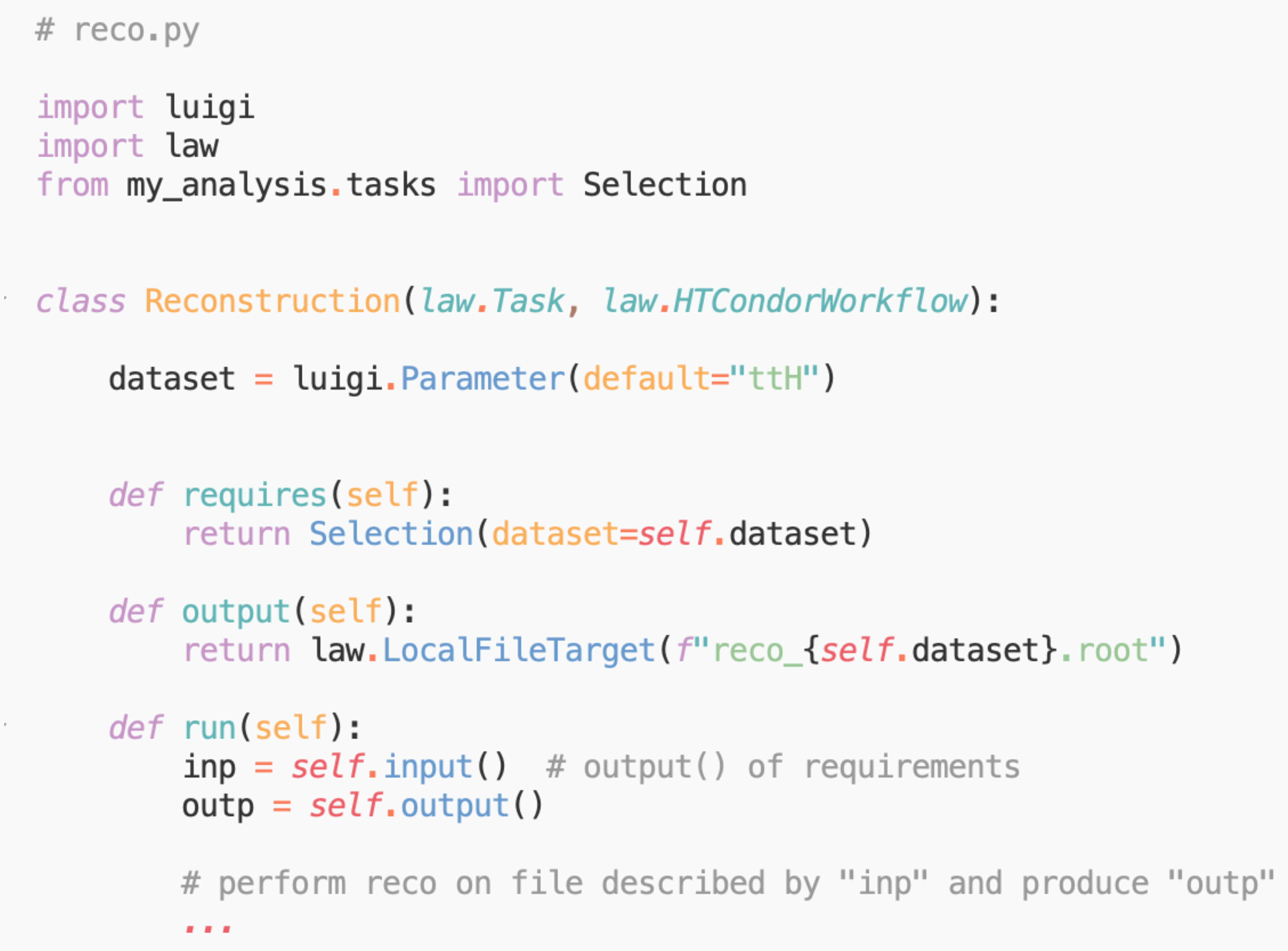
```
# reco.py

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
```

Law in action



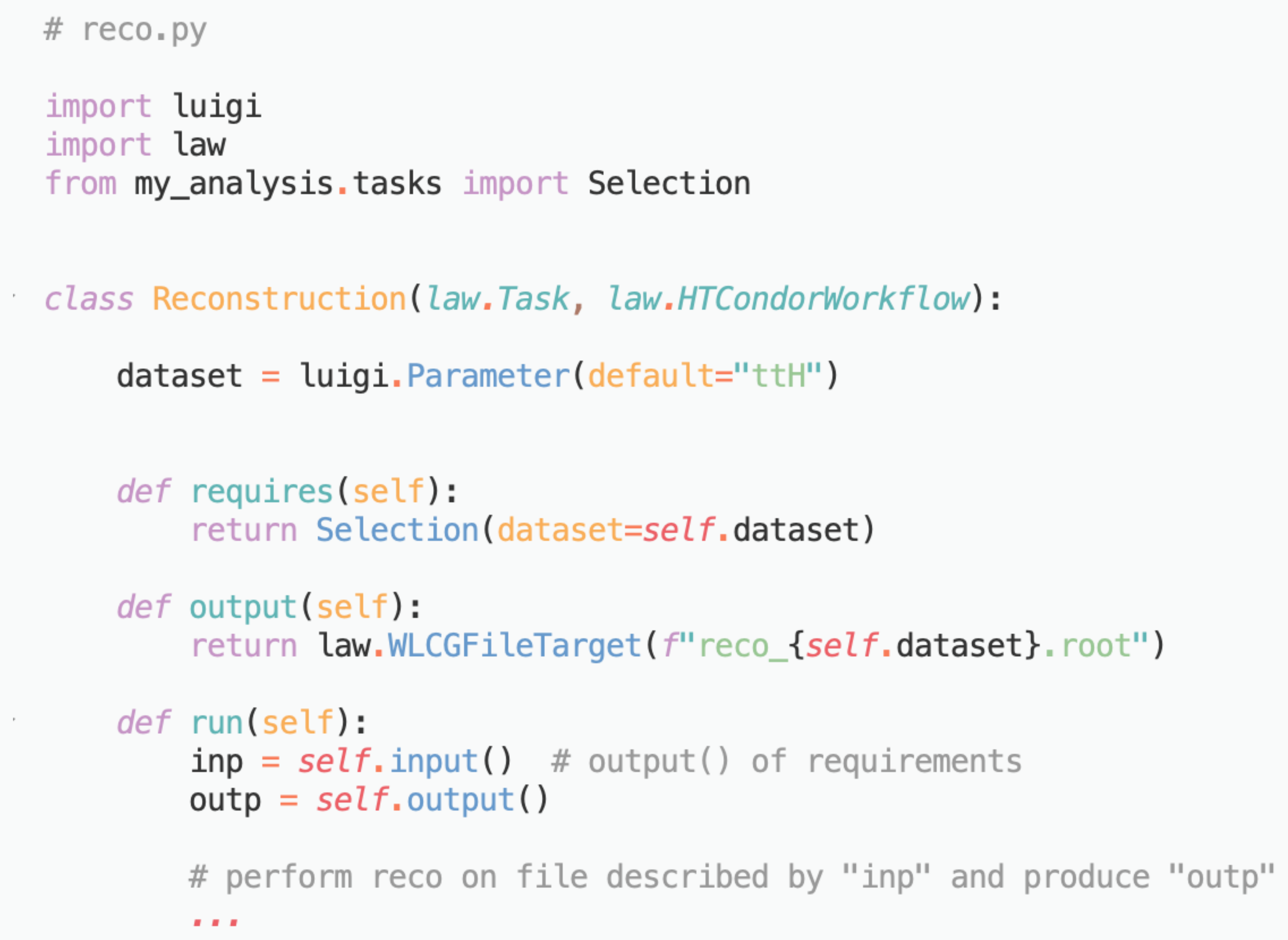
```python
# reco.py

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



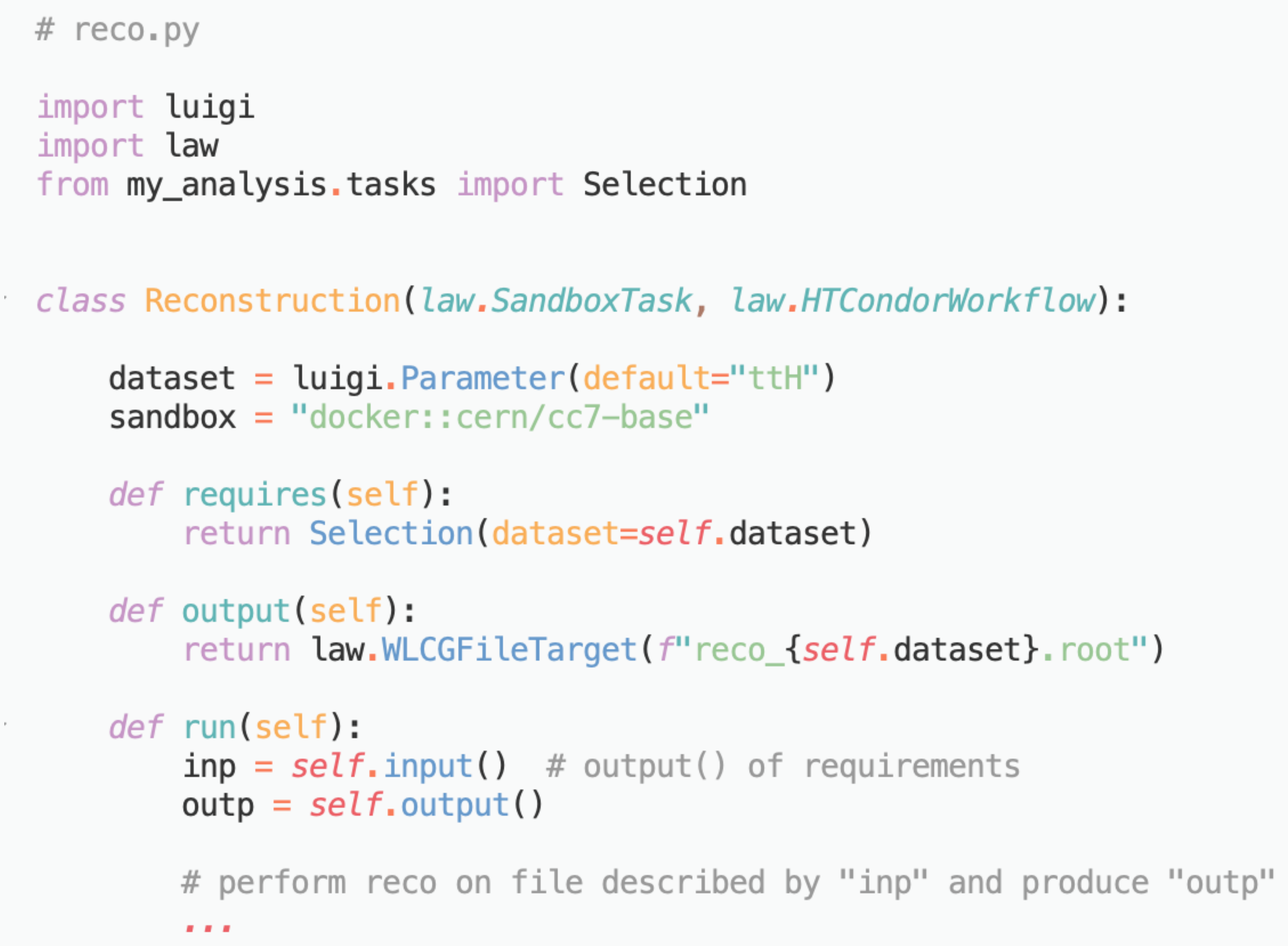
```python
# reco.py

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
```

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 ☞

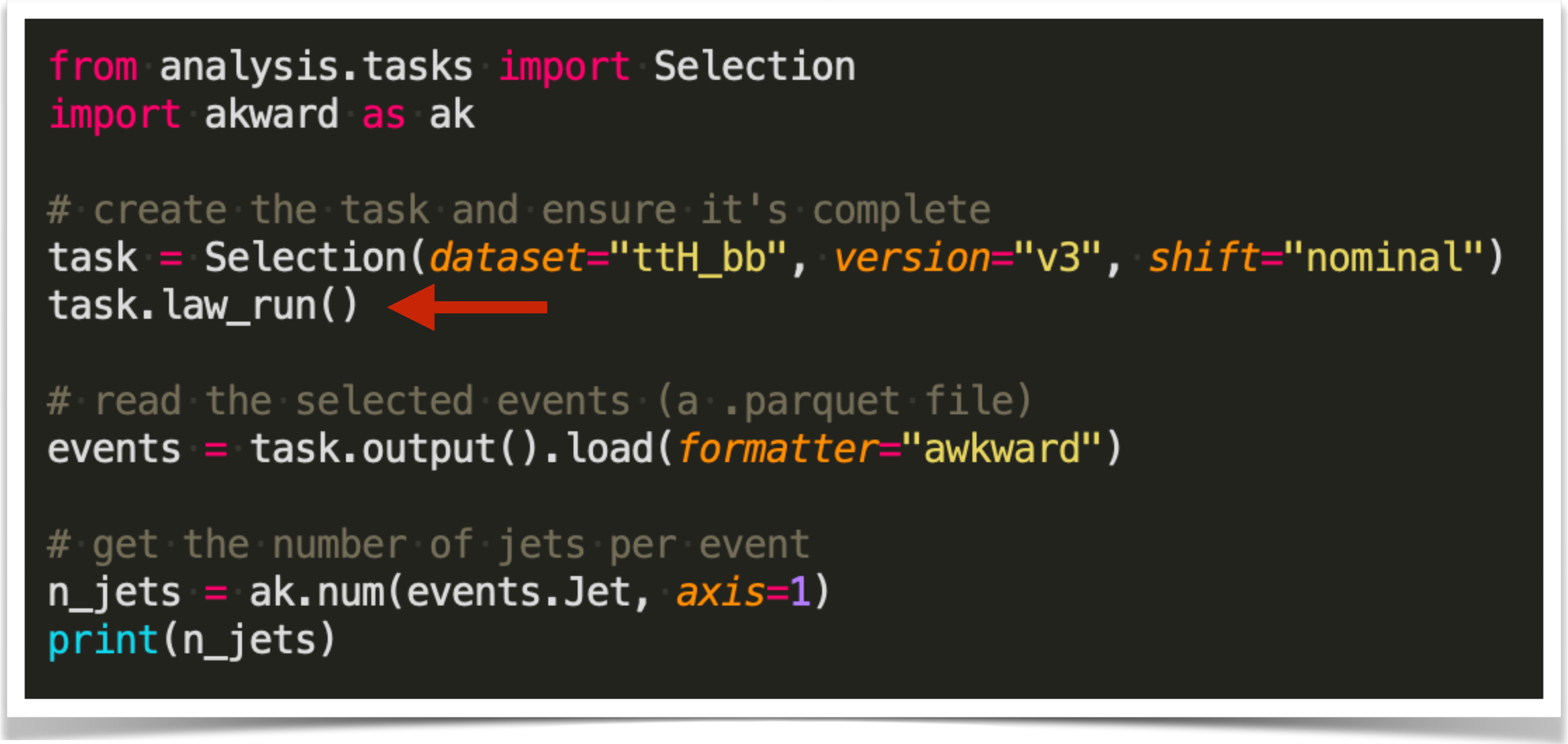
```
> law run Reconstruction --dataset ttbar --workflow htcondor
```

Triggers: CLI, scripting and notebooks

- **CLI**

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

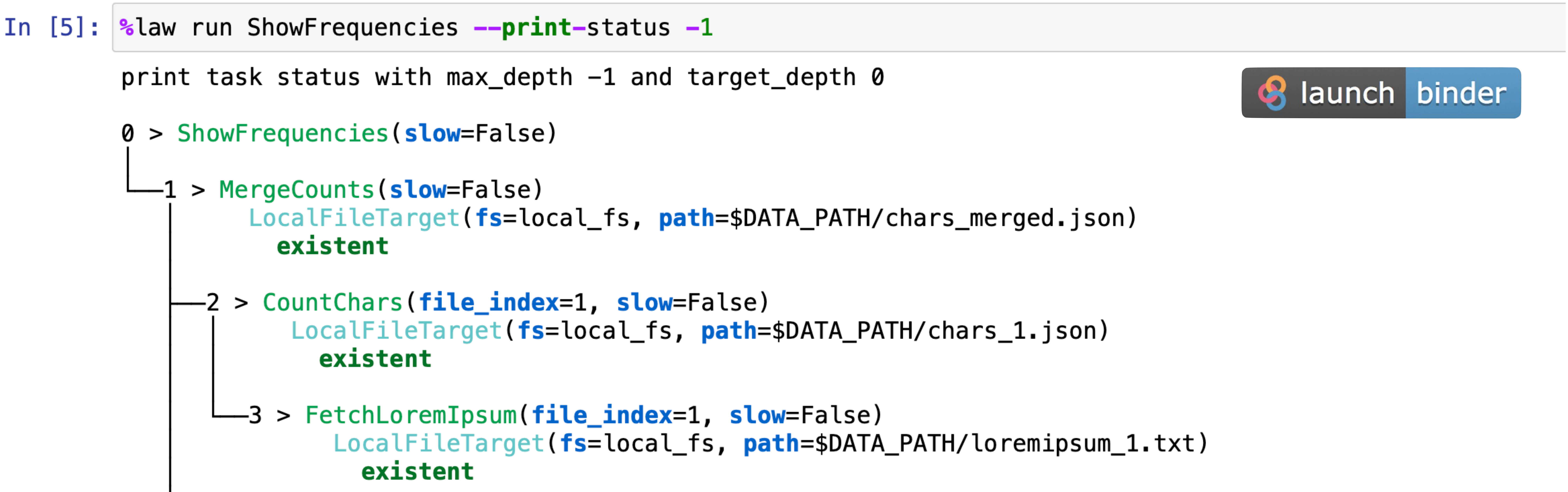
  - Full auto-completion of tasks and parameters



```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)
```

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

- **Notebooks**



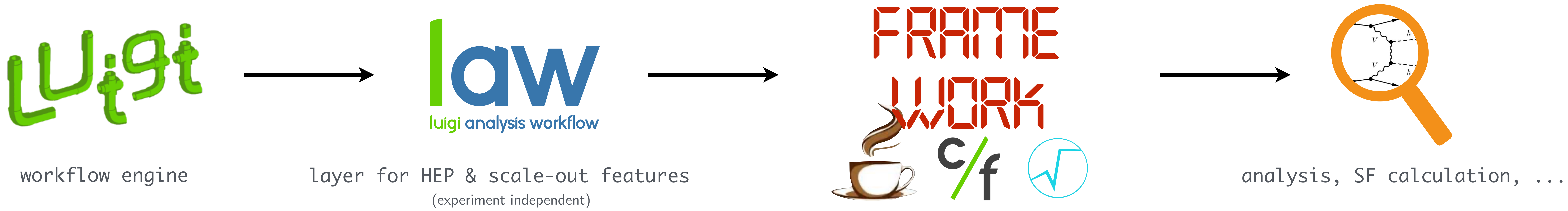
```
In [5]: %law run ShowFrequencies --print-status -1

        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
             │
```

launch binder

Summary

- Resource-agnostic workflow management **essential** for large & complex analyses

→ Need for a flexible **design pattern** to automate arbitrary workloads



workflow engine

layer for HEP & scale-out features
(experiment independent)

analysis, SF calculation, ...

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

→ Full decoupling of **run locations**, **storage locations** & **software environments**

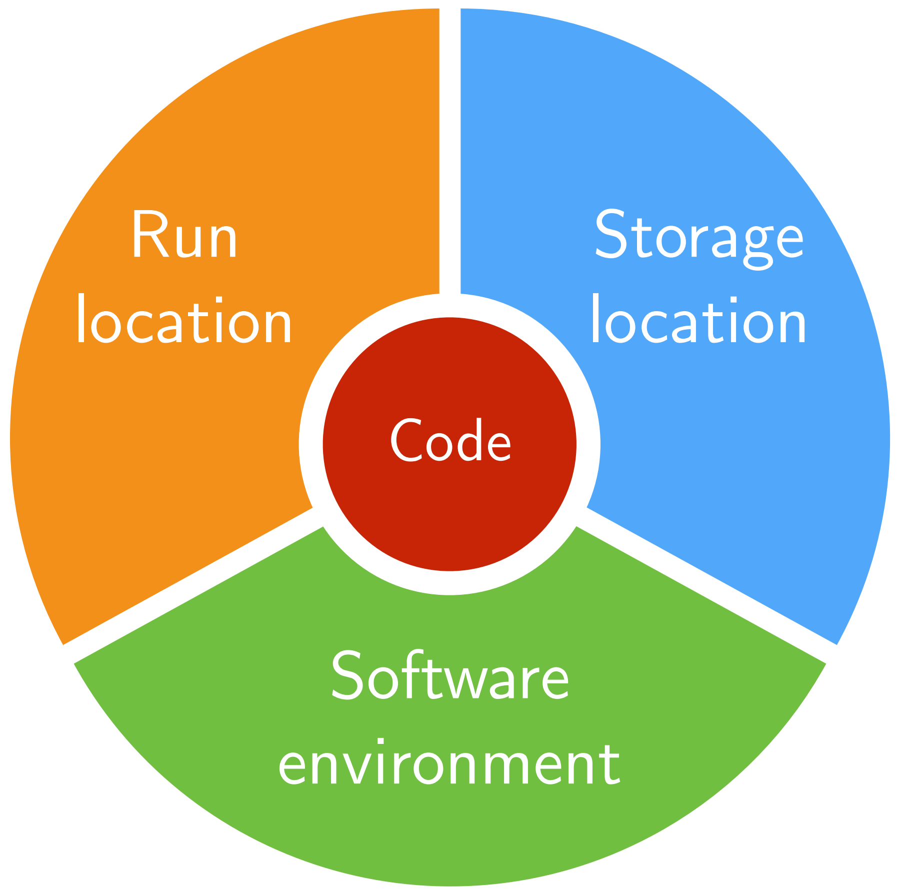
→ Allows to build frameworks that check every point in the CMS analysis wishlist

→ Currently working on full documentation and type annotations for next release

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

→ github.com/spotify/luigi, luigi.readthedocs.io

Collaboration & contributions welcome!



Run location

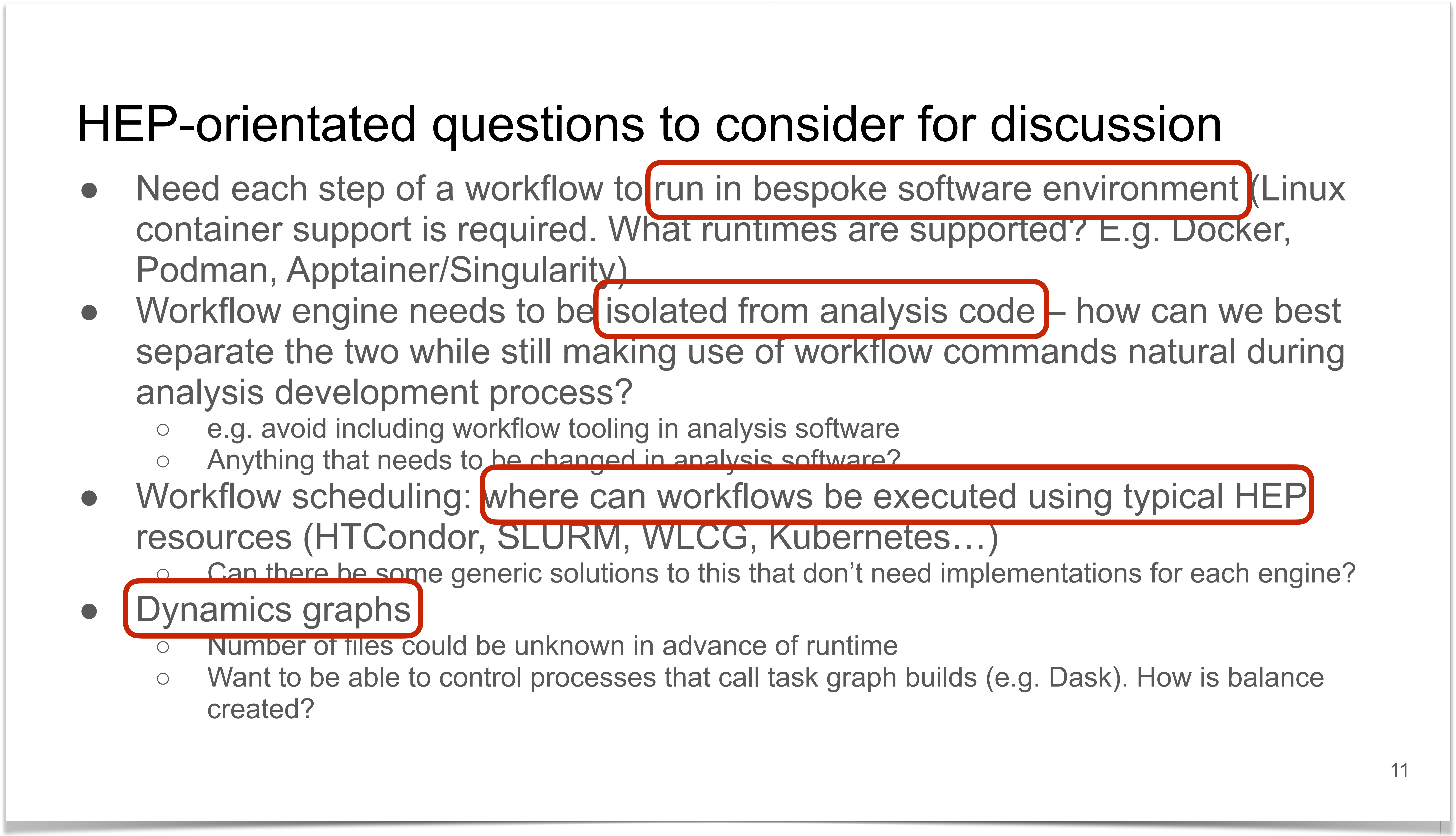
Storage location

Code

Software environment

Discussion starters

from Matthew's slides



HEP-orientated questions to consider for discussion

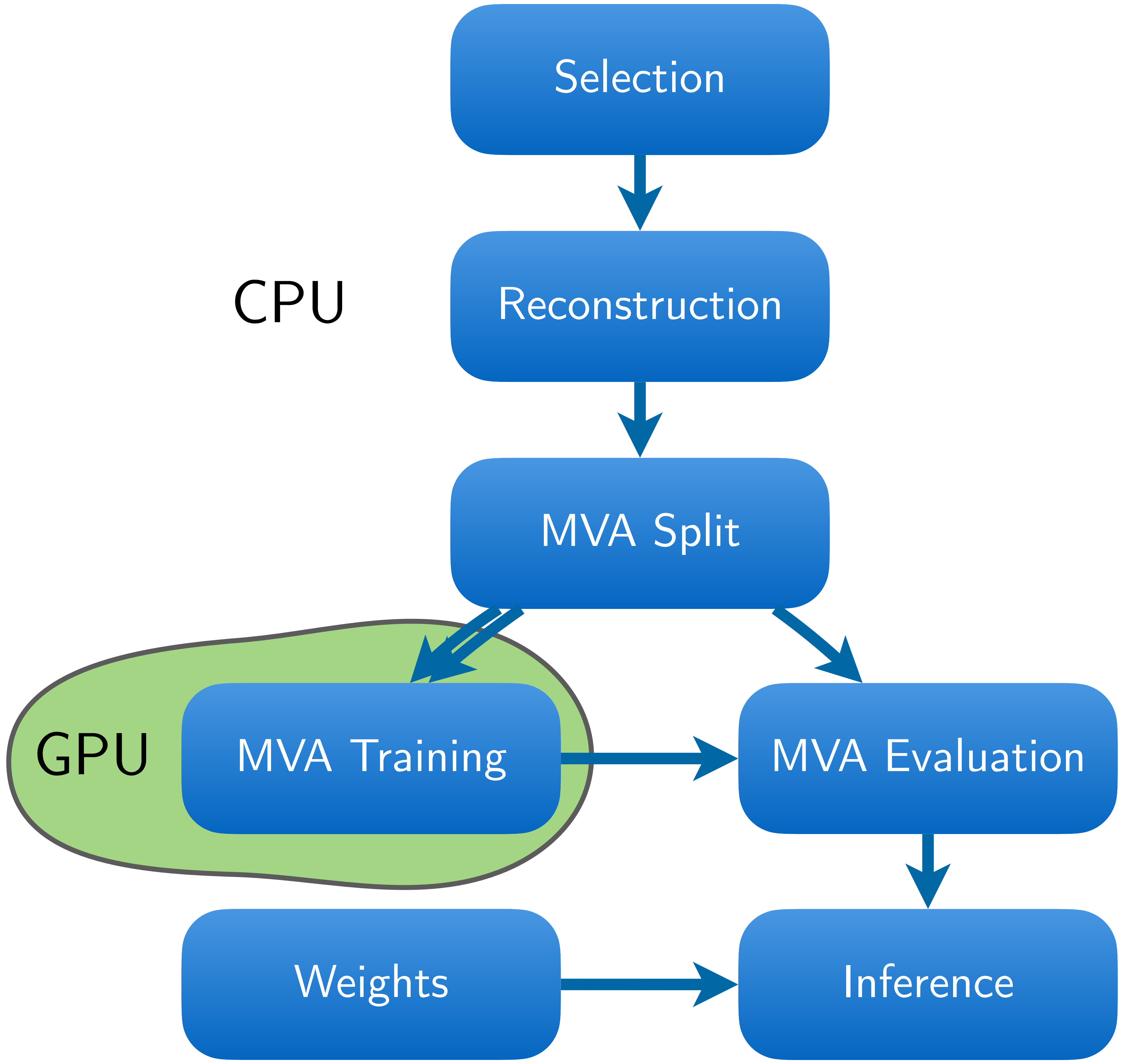
- Need each step of a workflow to run in bespoke software environment (Linux container support is required. What runtimes are supported? E.g. Docker, Podman, Apptainer/Singularity)
- Workflow engine needs to be isolated from analysis code – how can we best separate the two while still making use of workflow commands natural during analysis development process?
  - e.g. avoid including workflow tooling in analysis software
  - Anything that needs to be changed in analysis software?
- Workflow scheduling: where can workflows be executed using typical HEP resources (HTCondor, SLURM, WLCG, Kubernetes…)
  - Can there be some generic solutions to this that don't need implementations for each engine?
- Dynamics graphs
  - Number of files could be unknown in advance of runtime
  - Want to be able to control processes that call task graph builds (e.g. Dask). How is balance created?

Backup

Abstraction: analysis workflows

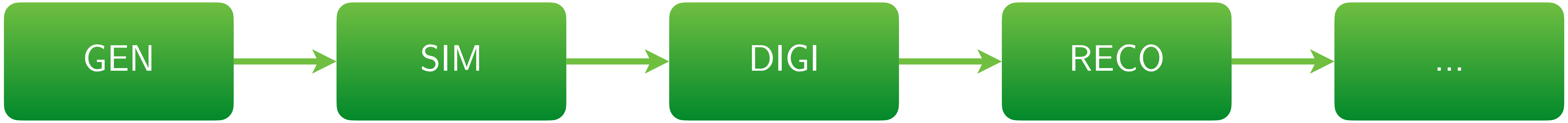
- Workflow, decomposable into particular workloads
- Workloads related to each other by common interface
  - In/outputs define directed acyclic graph (DAG)
- Alter default behavior via parameters
- Computing resources
  - Run location (CPU, GPU, WLCG, ...)
  - Storage location (local, dCache, EOS, ...)
- 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 production infrastructure
- Homogeneous software requirements

Wishlist for end-user analyses

- Structure “iterative”, a-priori unknown
- Dynamic workflows, fast R&D cycles
- DAG with arbitrary dependencies
- Incorporate *any* existing infrastructure
- Use custom software, everywhere

→ Requirements for HEP analyses mostly orthogonal

"Realistic" HEP workflow management

- **Consider this example again**

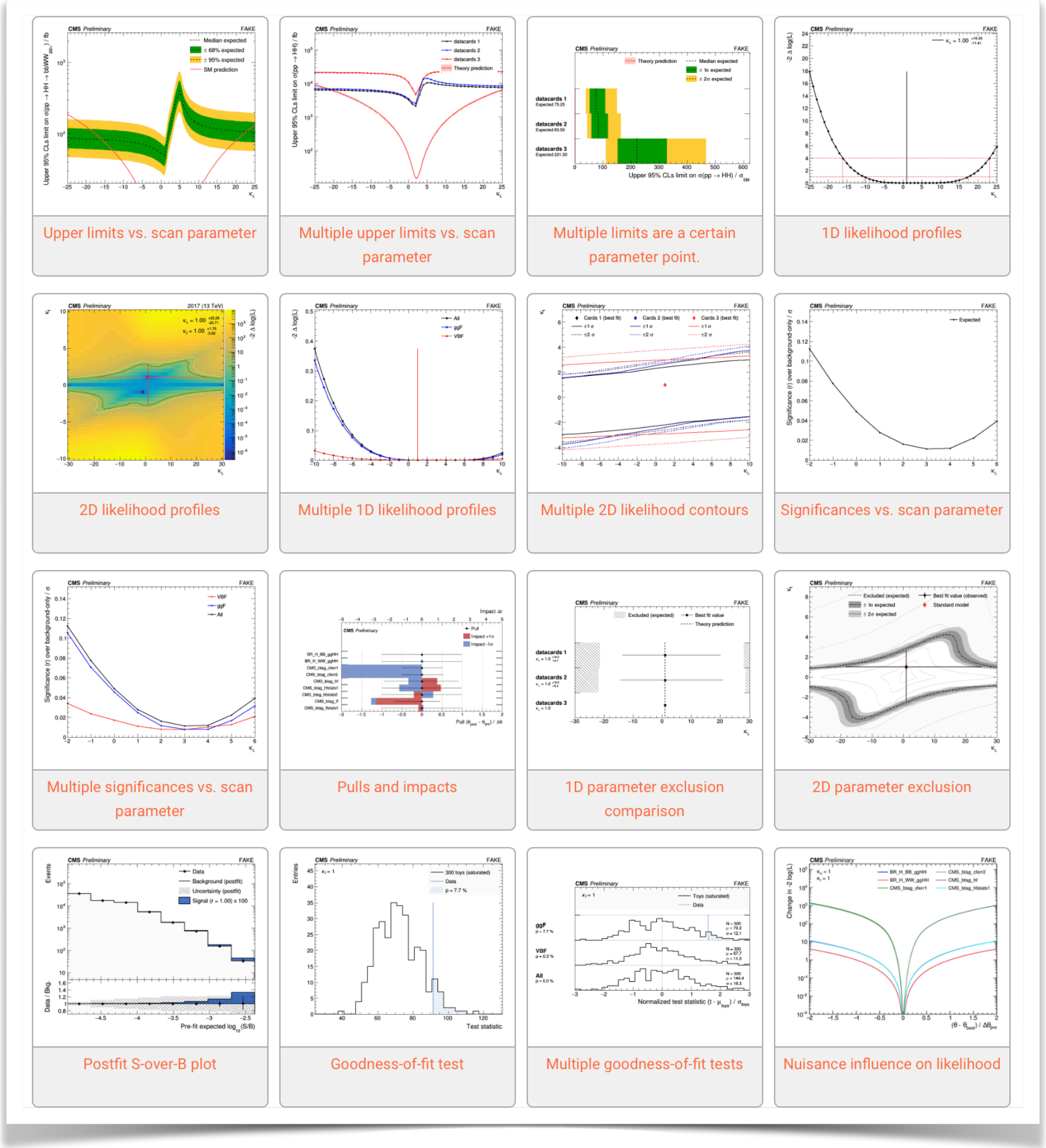
  ```
  > law run Reconstruction --dataset ttbar --workflow htcondor
  ```

  - $\mathcal{O}$(500 - 4k) files, stored either locally or remotely
  - Any workflow engine will first check if things need to be rerun
    - $\mathcal{O}$(500 - 4k) file requests (**via network**)!
    - Prepare for admins to find you 👀
  - *What law does*
    - Reconstruction is a workflow
    - Workflows output a so-called **TargetCollection**'s, containing all outputs of its branch tasks
    - **TargetCollection**'s can check if their files are located in the same directory
    - If they do, perform a single (remote) **listdir** and compare basenames → **single request**

- **There is no free lunch**
  - Our HEP resources (clusters, grid, storage elements, software environments) are very **inhomogeneous**
  - A **realistic** workflow engine
    - can make some good, yet simple assumptions based on known best-practices

      BUT
    - it should **always** allow users to transparently **change decisions** & **configure every single aspect!**

Inference tools used by HH searches



Upper limits vs. scan parameter

Multiple upper limits vs. scan parameter

Multiple limits are a certain parameter point.

1D likelihood profiles

2D likelihood profiles

Multiple 1D likelihood profiles

Multiple 2D likelihood contours

Significances vs. scan parameter

Multiple significances vs. scan parameter

Pulls and impacts

1D parameter exclusion comparison

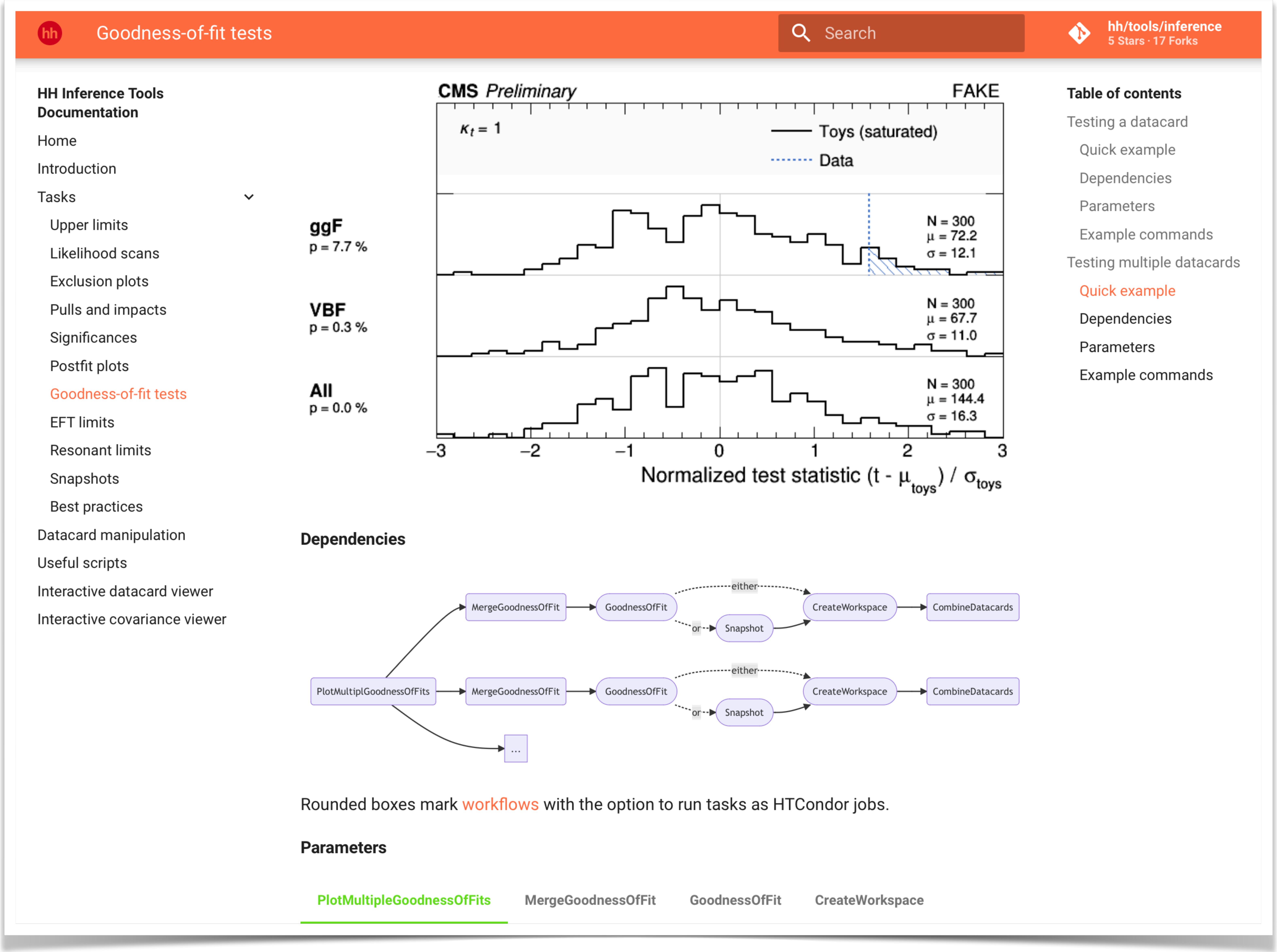
2D parameter exclusion

Postfit S-over-B plot

Goodness-of-fit test

Multiple goodness-of-fit tests

Nuisance influence on likelihood



HH Inference Tools Documentation

- Home
- Introduction
- Tasks
  - Upper limits
  - Likelihood scans
  - Exclusion plots
  - Pulls and impacts
  - Significances
  - Postfit plots
  - Goodness-of-fit tests
  - EFT limits
  - Resonant limits
  - Snapshots
  - Best practices
- Datacard manipulation
- Useful scripts
- Interactive datacard viewer
- Interactive covariance viewer

Dependencies

Rounded boxes mark workflows with the option to run tasks as HTCondor jobs.

Parameters

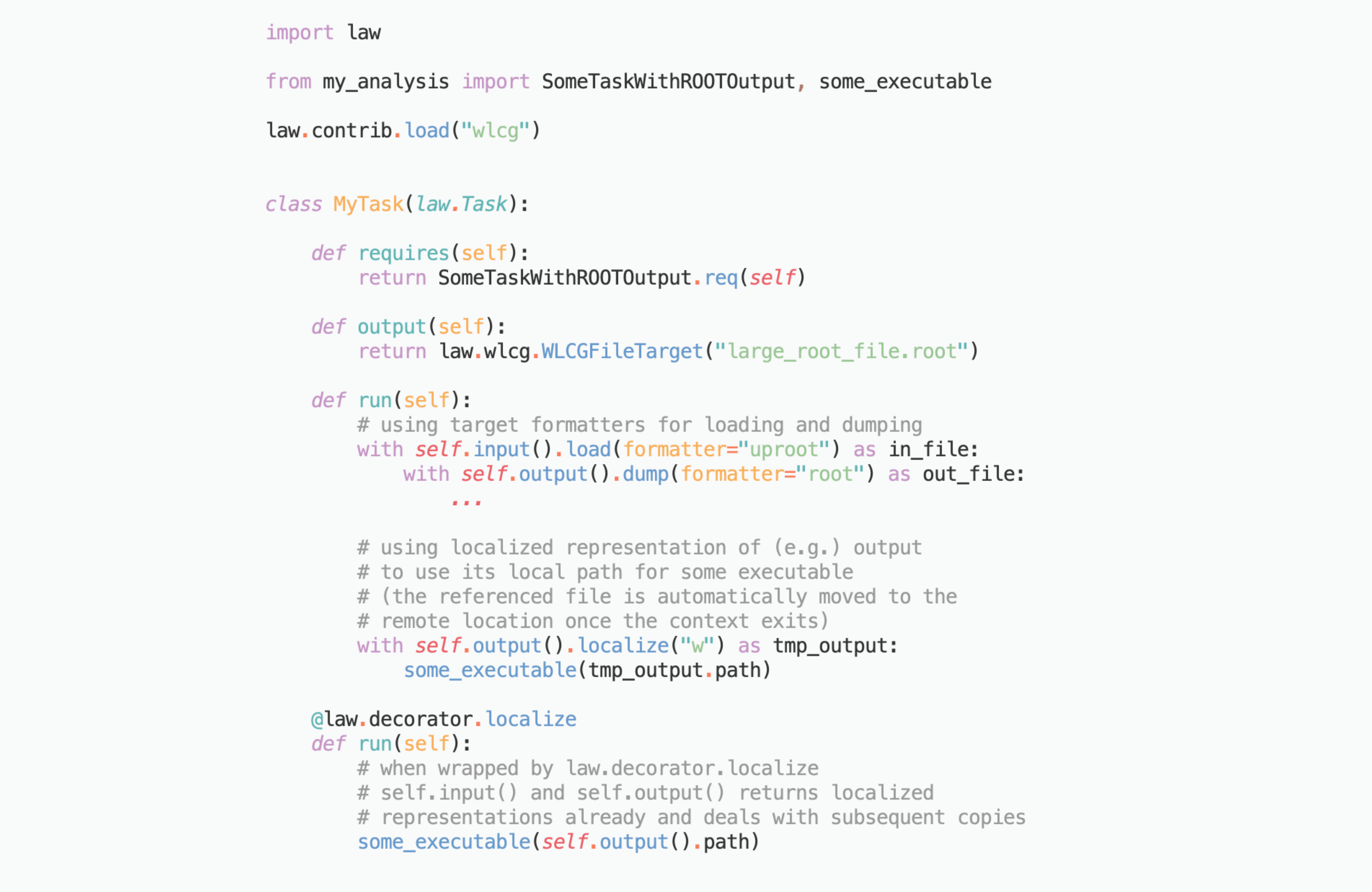
Table of contents

- Testing a datacard
  - Quick example
  - Dependencies
  - Parameters
  - Example commands
- Testing multiple datacards
  - Quick example
  - Dependencies
  - Parameters
  - Example commands

cms-hh.web.cern.ch/cms-hh/tools/inference

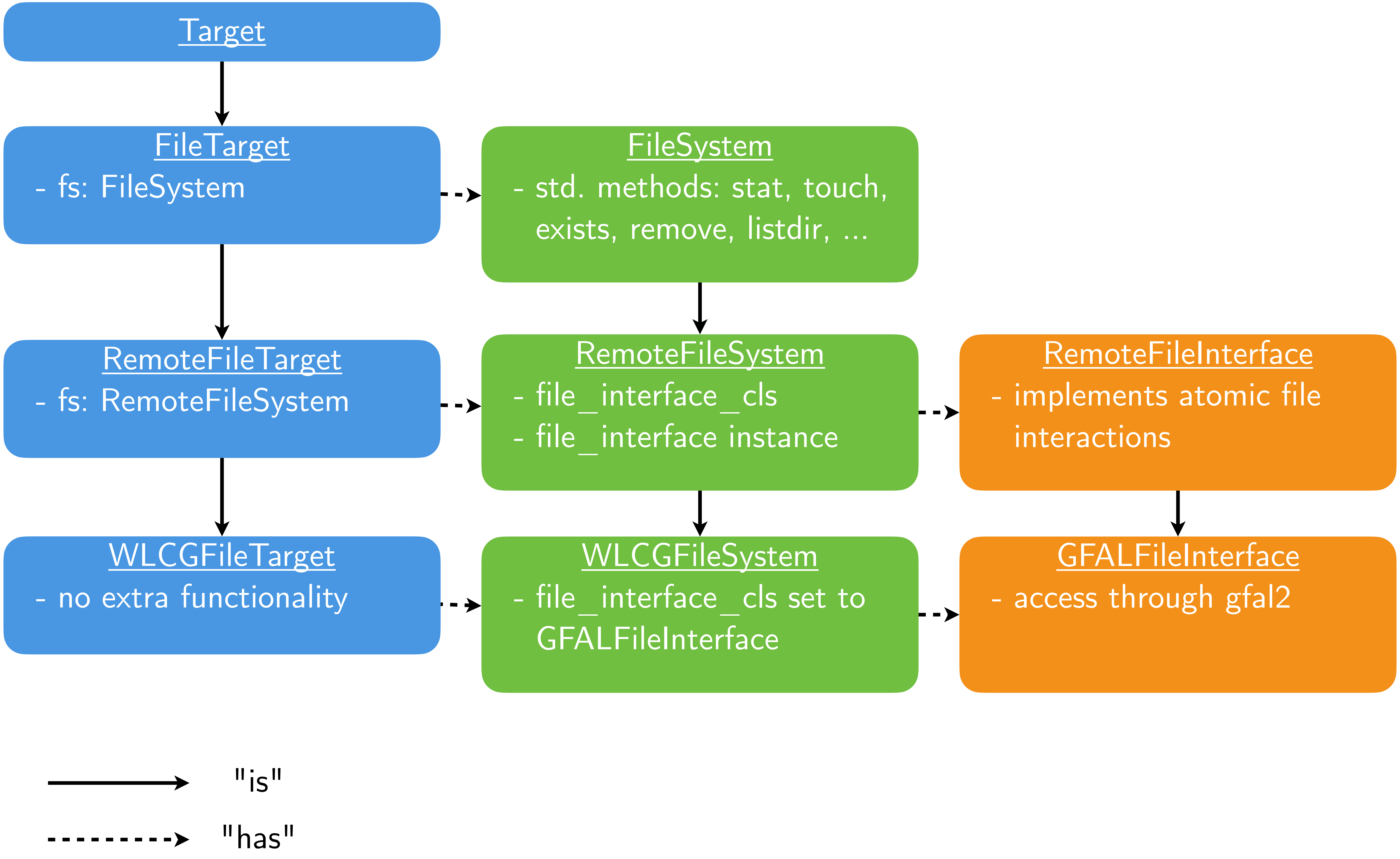
(Remote) targets

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)
```

Remote target implementation details



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

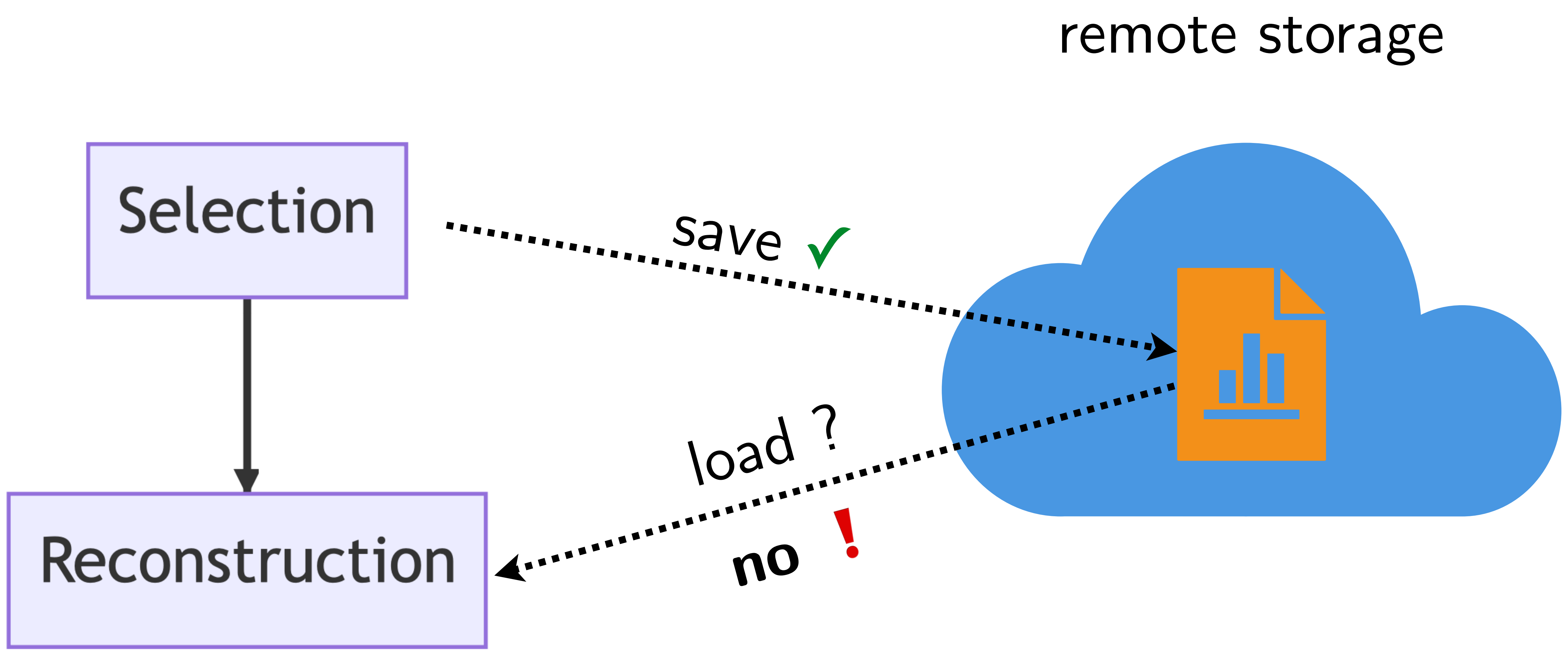
GFALFileInterface
- access through gfal2

"is"

"has"

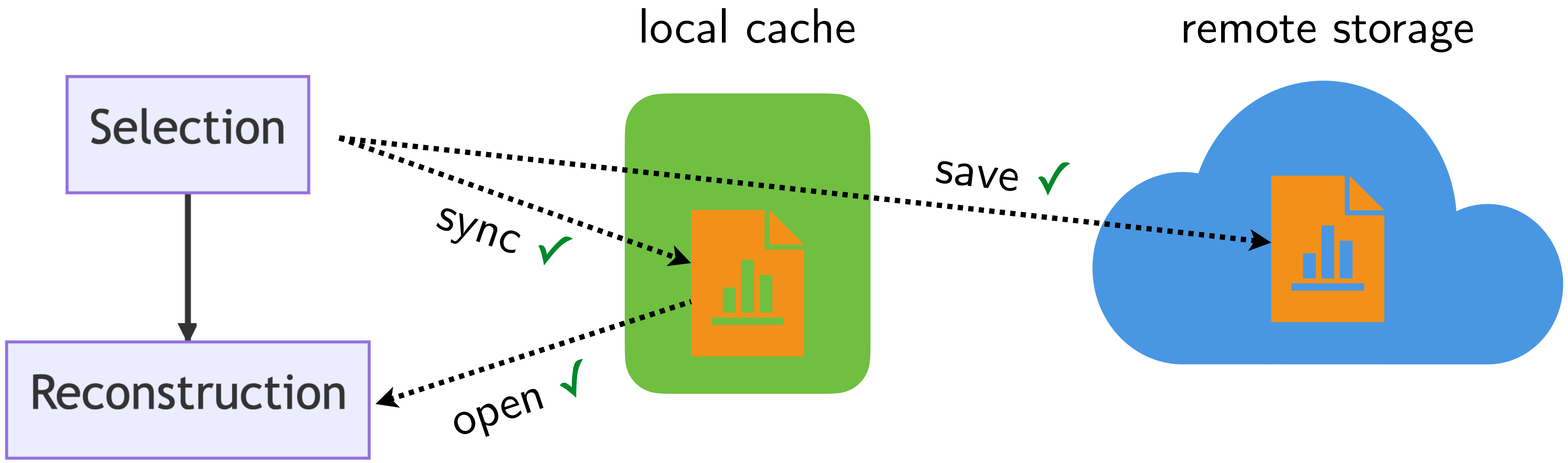
Effective remote targets — Caching

- **Local cache for remote targets**



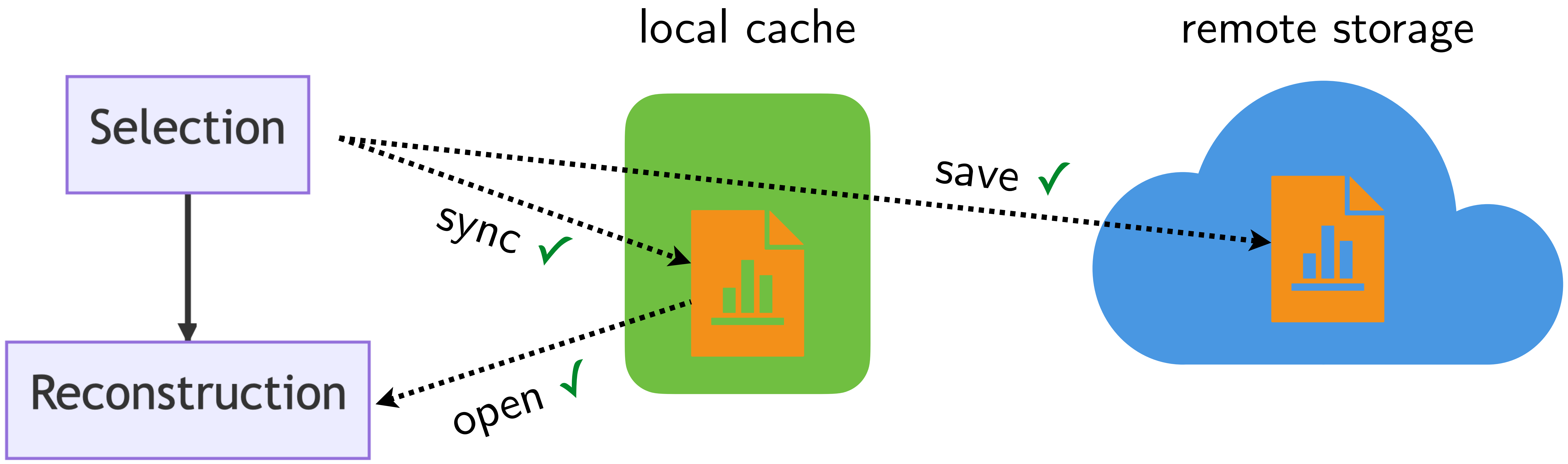
Effective remote targets — Caching

- **Local cache for remote targets**



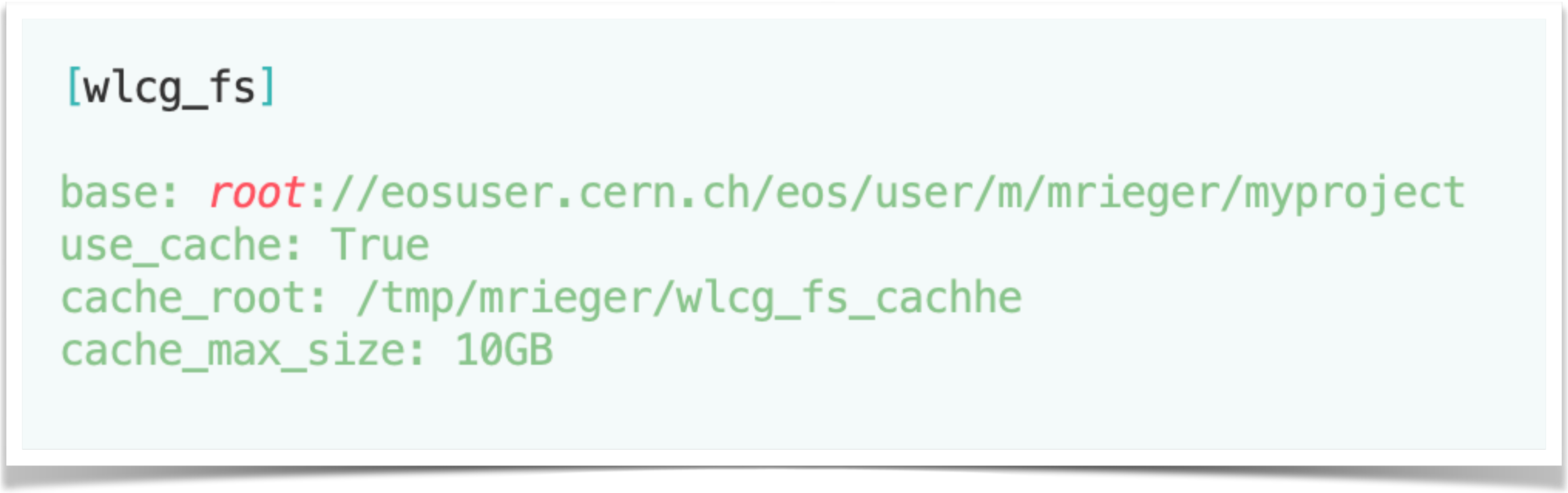
Effective remote targets — Caching

- **Local cache for remote targets**



- **Simple configuration**
  - When enabled, all operations on remote targets are cached

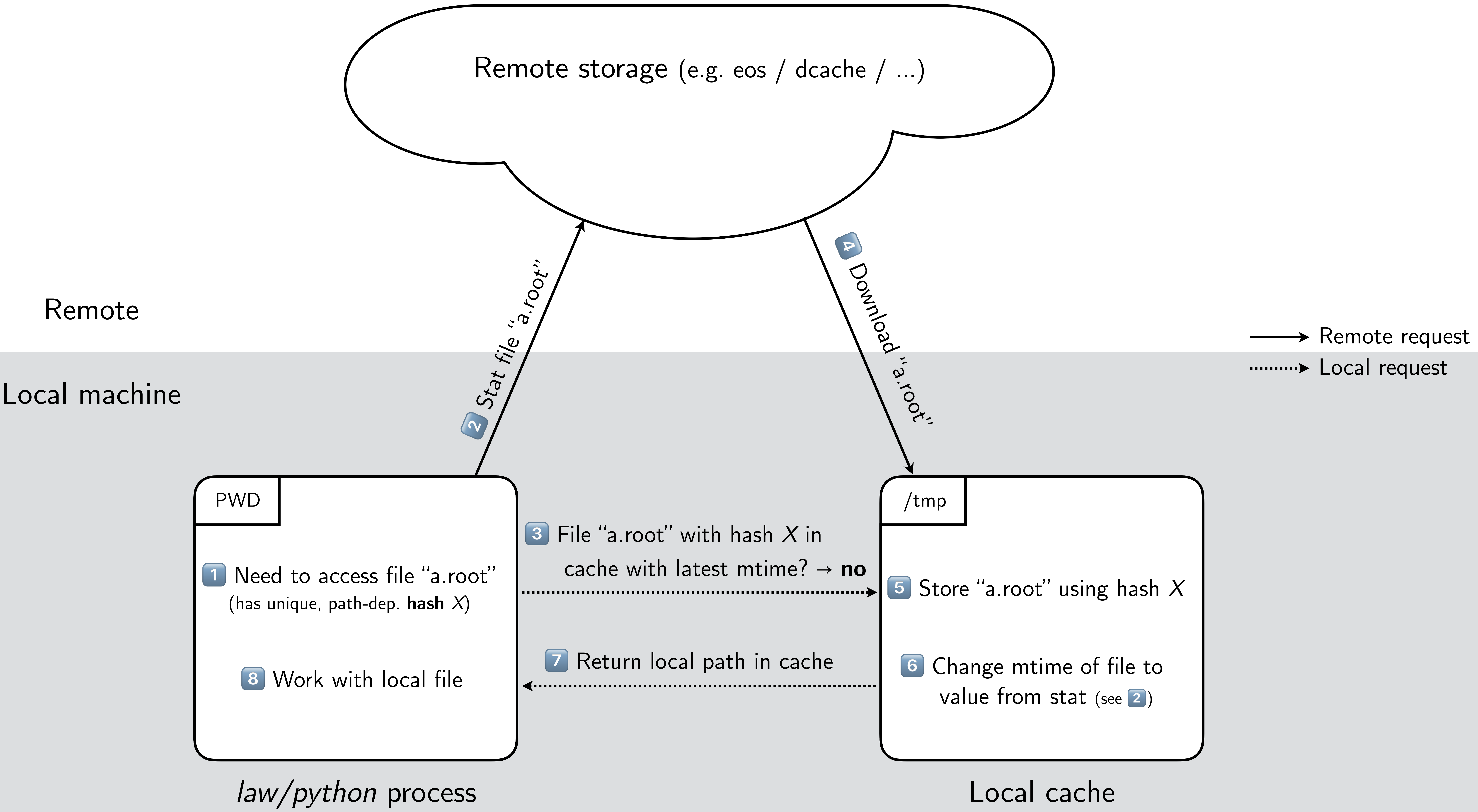
law.cfg



```
[wlcg_fs]

base: root://eosuser.cern.ch/eos/user/m/mrieger/myproject
use_cache: True
cache_root: /tmp/mrieger/wlcg_fs_cachhe
cache_max_size: 10GB
```

Local caching (1)

Configuration ☞



Remote storage (e.g. eos / dcache / ...)

Remote

Local machine

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

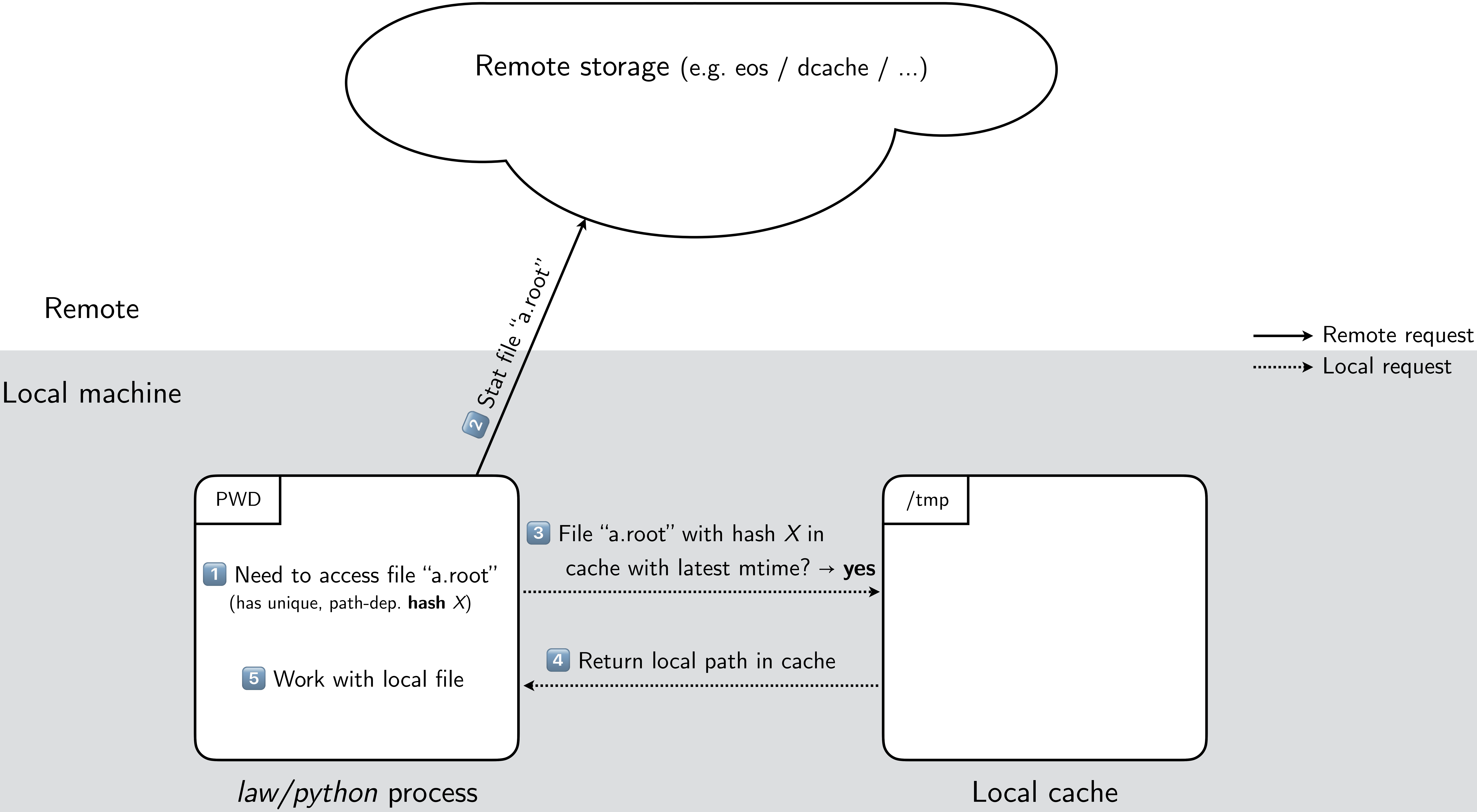
law/python process

Local cache

⟶ Remote request
⇢ Local request

Local caching (2)

Configuration ☞



Remote storage (e.g. eos / dcache / ...)

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? → **yes**

4 Return local path in cache

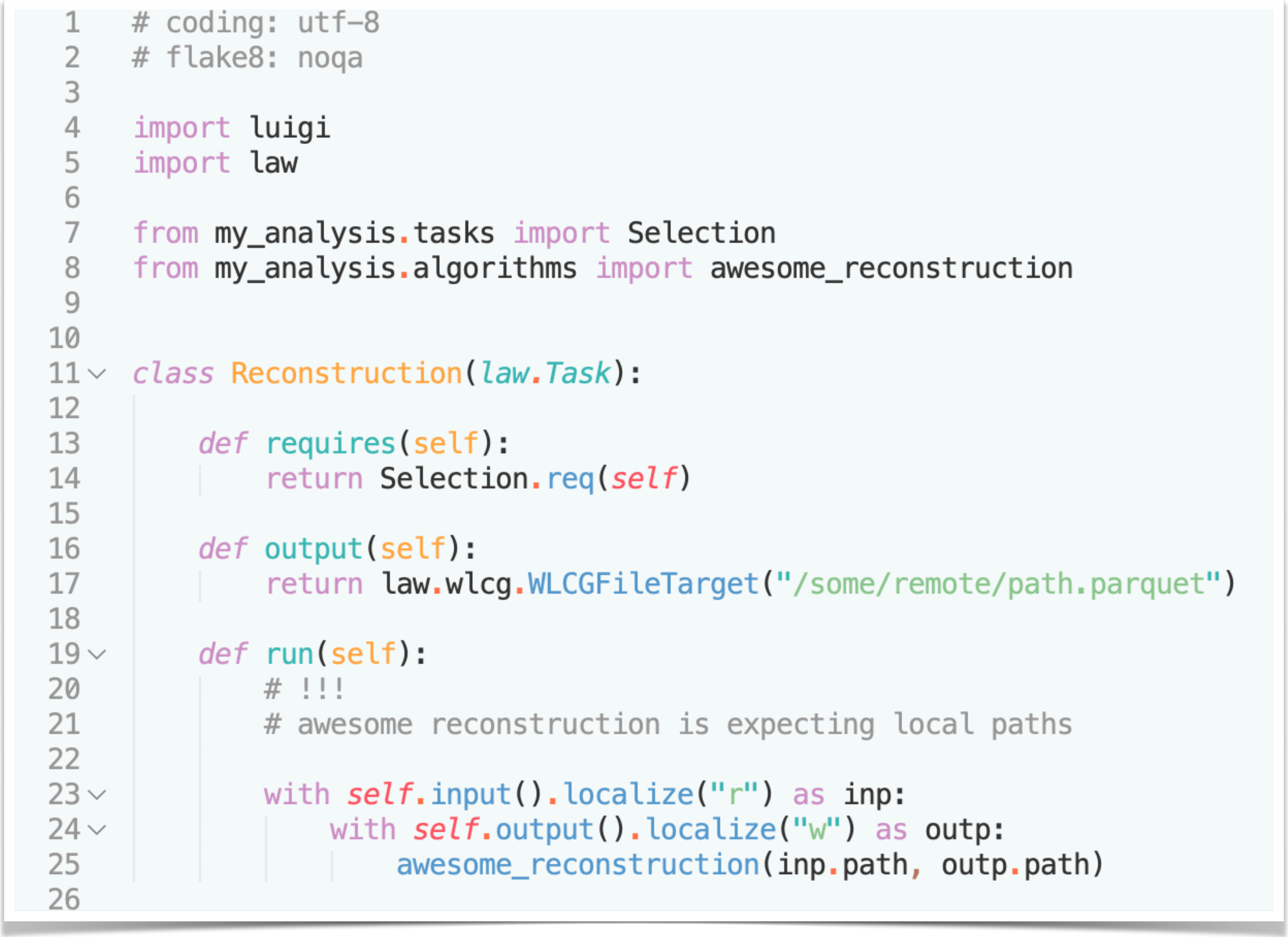
5 Work with local file

/tmp

law/python process

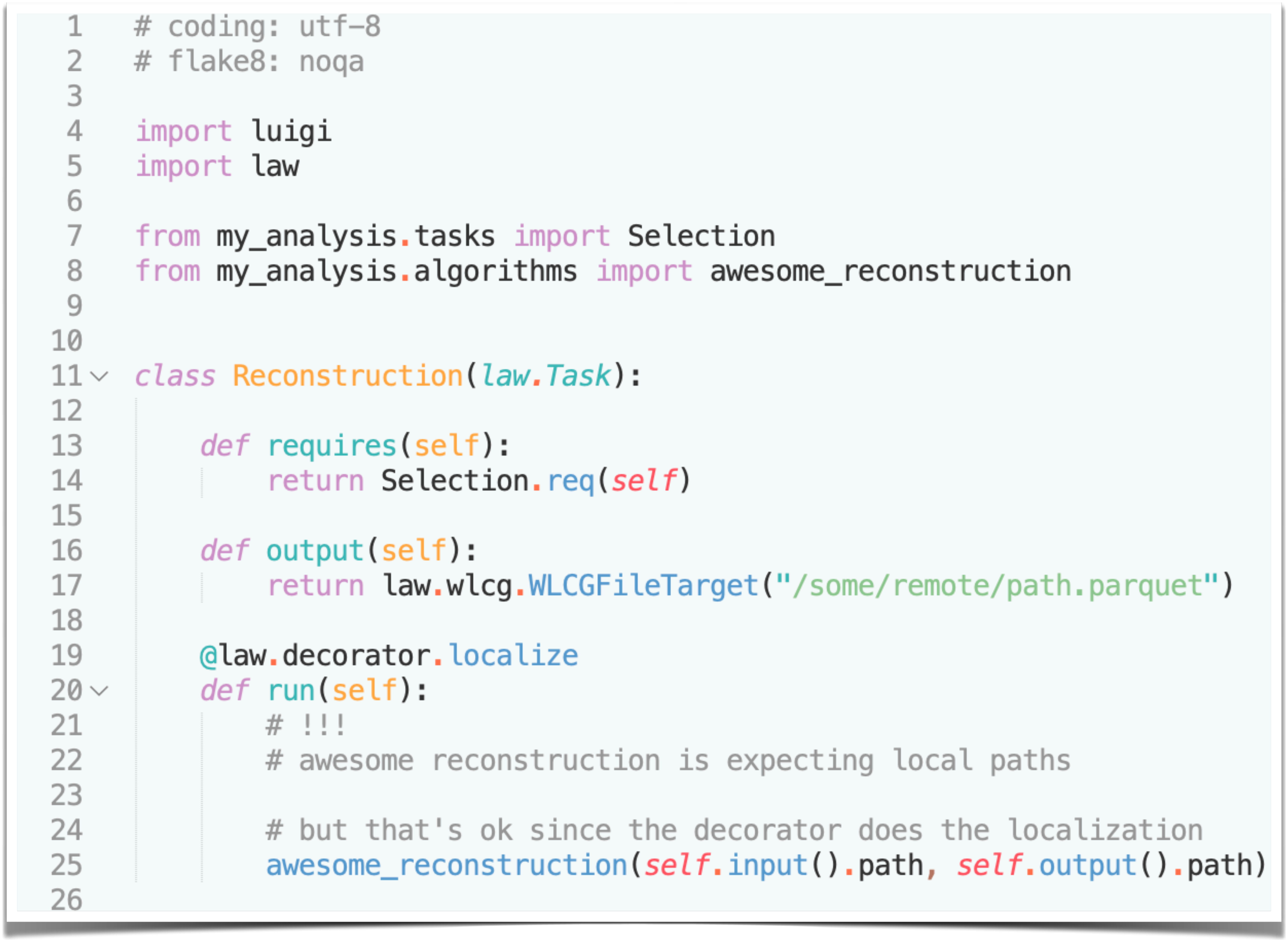
Local cache

Effective remote targets — "Localization"



```python
# coding: utf-8
# flake8: noqa

import luigi
import law

from my_analysis.tasks import Selection
from my_analysis.algorithms import awesome_reconstruction


class Reconstruction(law.Task):

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

    def output(self):
        return law.wlcg.WLCGFileTarget("/some/remote/path.parquet")

    def run(self):
        # !!!
        # awesome reconstruction is expecting local paths

        with self.input().localize("r") as inp:
            with self.output().localize("w") as outp:
                awesome_reconstruction(inp.path, outp.path)

```

Effective remote targets — "Localization"



```python
# coding: utf-8
# flake8: noqa

import luigi
import law

from my_analysis.tasks import Selection
from my_analysis.algorithms import awesome_reconstruction


class Reconstruction(law.Task):

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

    def output(self):
        return law.wlcg.WLCGFileTarget("/some/remote/path.parquet")

    @law.decorator.localize
    def run(self):
        # !!!
        # awesome reconstruction is expecting local paths

        # but that's ok since the decorator does the localization
        awesome_reconstruction(self.input().path, self.output().path)

```

Workflows

Workflows: General ideas

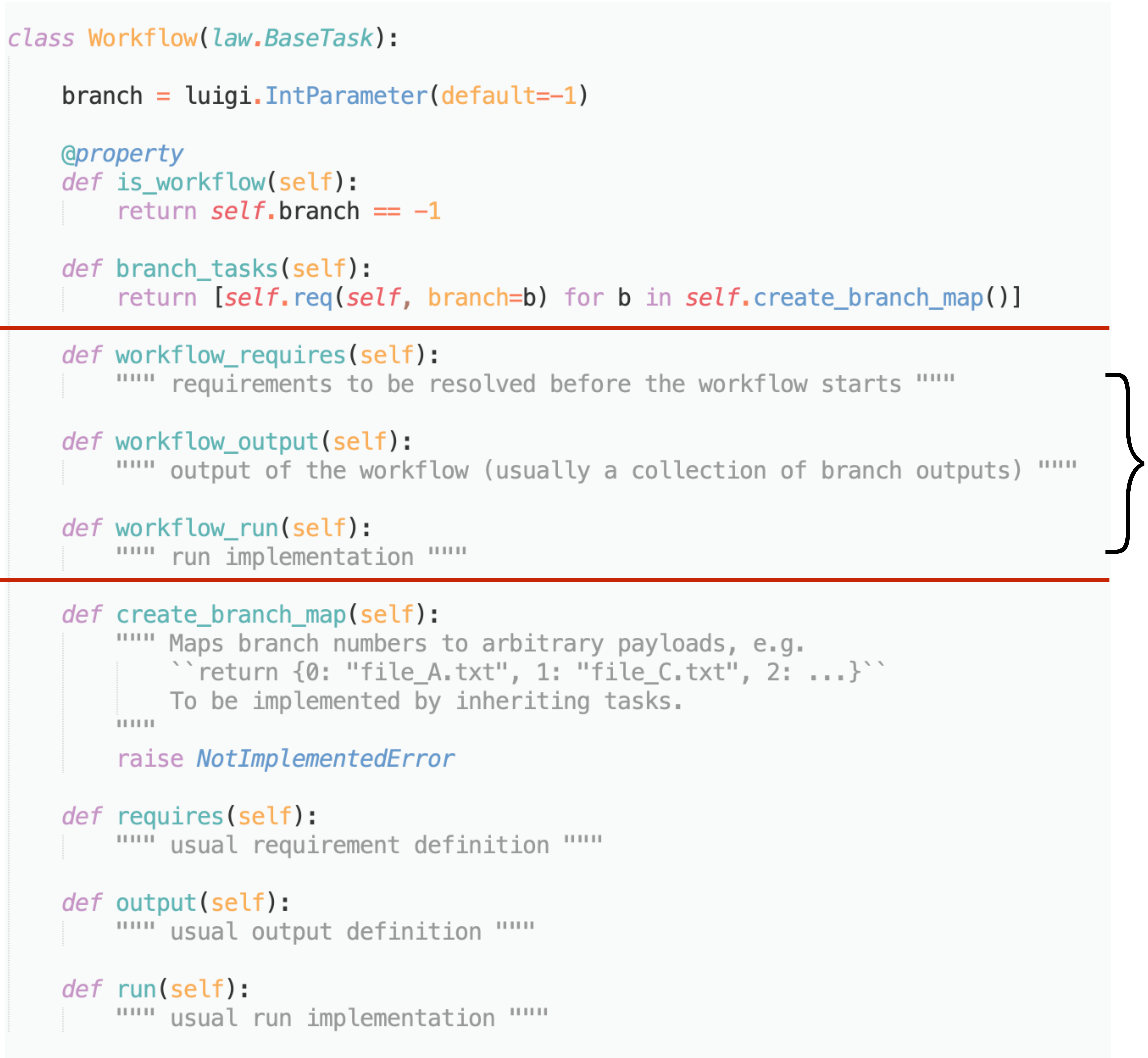
- **Many tasks exhibit the same overall structure and/or purpose**
  - *"Run over N existing files"* / *"Generate N events/toys"* / *"Merge N into M files"*
  - All these tasks can **profit from the same features**
    - ▻ *"Only process file x and/to y", "Remove outputs of "x, y & z", "Process N files,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



```
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 """
```

Common

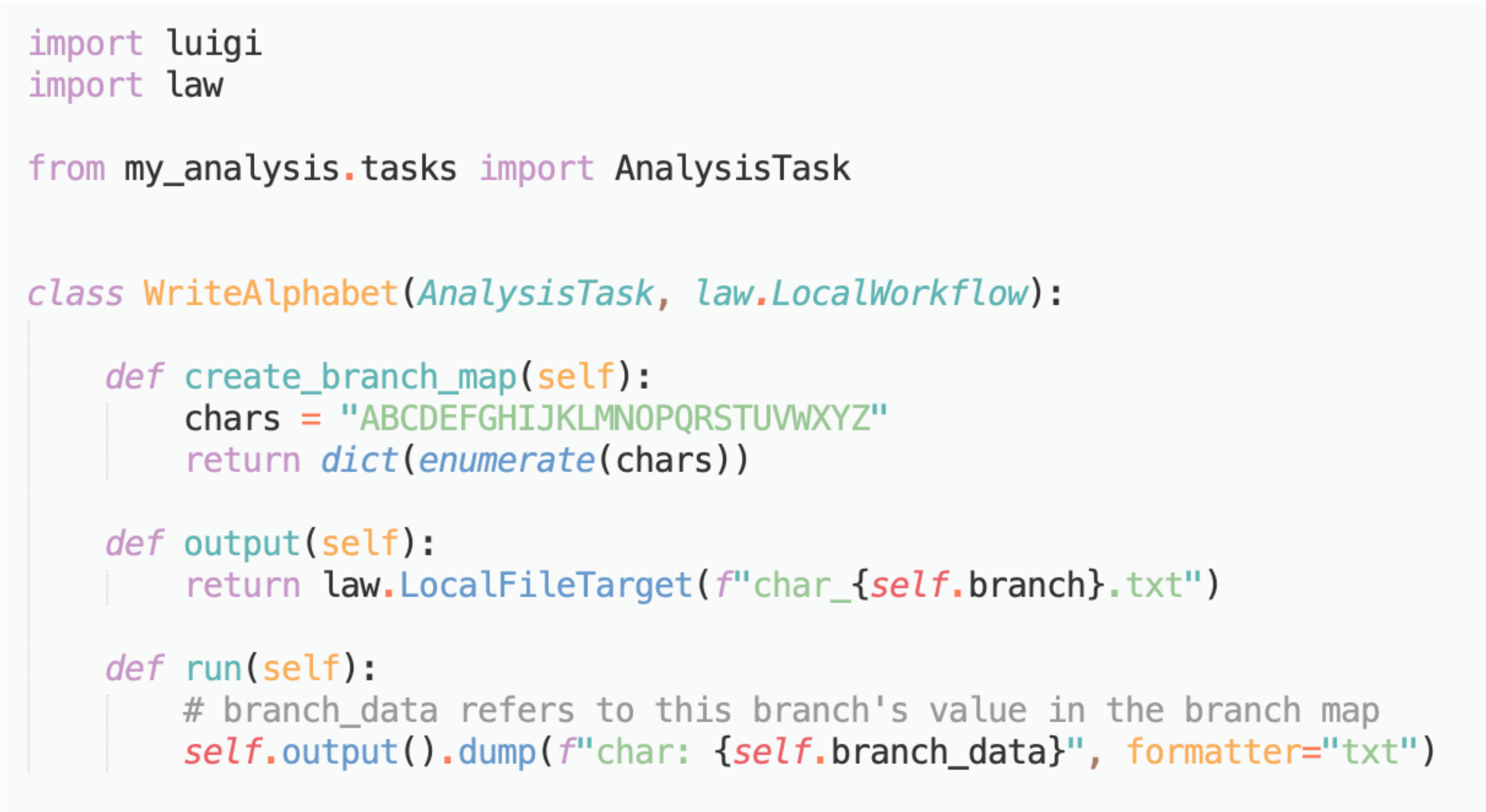
Workflow specific

Implemented by task

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

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

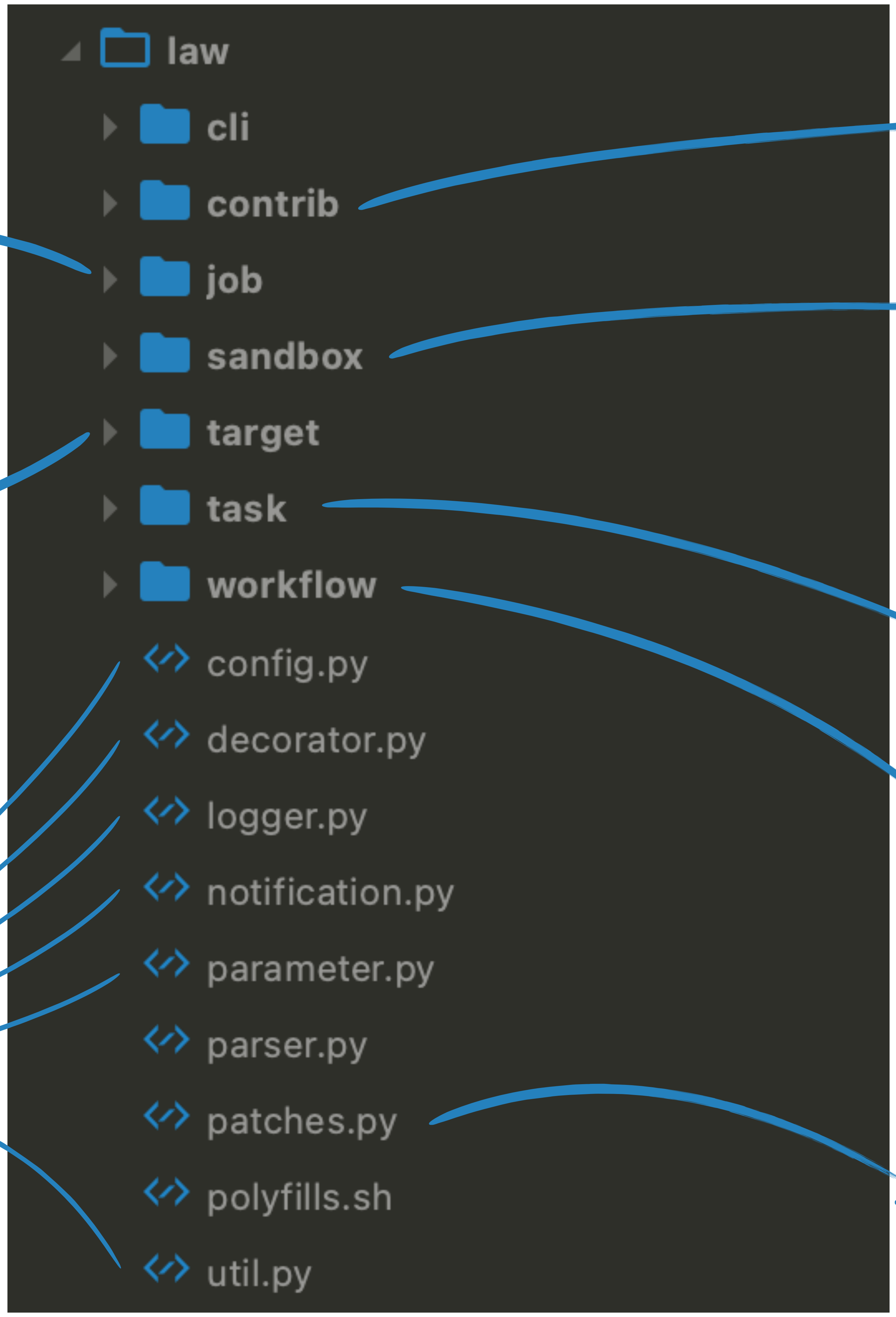
- 6 **remote workflow implementations come with law**
  - htcondor, glite, lsf, arc, slurm, cms-crab (in PR#150)
  - Based on generic "job manager" implementations in contrib packages

- **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:
    - `--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

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

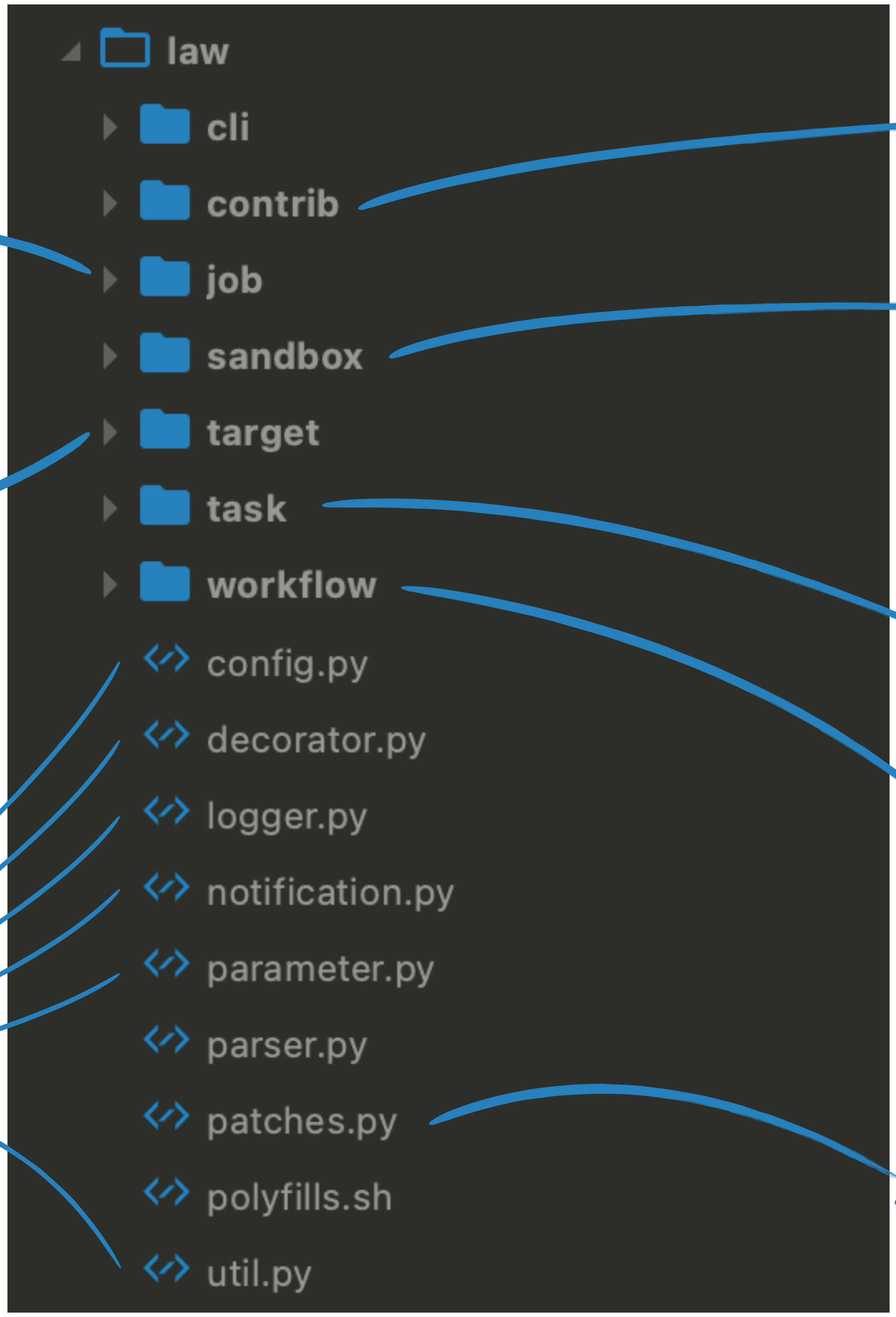
Job interface

- 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
```

S

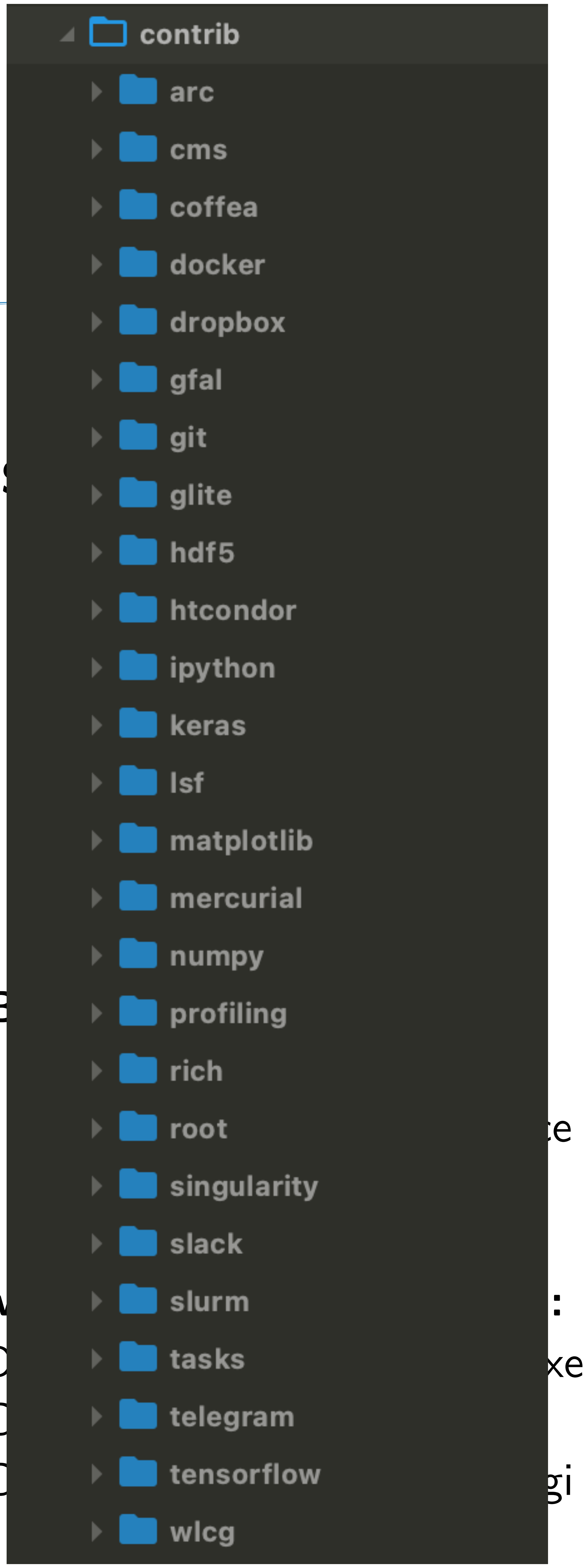
E

e

Lightw :

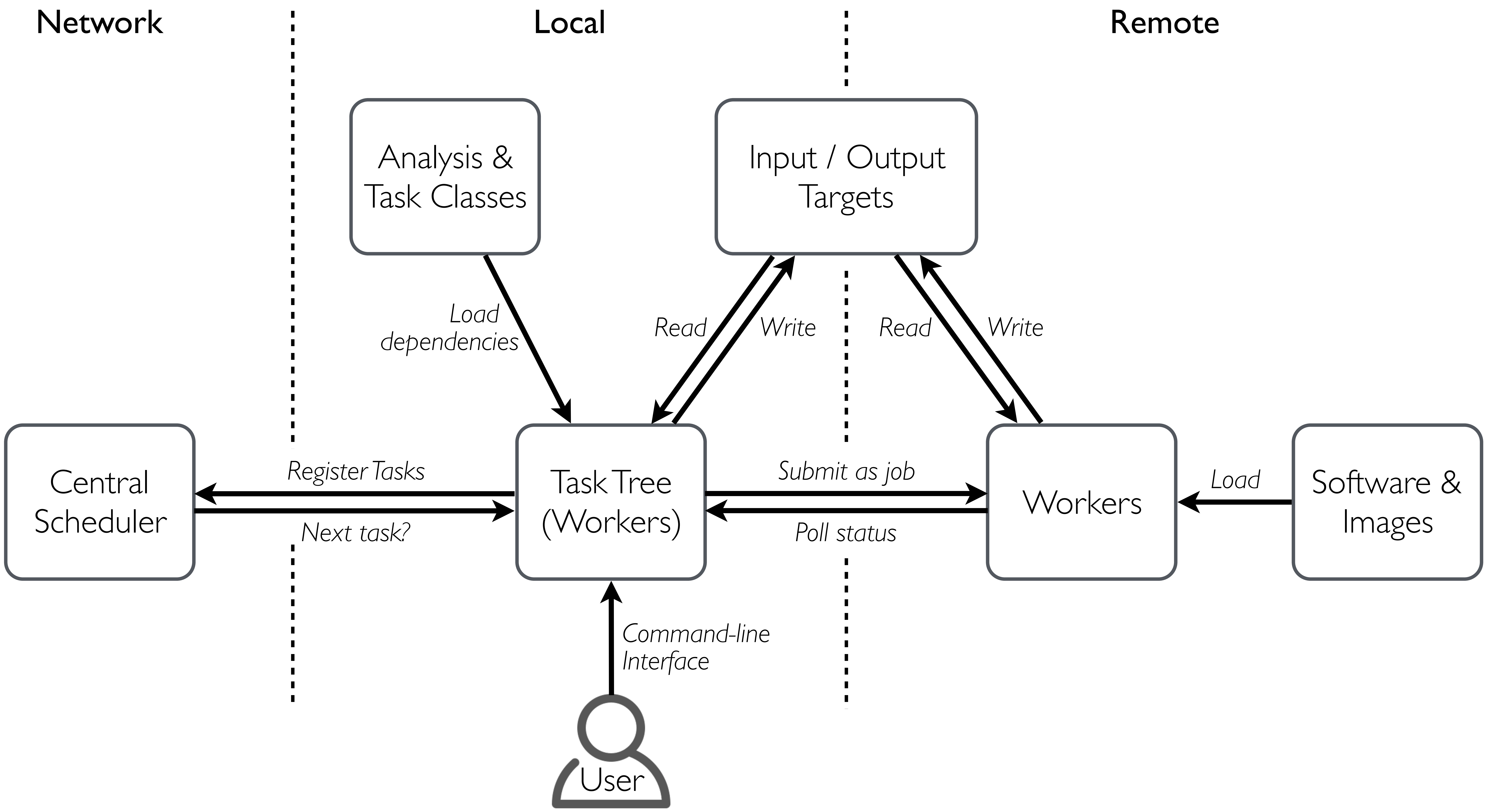
- D xes
- C

→ C gi

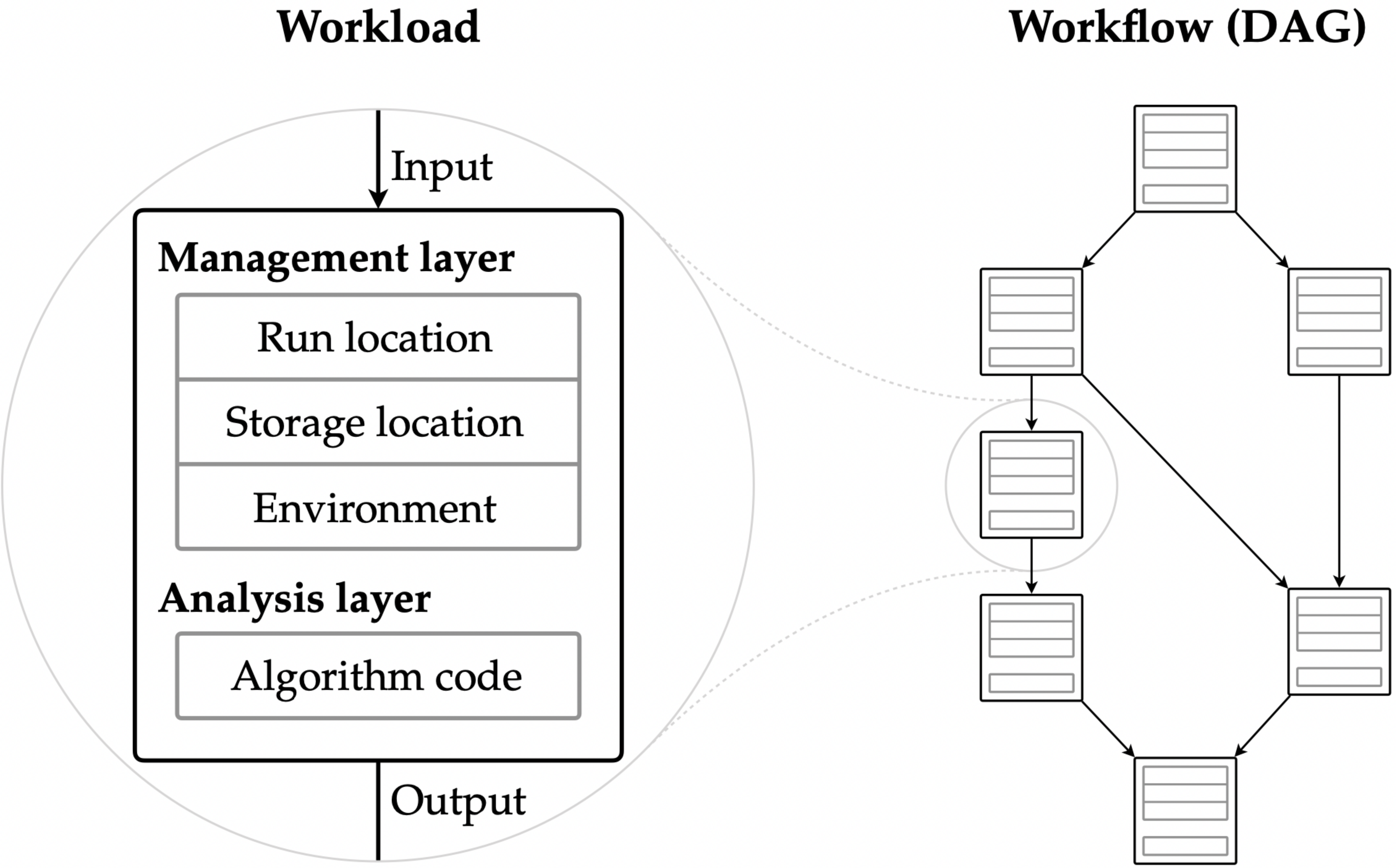


```
contrib
  arc
  cms
  coffea
  docker
  dropbox
  gfal
  git
  glite
  hdf5
  htcondor
  ipython
  keras
  lsf
  matplotlib
  mercurial
  numpy
  profiling
  rich
  root
  singularity
  slack
  slurm
  tasks
  telegram
  tensorflow
  wlcg
```

luigi/law architecture



DAG abstraction



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