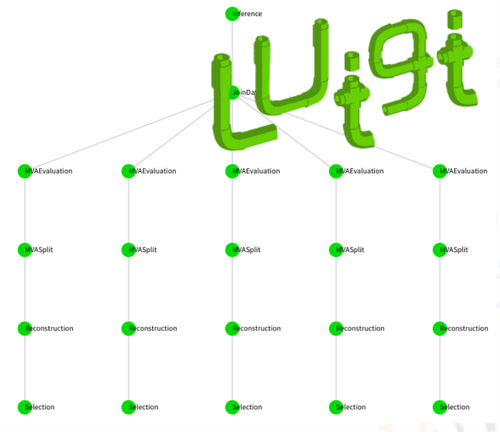


law

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

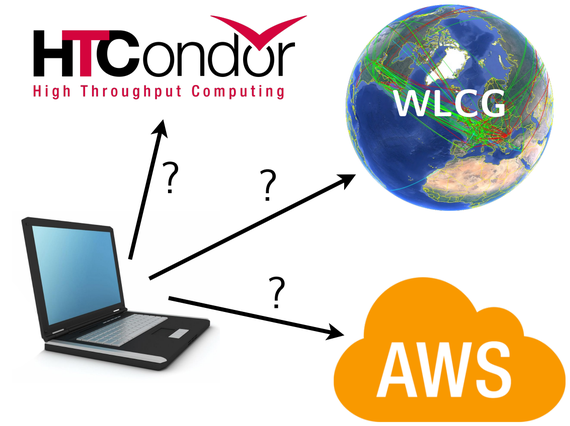
luigi analysis workflow

— Large Scale Analysis Automation over Distributed Resources —

Marcel Rieger

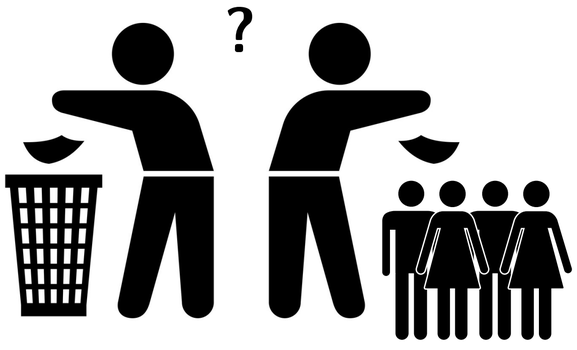
IRIS-HEP Topical Meeting

12.12.2022



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

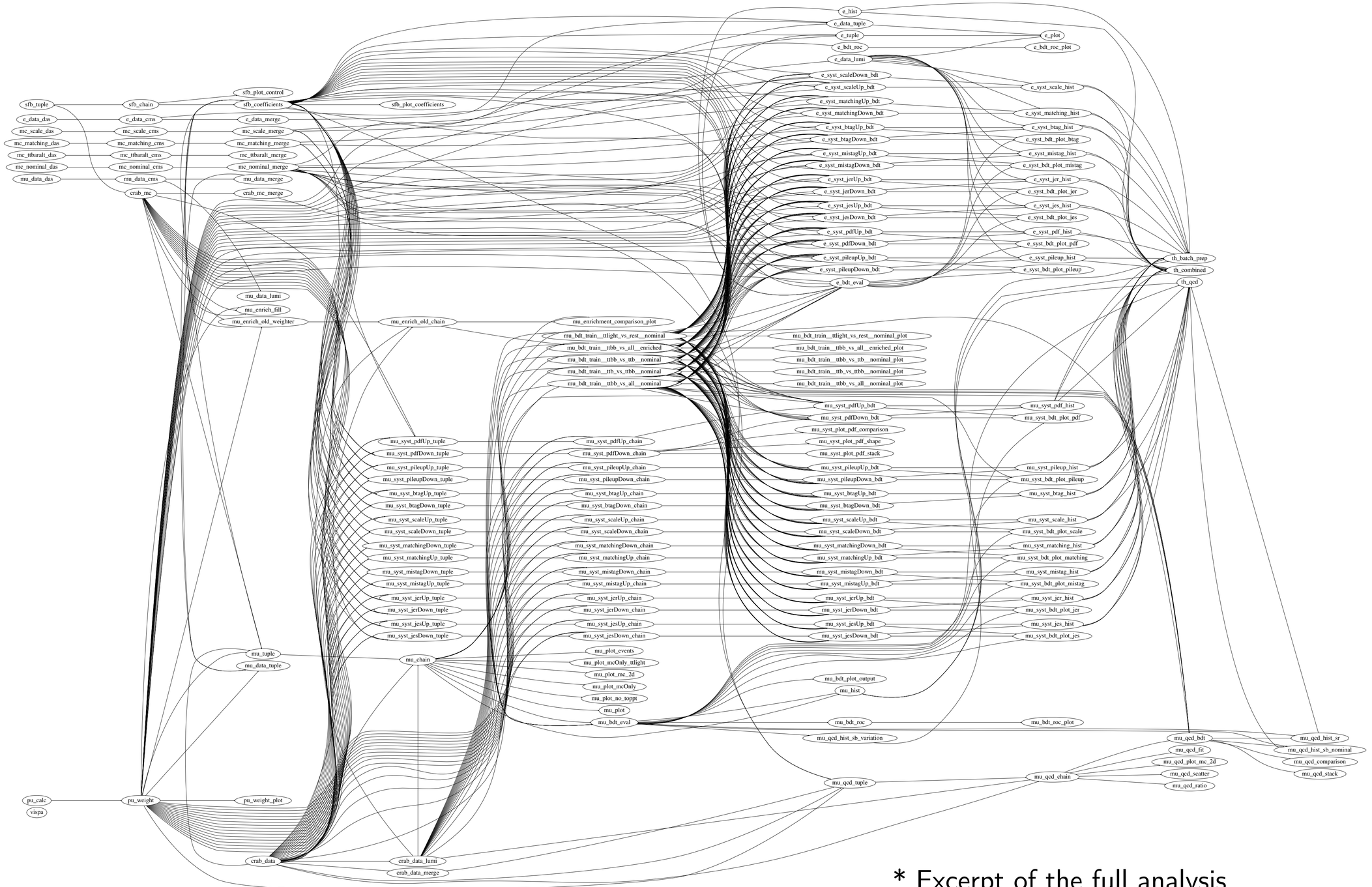
- **Reproducibility**: When a postdoc / PhD student leaves, ...
  - can someone else run the analysis?
  - is there a loss of information? Is a new *framework* required?
    - ▻ Dependencies often **only** exist in the physicists head!



- **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**!

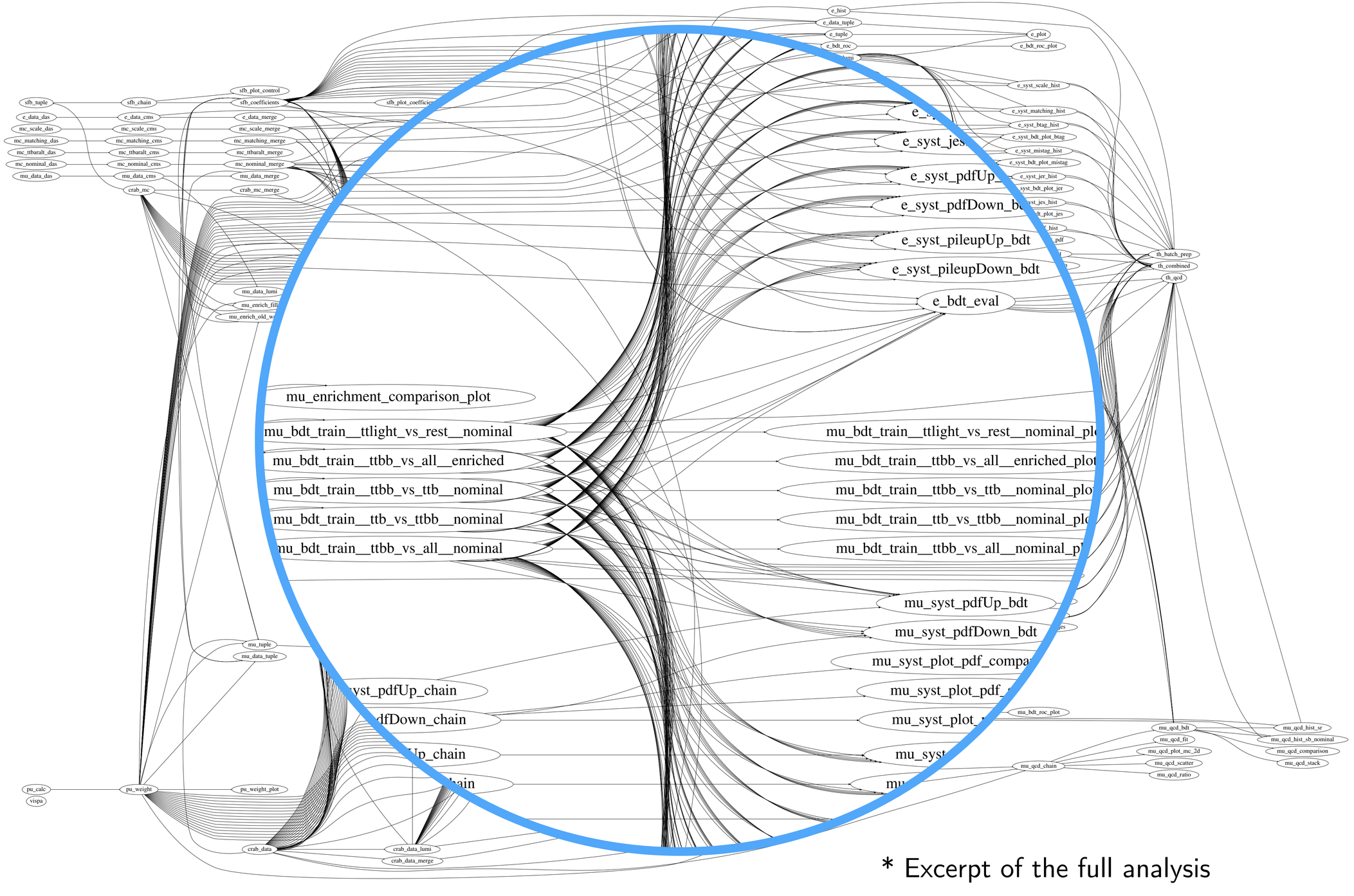
- Personal experience: ⅔ of "analysis" time for technicalities, ⅓ left for physics
  → **Physics output doubled if it were the other way round?**

Example: ttbb cross section measurement



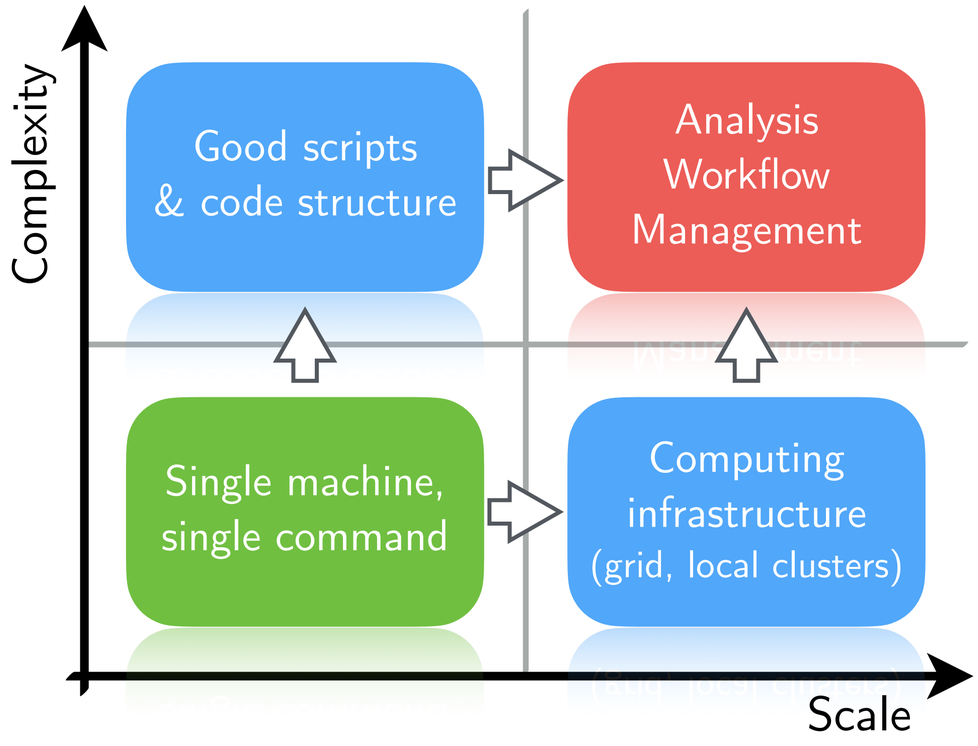
* Excerpt of the full analysis

Example: ttbb cross section measurement



* Excerpt of the full analysis

- 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



- **In the following**
  - → Approach **complexity** with luigi
  - → Enabling **large-scale** with law

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

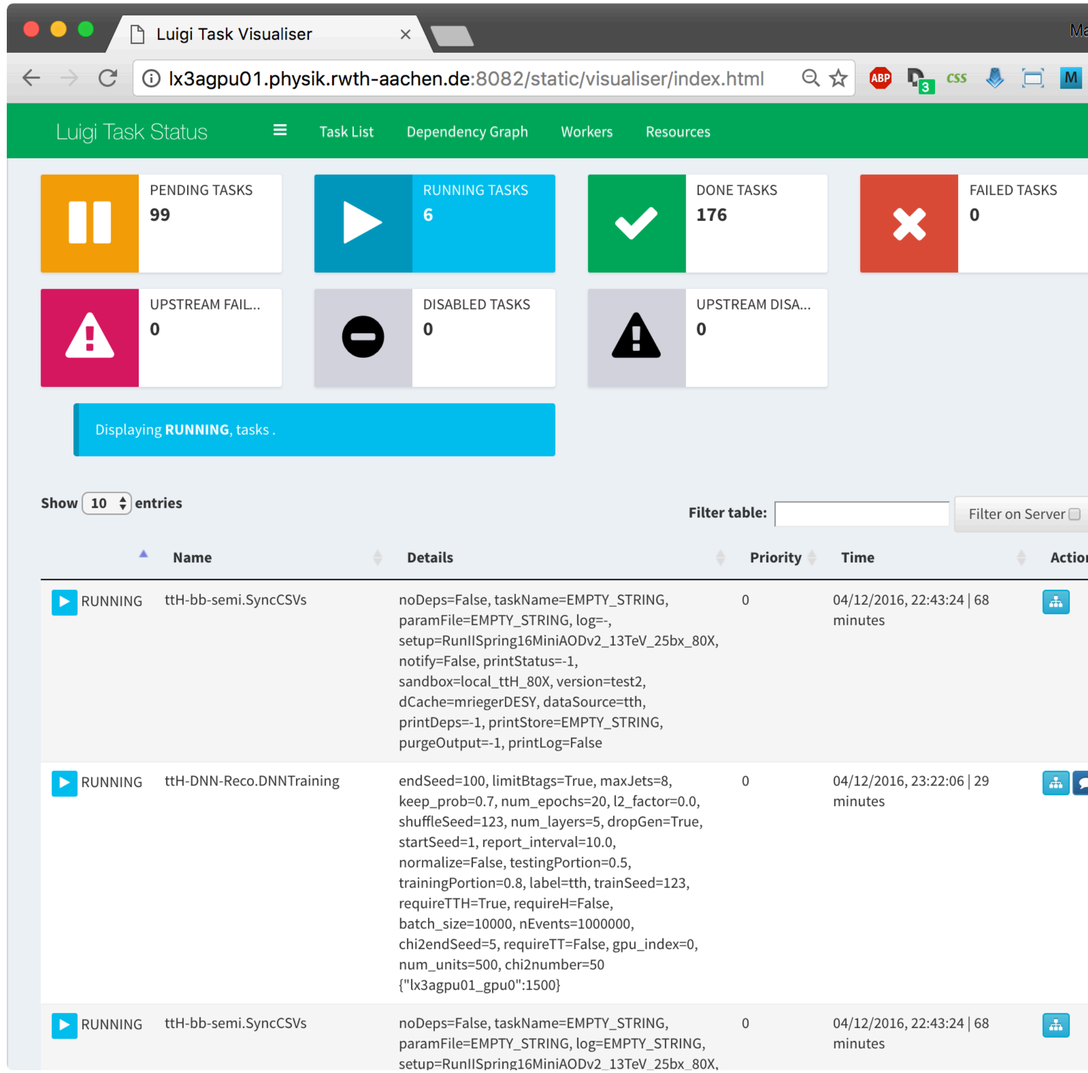
→ Requirements for HEP analyses mostly orthogonal

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



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, ...

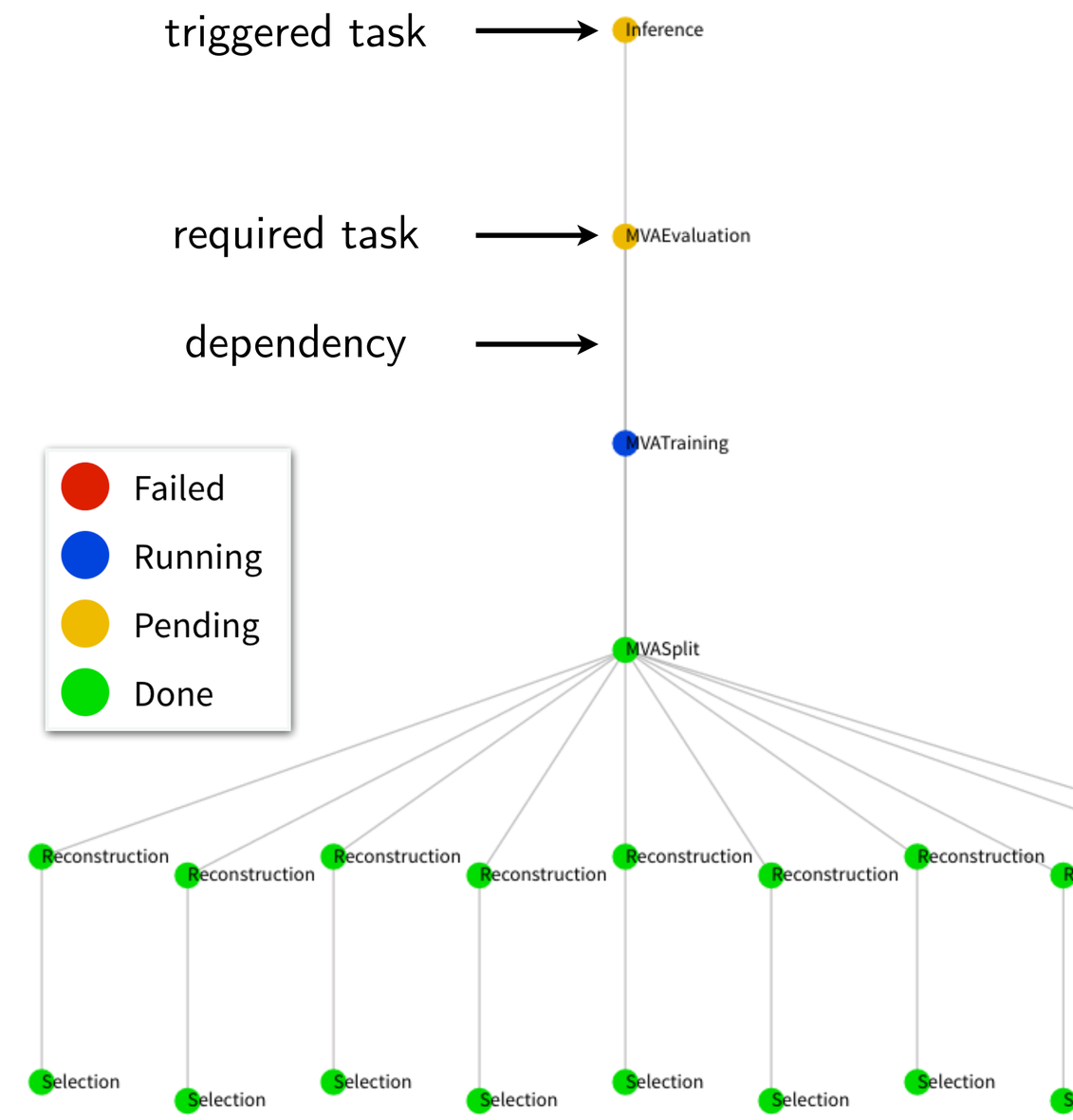
make-like execution system

- 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`



```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"
        ...
```

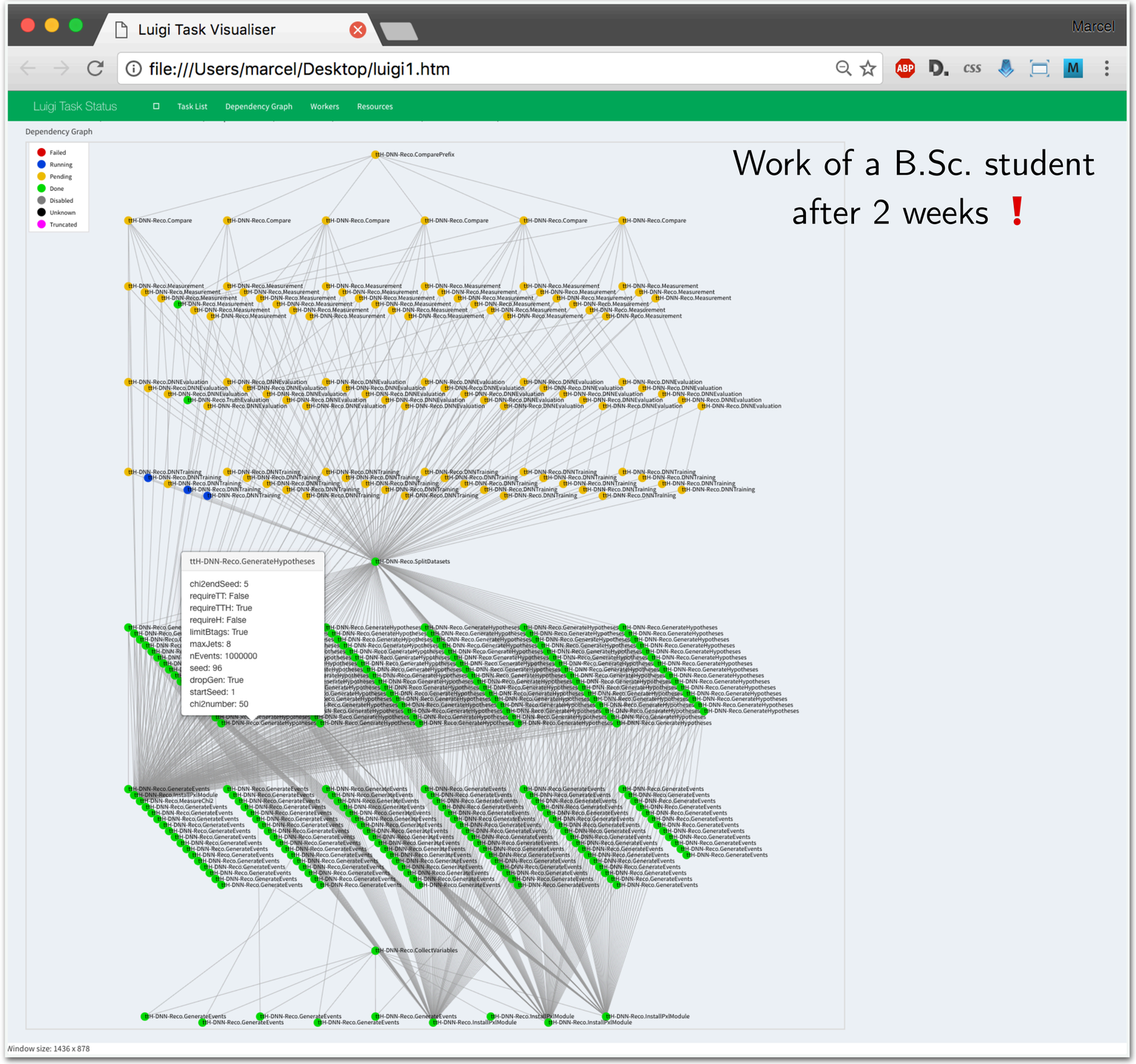
`Parameter` object on class-level

`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
```


Luigi Task Visualiser
Marcel
file:///Users/marcel/Desktop/luigi1.htm
Luigi Task Status
Task List
Dependency Graph
Workers
Resources
Dependency Graph
Failed
Running
Pending
Done
Disabled
Unknown
Truncated
Work of a B.Sc. student
after 2 weeks !
ttH-DNN-Reco.ComparePrefix
ttH-DNN-Reco.Compare
ttH-DNN-Reco.Measurement
ttH-DNN-Reco.DNNEvaluation
ttH-DNN-Reco.DNNTraining
ttH-DNN-Reco.SplitDatasets
ttH-DNN-Reco.GenerateHypotheses
ttH-DNN-Reco.GenerateEvents
ttH-DNN-Reco.InstallPxlModule
ttH-DNN-Reco.MeasureChi2
ttH-DNN-Reco.CollectVariables
ttH-DNN-Reco.TruthEvaluation
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
Window size: 1436 x 878

law

luigi analysis workflow

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

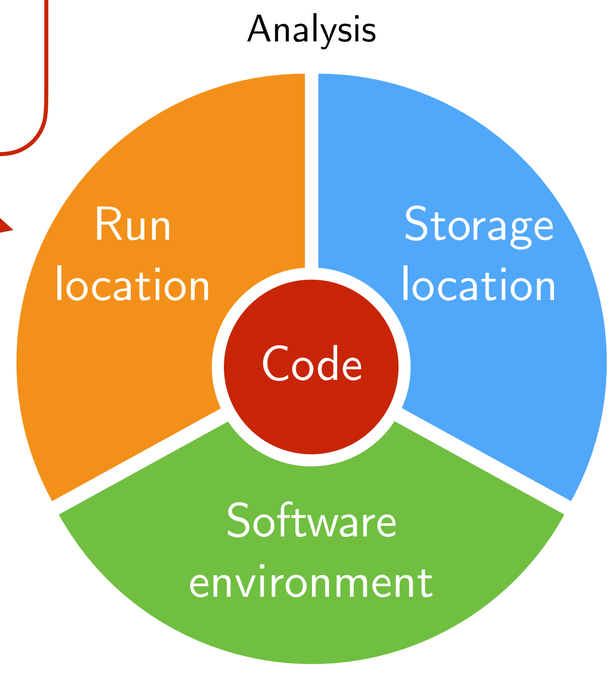
- Software design follows 3 primary goals:

  1. Experiment-agnostic core (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



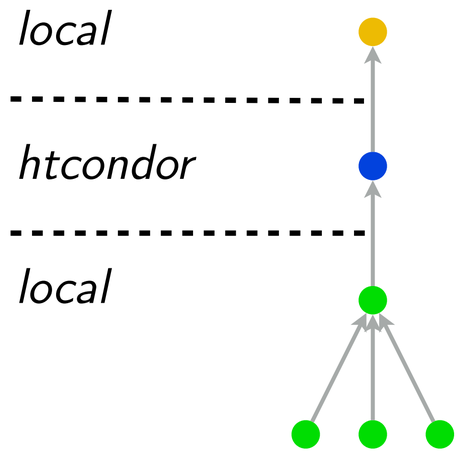
- Toolbox to follow an **analysis design pattern**
  - No constraint on language or data structures
  - → Not a *framework*

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

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 >
```



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)
```

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)
```

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, ...

```python
def run(self):
    # get the input to this task, which is a *.gz file
    # (the output of the requirements)
    inp = self.input()

    # create the correction set
    import correctionlib
    correction_set = correctionlib.CorrectionSet.from_string(
        inp.load(formatter="gzip"),
    )

    ...
```

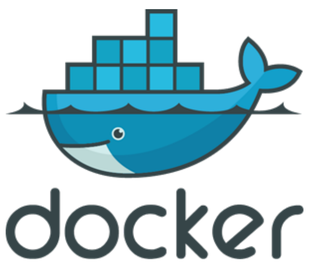
downloads the file if it is remote → *puts it into the local cache for later use* → *opens the file and decompresses the content*

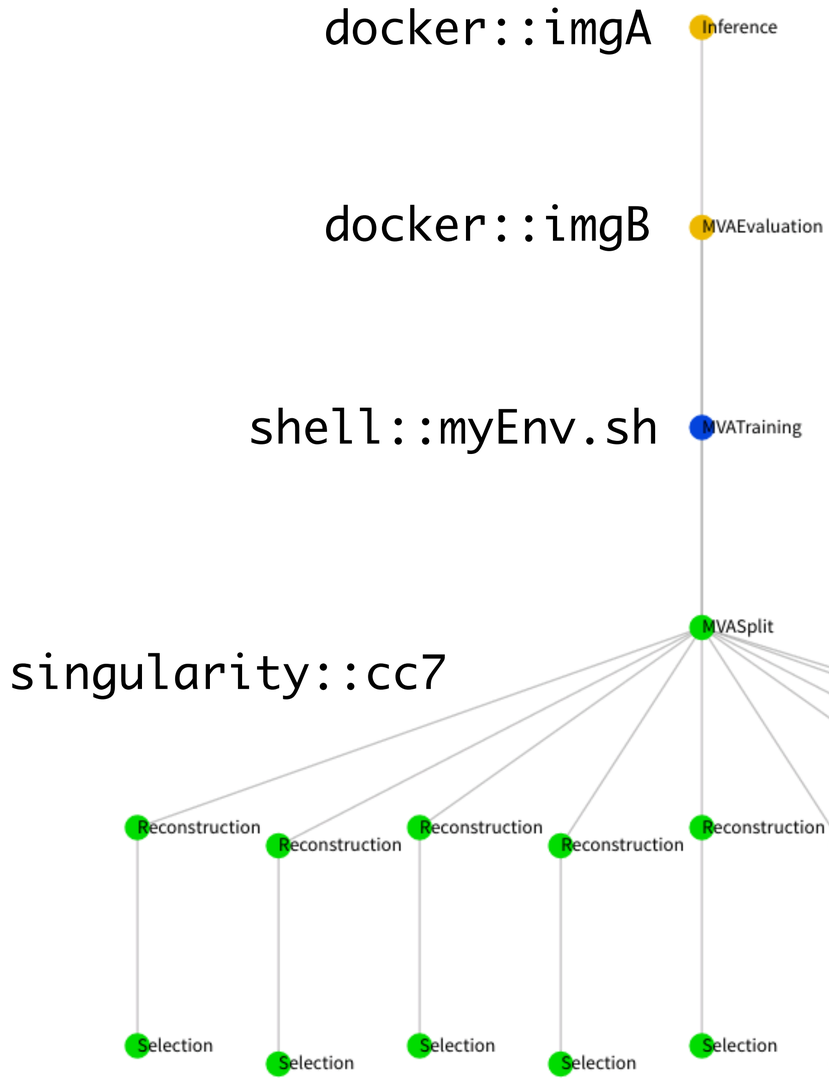
3. Environment sandboxing

- 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





```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
```

- **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

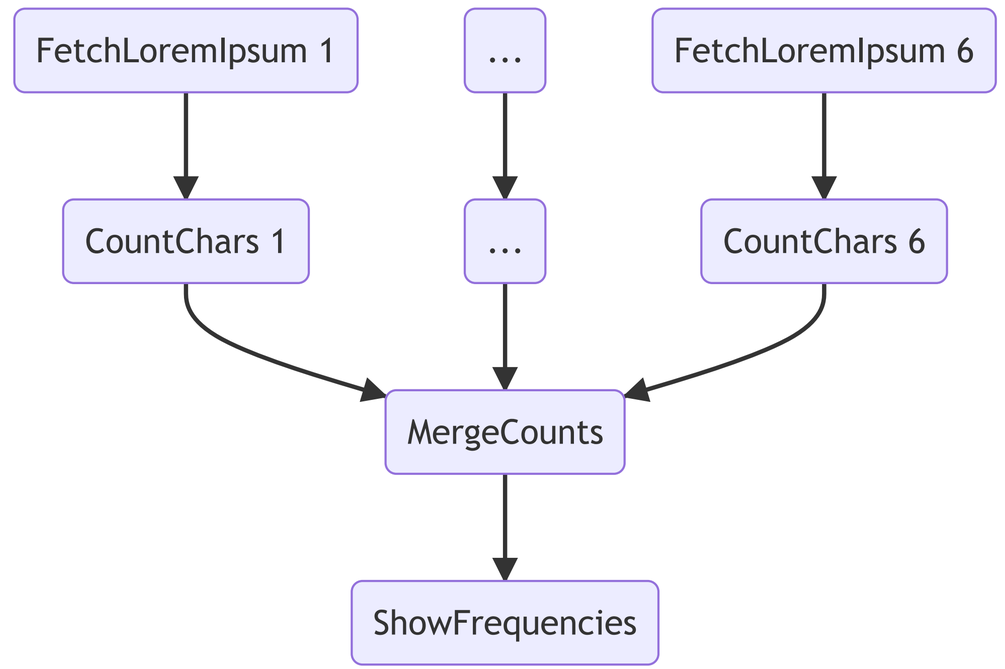
```python
from analysis.tasks import Selection
import akward as ak

# 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**

```
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

- Print character frequencies in the "loremipsum" placeholder text (from examples/loremipsum)
  - ▻ Fetch 6 paragraphs as txt files from some server
  - ▻ Count character frequencies and save them in json
  - ▻ Merge into a single json file
  - ▻ Print frequencies



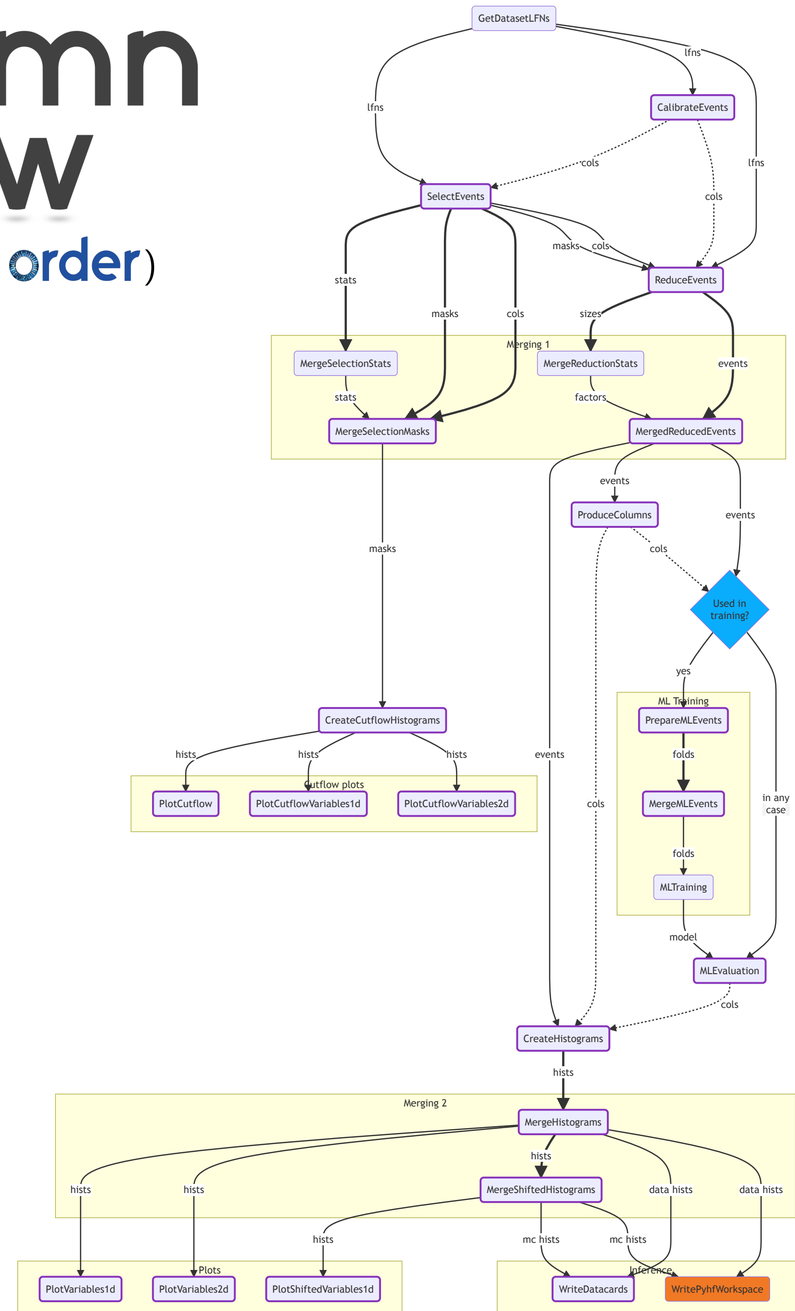
- Sowing CLI usage in the following, but launch binder for the notebook version

Example: large scale analyses

- **columnflow**: Backend for large-scale columnar analyses
  - Reads and writes columns only if necessary
  - Creates new columns and merges with existing ones at the **latest possible instance**
  - Stores intermediate outputs for
    - ▻ computations downstream
    - ▻ sharing results of same computations across groups
    - ▻ applications requiring per-event info (ML)
    - ▻ studies done by students
    - ▻ debugging purposes

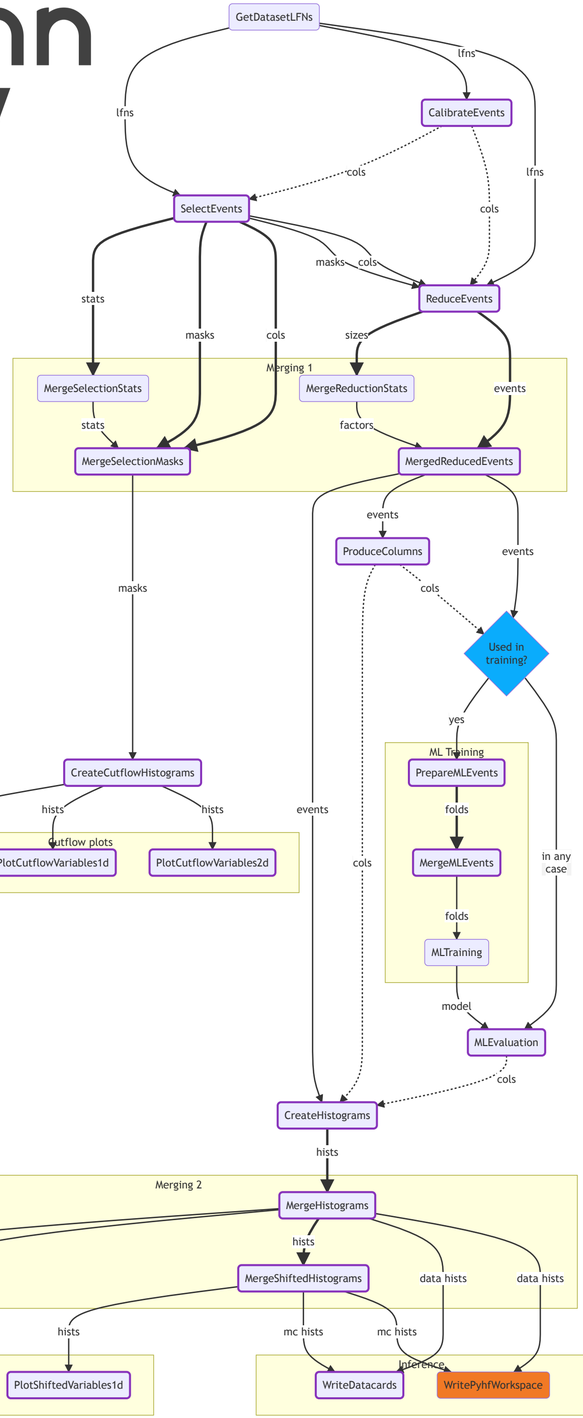
    → difference to map-reduce pattern in coffea processors
- Heavy use of bare NumPy & TensorFlow & awkward, plus coffea object behavior
- Full resolution of systematic uncertainties (next slide)
- Checks every point of the CMS analysis wishlist in the ATTF report

columnflow
(using law & order)



workflow *suggested* by columnflow,
but can be fully customized

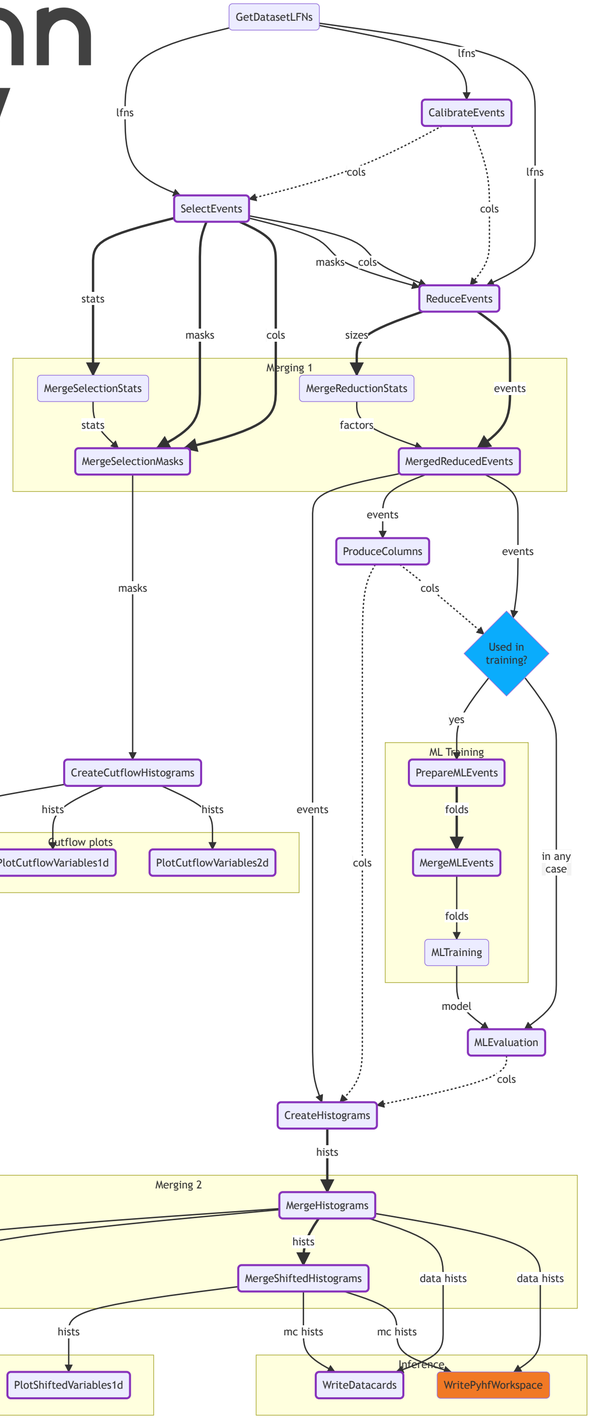
column flow



Initial tasks

Final results

column flow



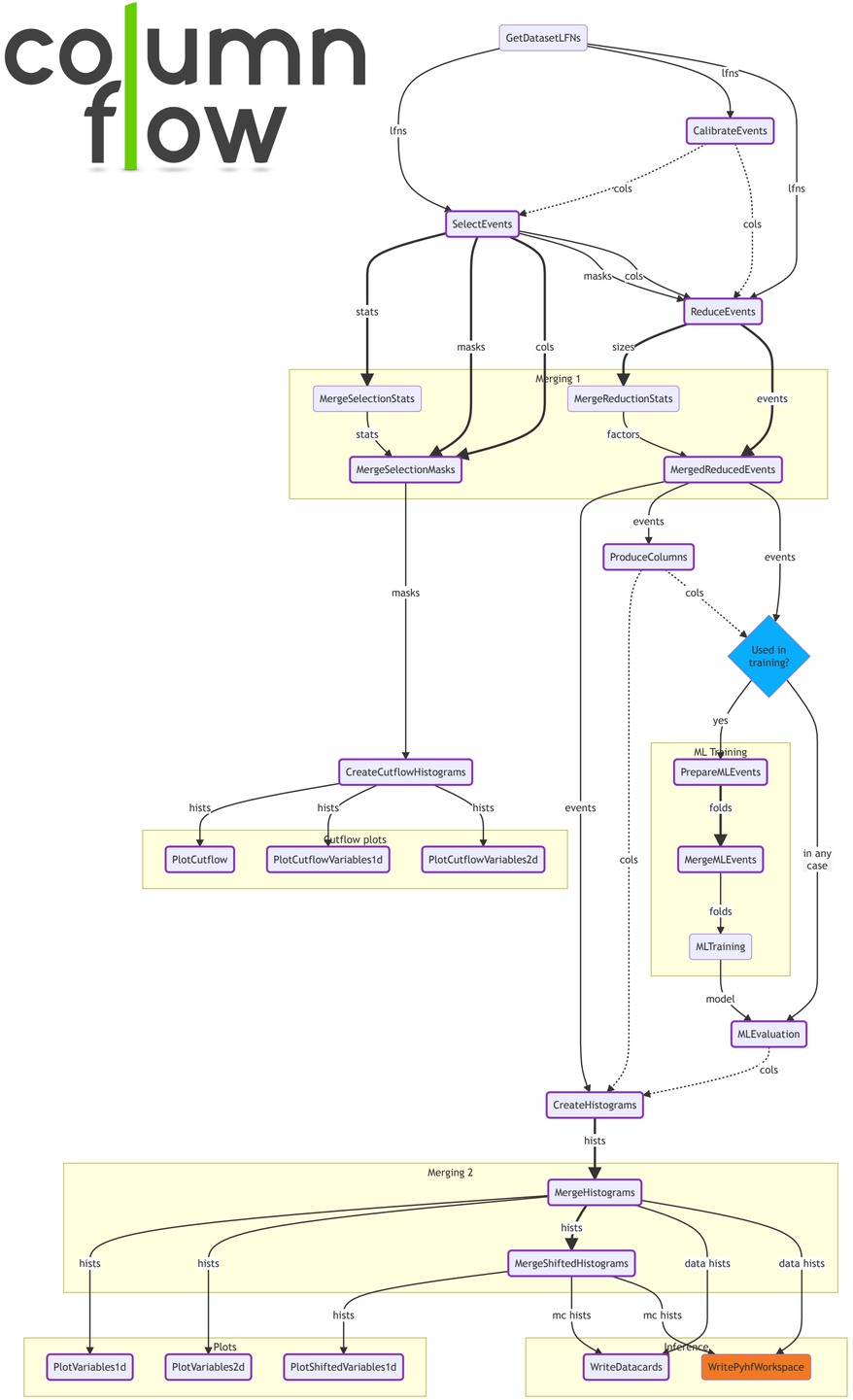
Initial tasks

Final results

"nominal"

Example: Resolution of systematic uncertainties

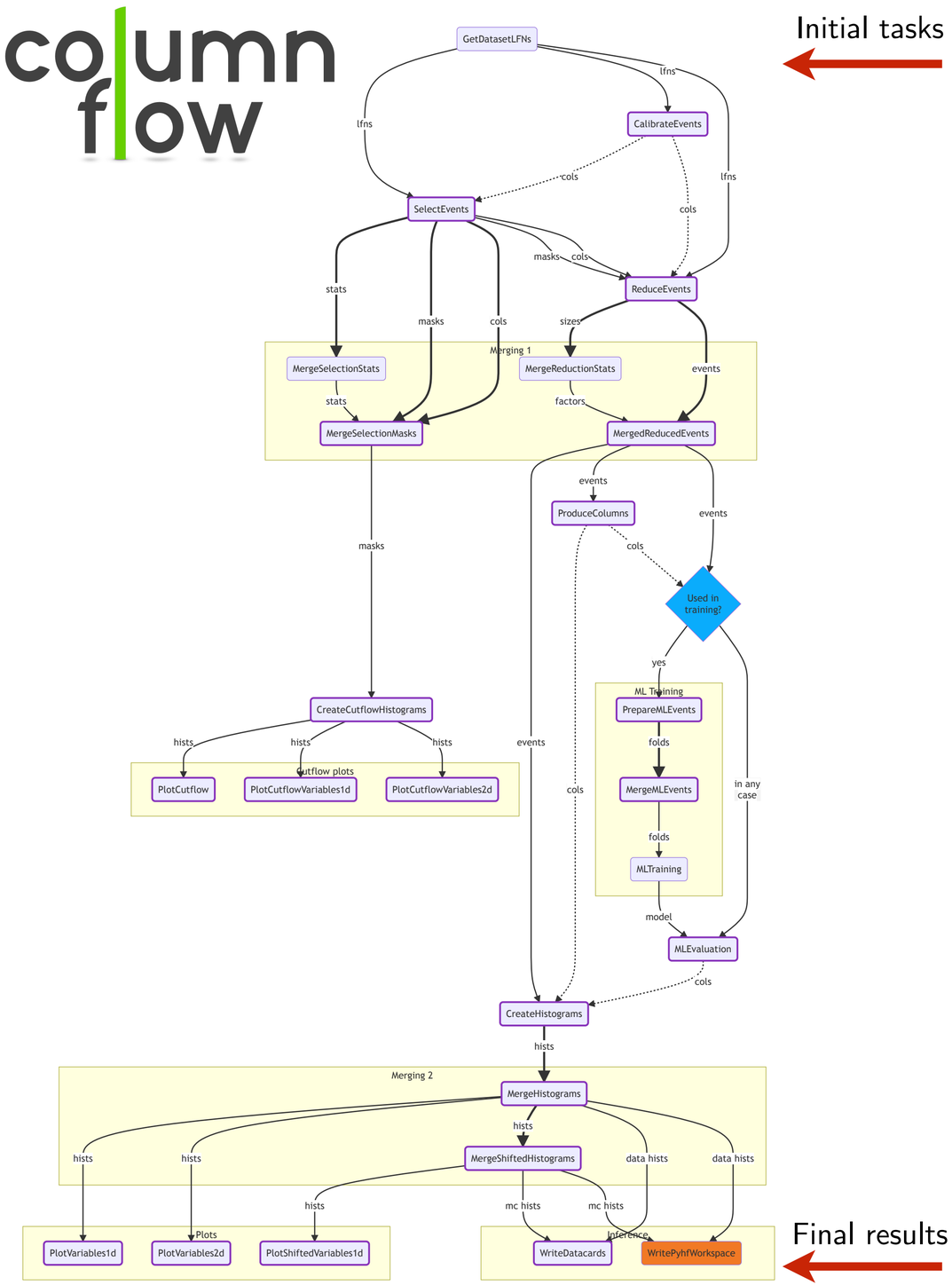
column flow



Initial tasks

Final results

"nominal" "tune(up|down)" "jec(up|down)" "pileup(up|down)" ...

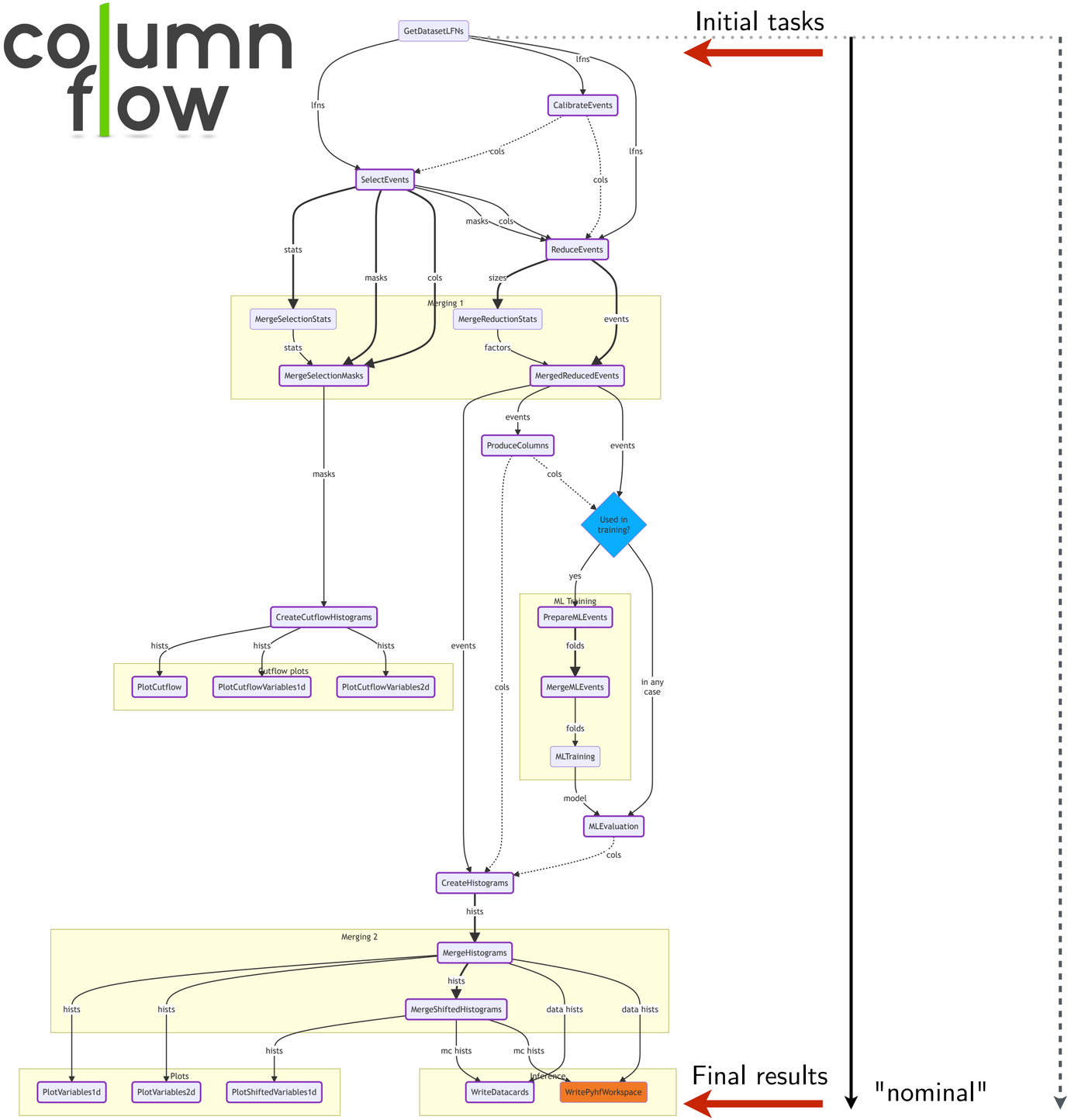


Key idea

Tasks *know* which uncertainties

- ▻ *they implement*
- ▻ they *depend on*
  (through upstream tasks)

column flow



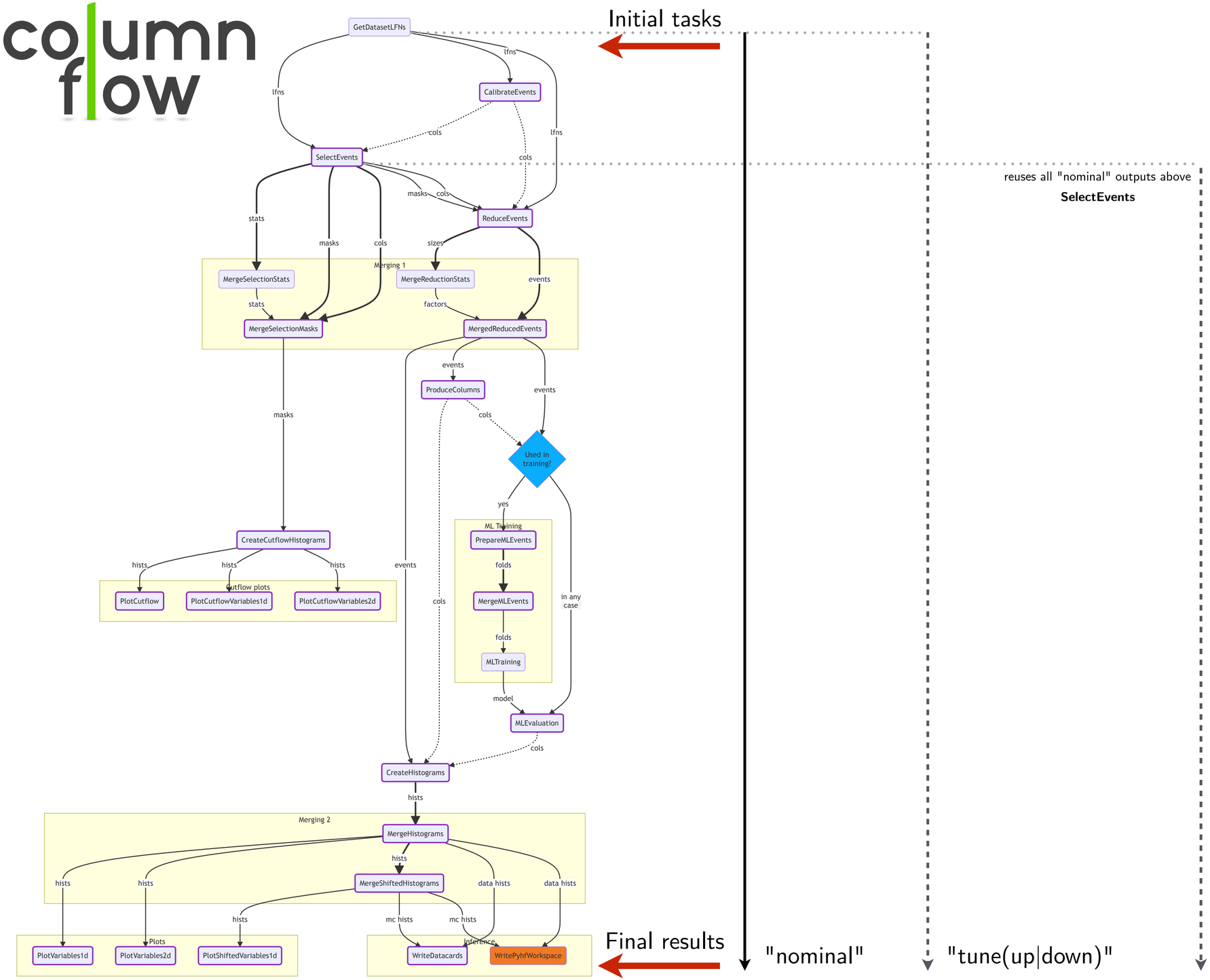
Key idea

Tasks *know* which uncertainties

- ▻ *they implement*
- ▻ they *depend on*
  (through upstream tasks)

Example: Resolution of systematic uncertainties

column flow

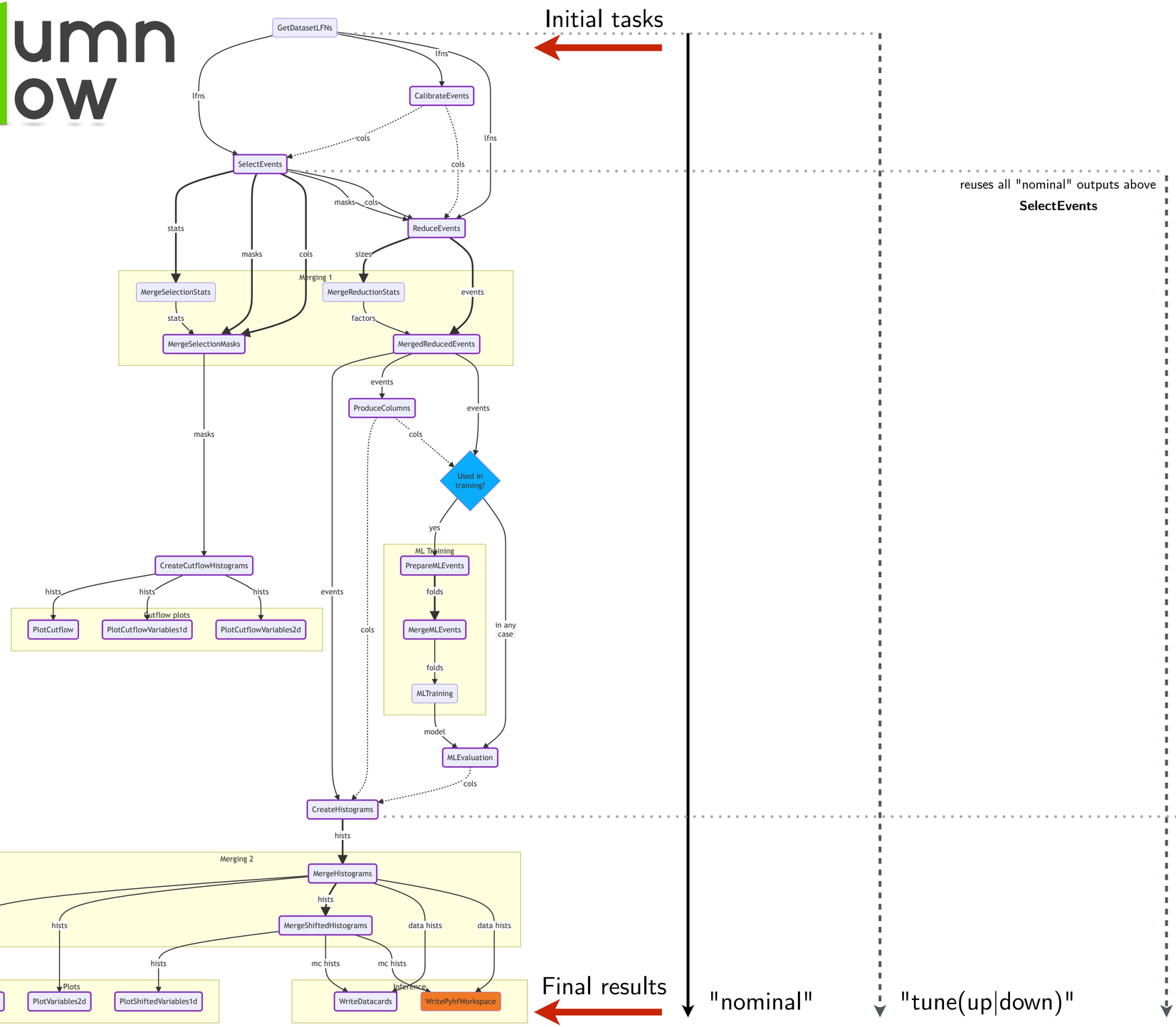


Key idea

Tasks *know* which uncertainties

- ▻ *they implement*
- ▻ they *depend on*
  (through upstream tasks)

column flow



Key idea

Tasks *know* which uncertainties

- ▻ *they implement*
- ▻ they *depend on*
  (through upstream tasks)

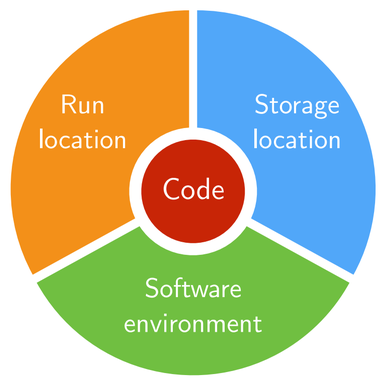
- Resource-agnostic workflow management **essential** for large & complex analyses
  → Need for a flexible **analysis design pattern**

- 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 build frameworks that check every point in the CMS analysis wishlist

law
luigi analysis workflow

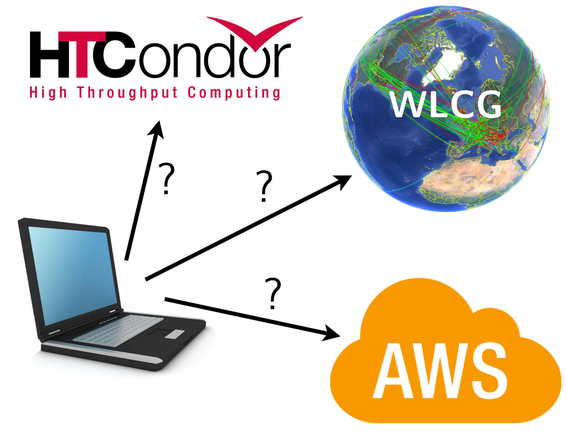
github.com/riga/law
law.readthedocs.io



Collaboration & contributions welcome!

Backup

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

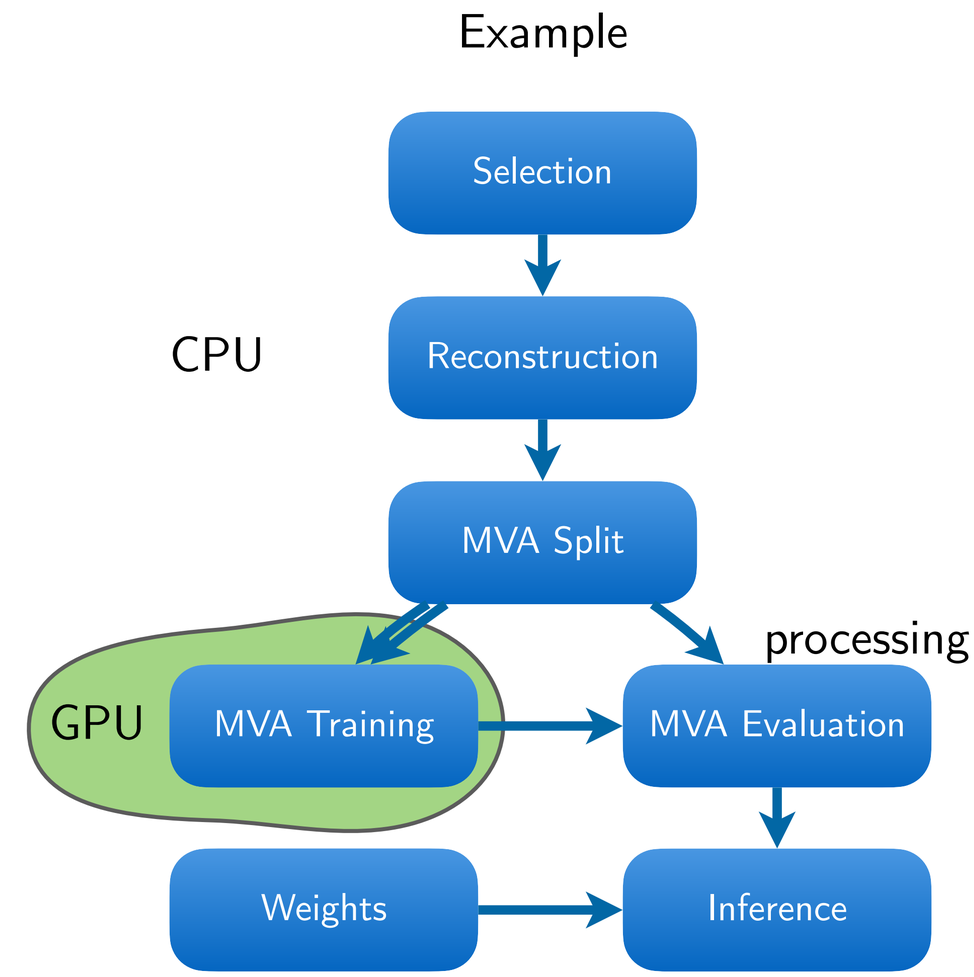


- **Familiar situations**
  - "We couldn't produce updates, our local cluster is down for maintenance."
  - "We need to run things again, we forgot to change some paths in script XYZ."
  - "No updates from my side, I had to do job sitting the whole week ..."

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

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

- 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
- Reproducible intermediate and



→ Reads like a checklist for analysis workflow management

Tailored systems

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

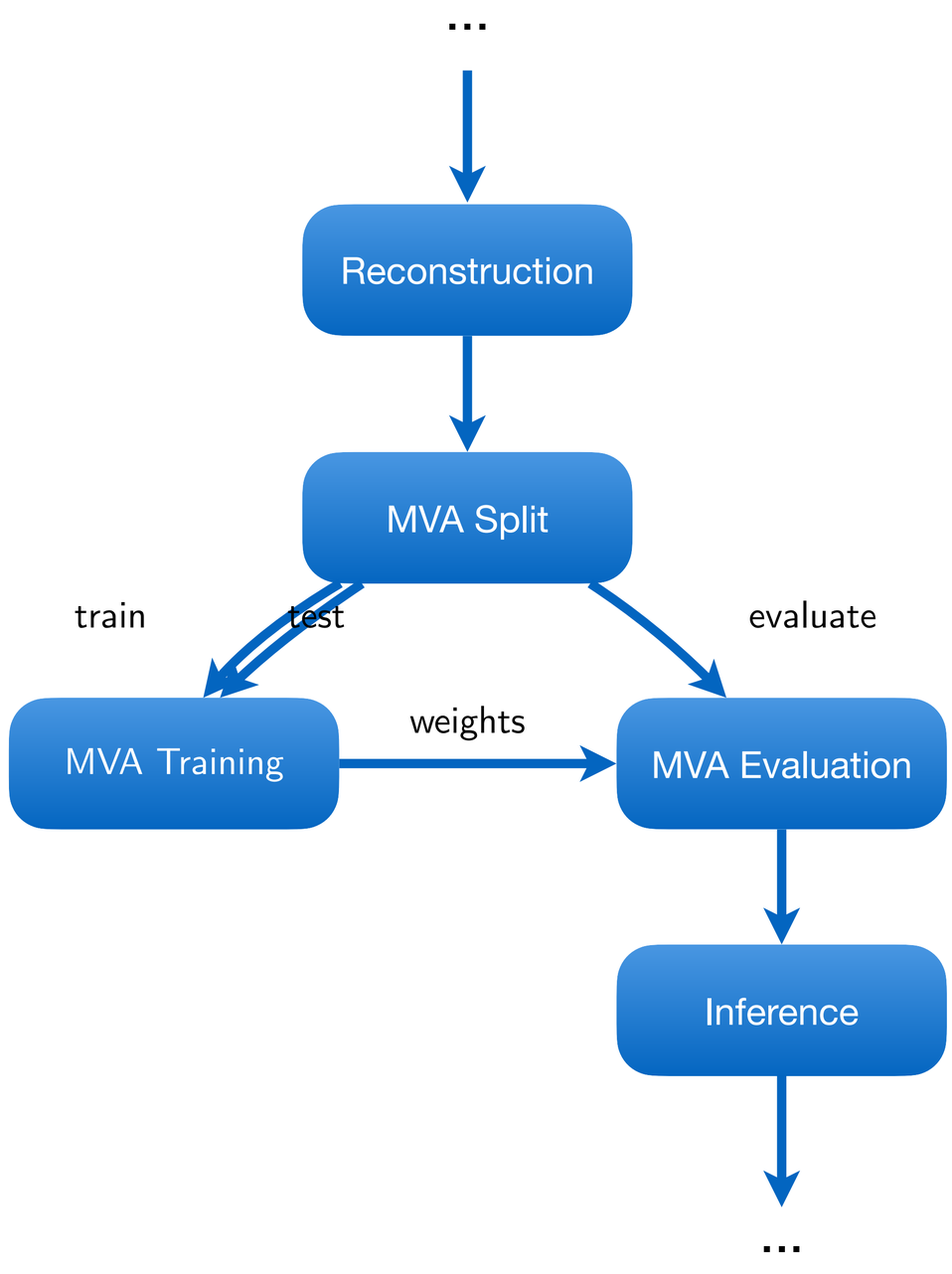
Wishlist for end-user analyses

- 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

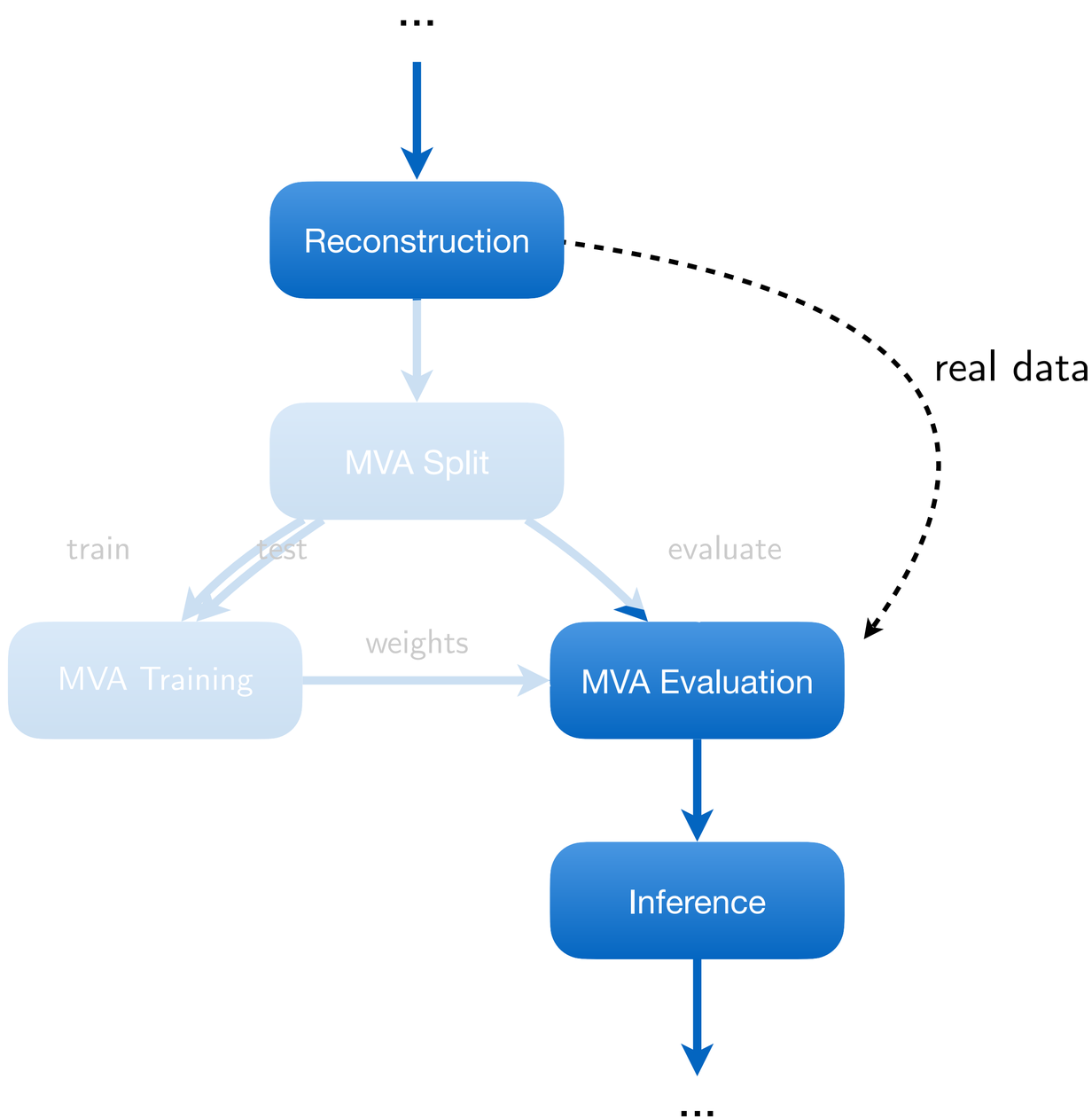
A typical example: ML workflow with uncertainties

Nominal MC



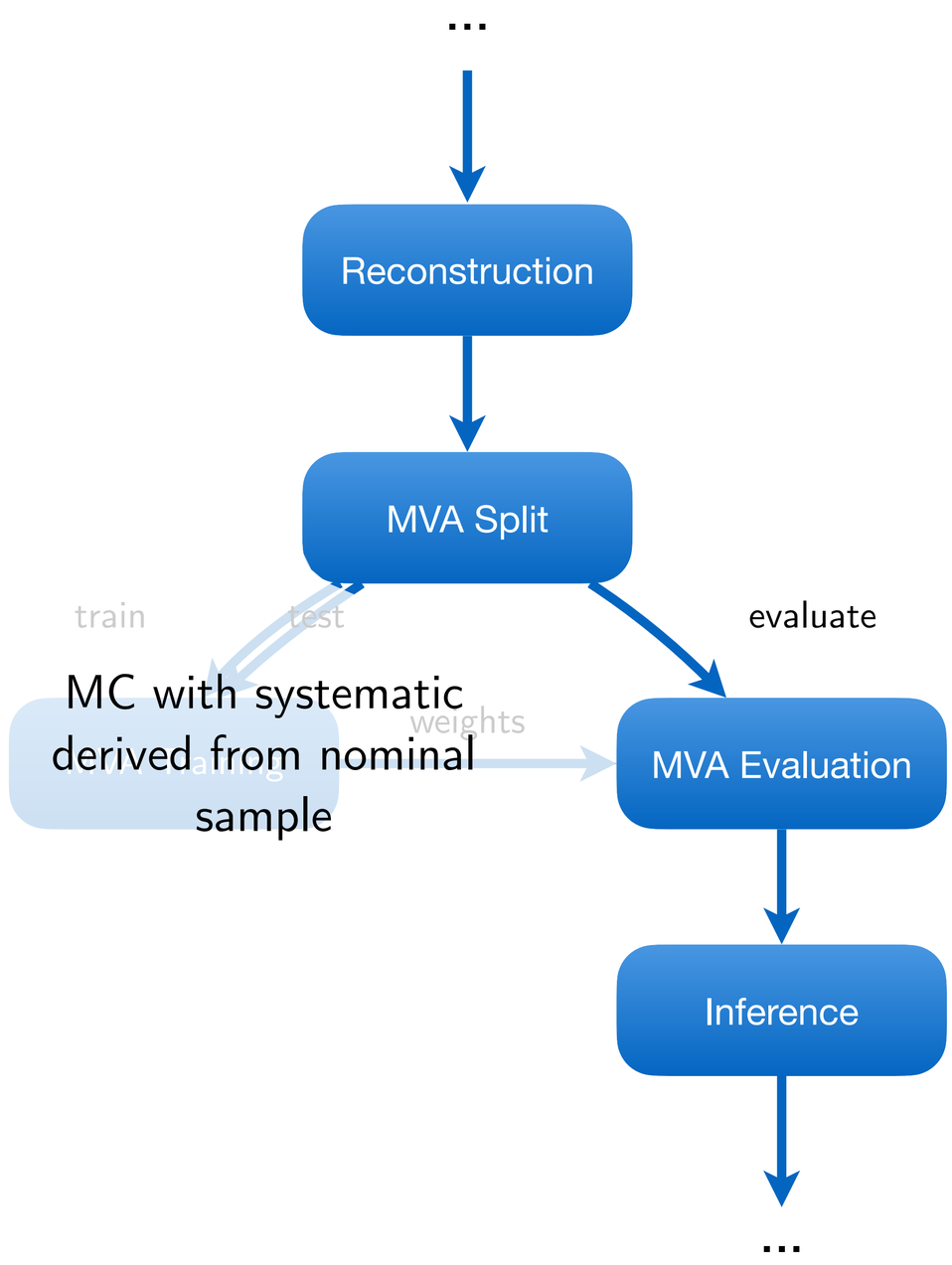
A typical example: ML workflow with uncertainties

Data

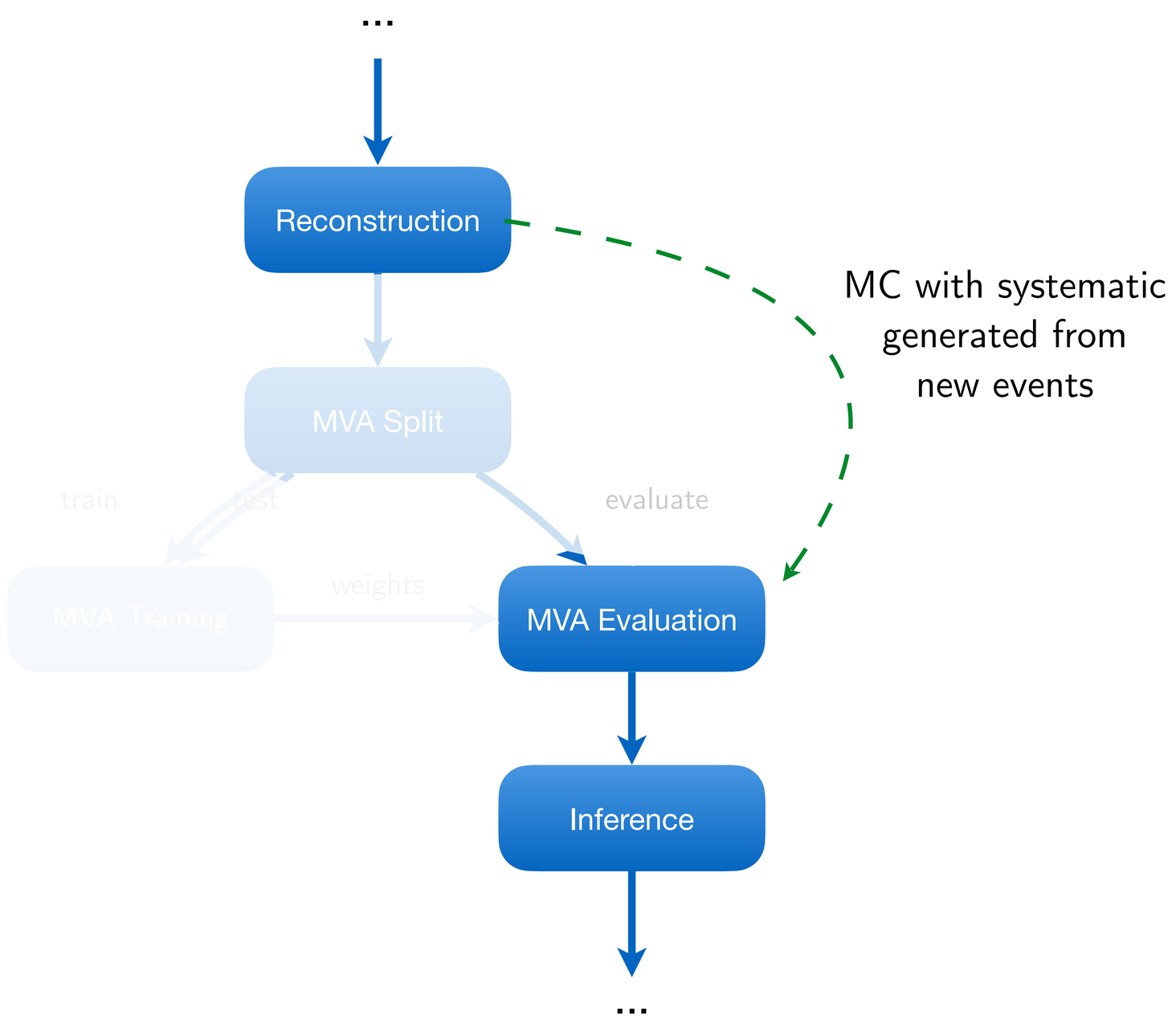


A typical example: ML workflow with uncertainties

MC, Syst. I



MC, Syst. II


...
Reconstruction
MC with systematic generated from new events
MVA Split
train
test
evaluate
MVA Training
weights
MVA Evaluation
Inference
...

- **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 easily work with files after O(months / years)

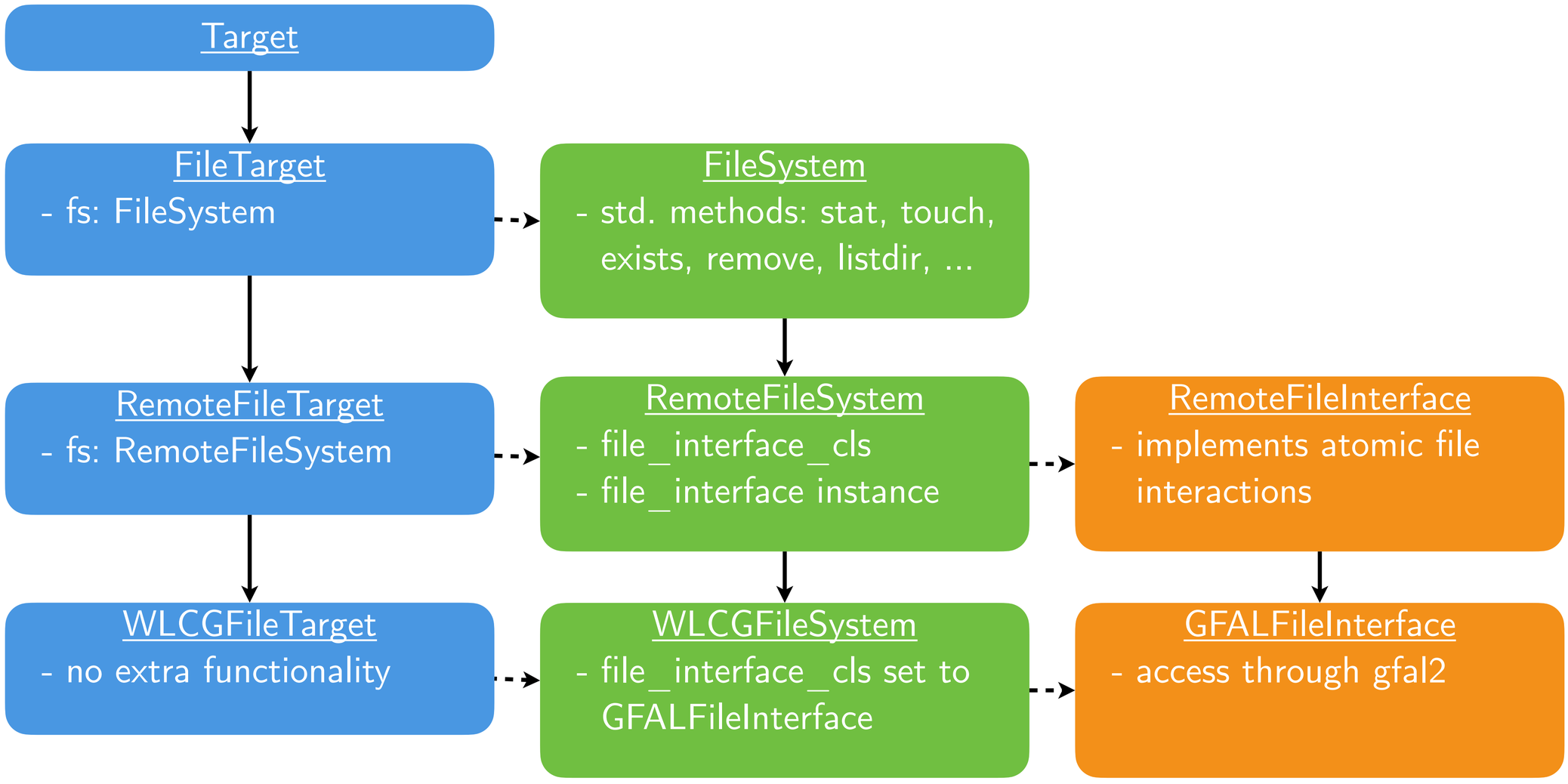
- **No fixation of certain resources**
  → "We have to run at XYZ because the ntuples are there" (can't always be avoided though)
  → Should at least be possible, even if not 100% efficient

- **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

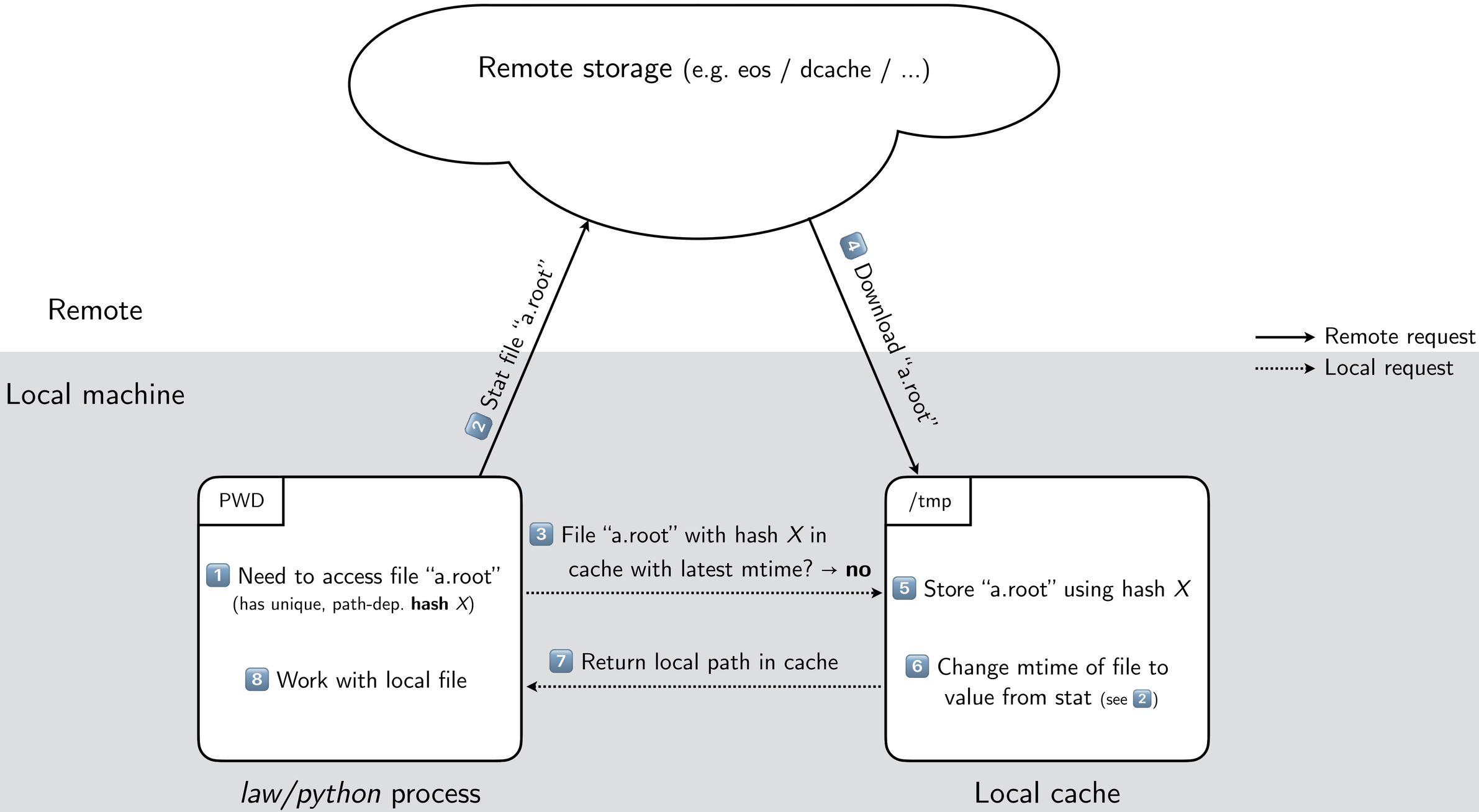
(Remote) targets

```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)
```

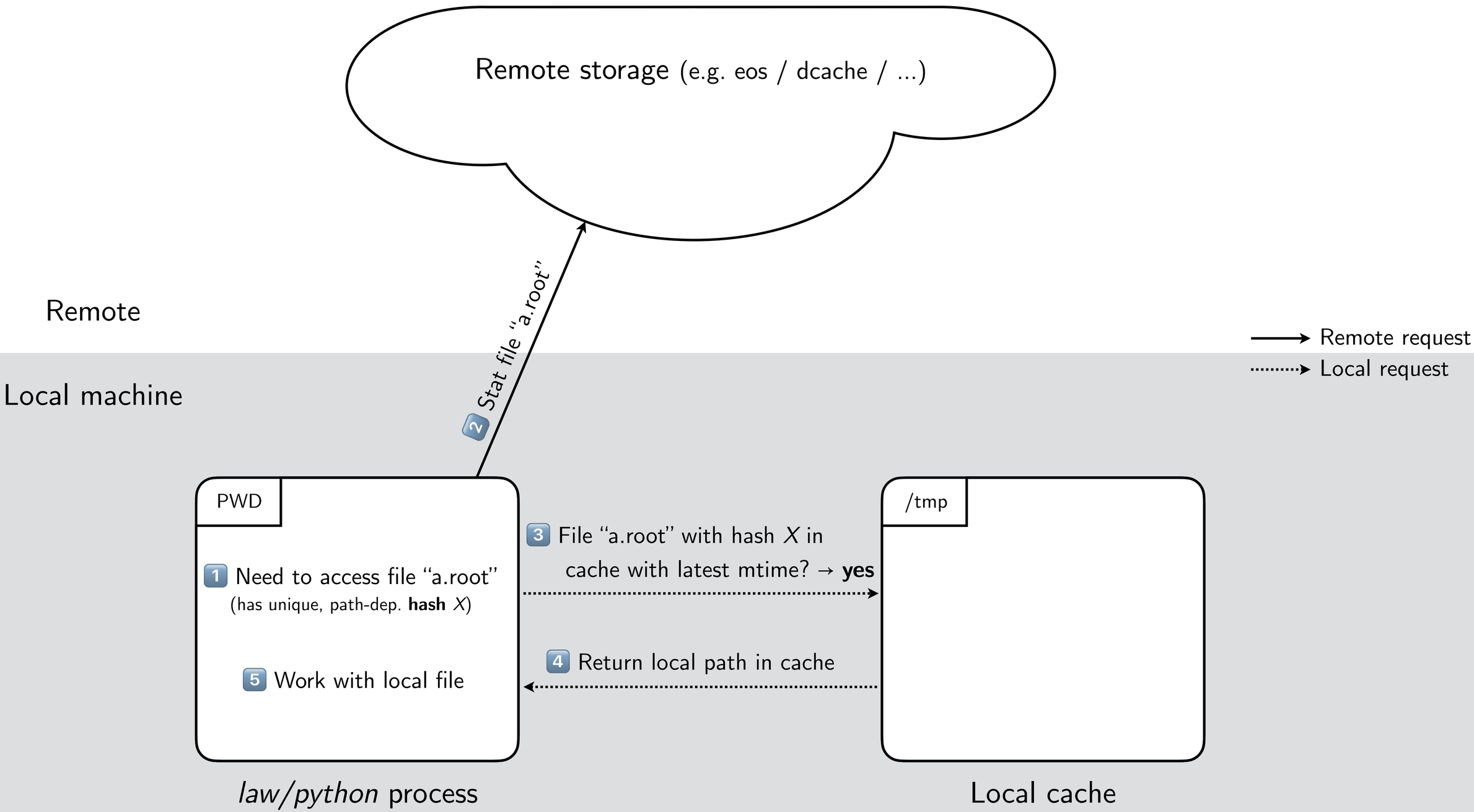
Remote target implementation details



Configuration ☞



Configuration ☞



Workflows

- **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

Common

Workflow specific

Implemented by task

```python
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()`

- 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")
```

- **5 remote workflow implementations come with law**
  - htcondor, glite, lsf, arc, slurm (cms-crab in development)
  - 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

Status polling of remote jobs

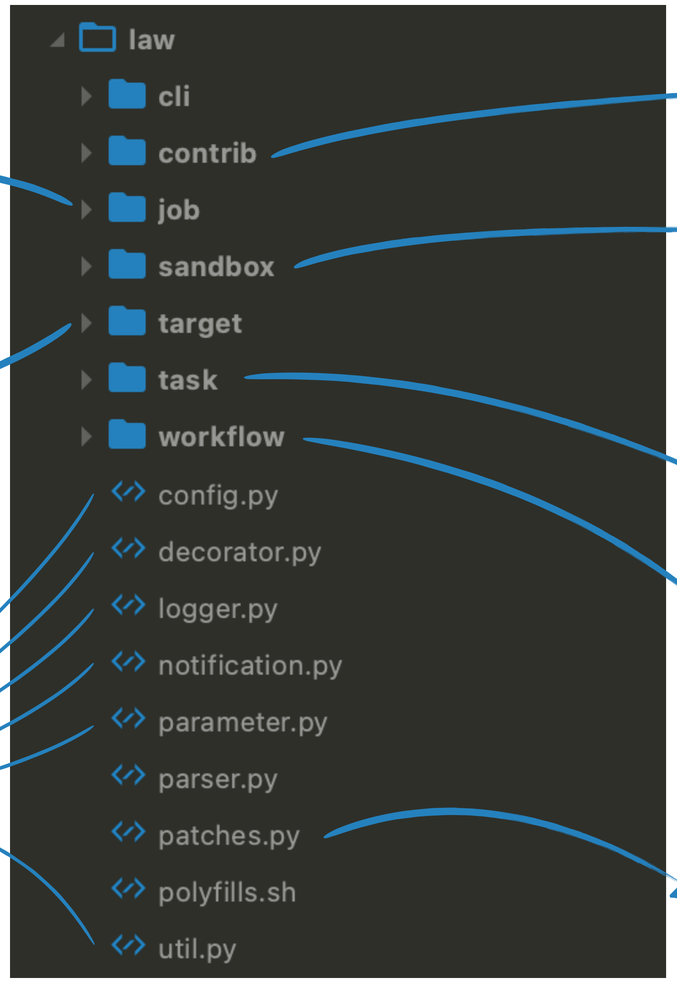

```
16:04:23: all: 3321, pending: 2821 (+2821), running: 426 (+426), finished: 74 (+74), retry: 0 (+0), failed: 0 (+0)
16:04:37: all: 3321, pending: 2829 (+2829), running: 5 (+5), finished: 487 (+487), retry: 0 (+0), failed: 0 (+0)
16:06:15: all: 3321, pending: 2827 (-2), running: 6 (+1), finished: 488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

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



3rd party tools

Sandboxing mechanism
- Sandbox task
- Sandbox interface
- Bash sandbox impl.

Base task definitions

Base workflow definition
- Local workflow impl.
- Remote workflow interface

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

Lightweight patches of luigi, e.g.:
- Disable dep. checks in sandboxes
- Colorize logs

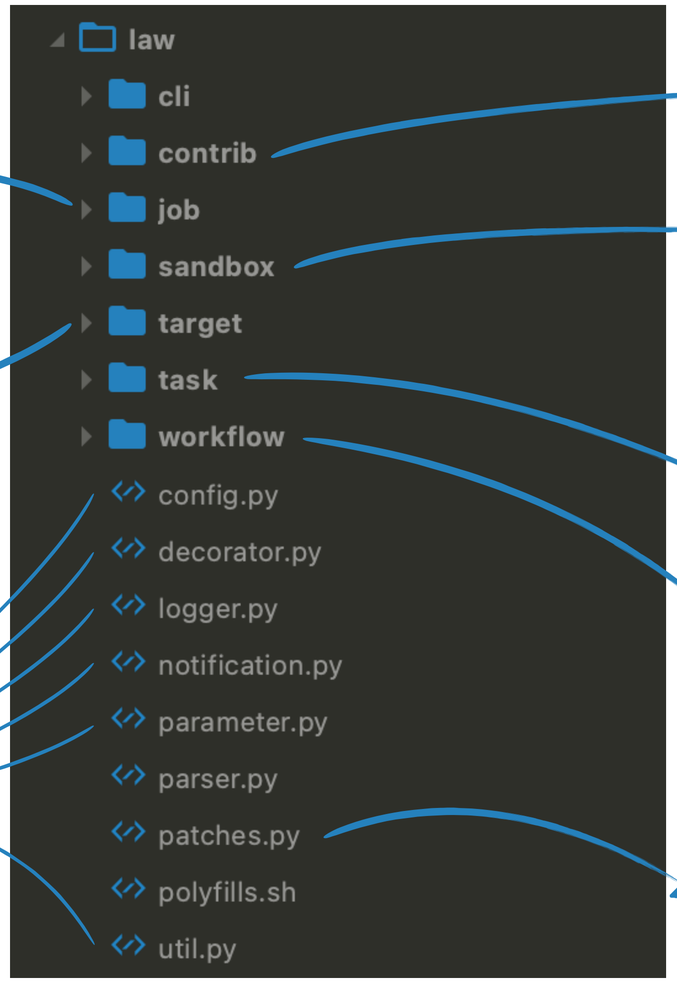
→ Could be added directly to luigi

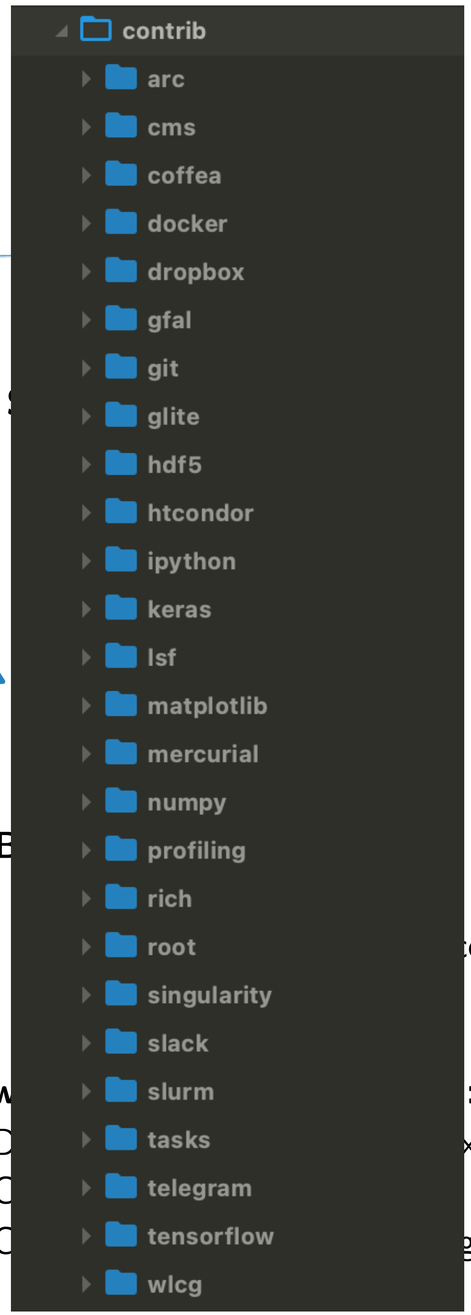
Command-line interface

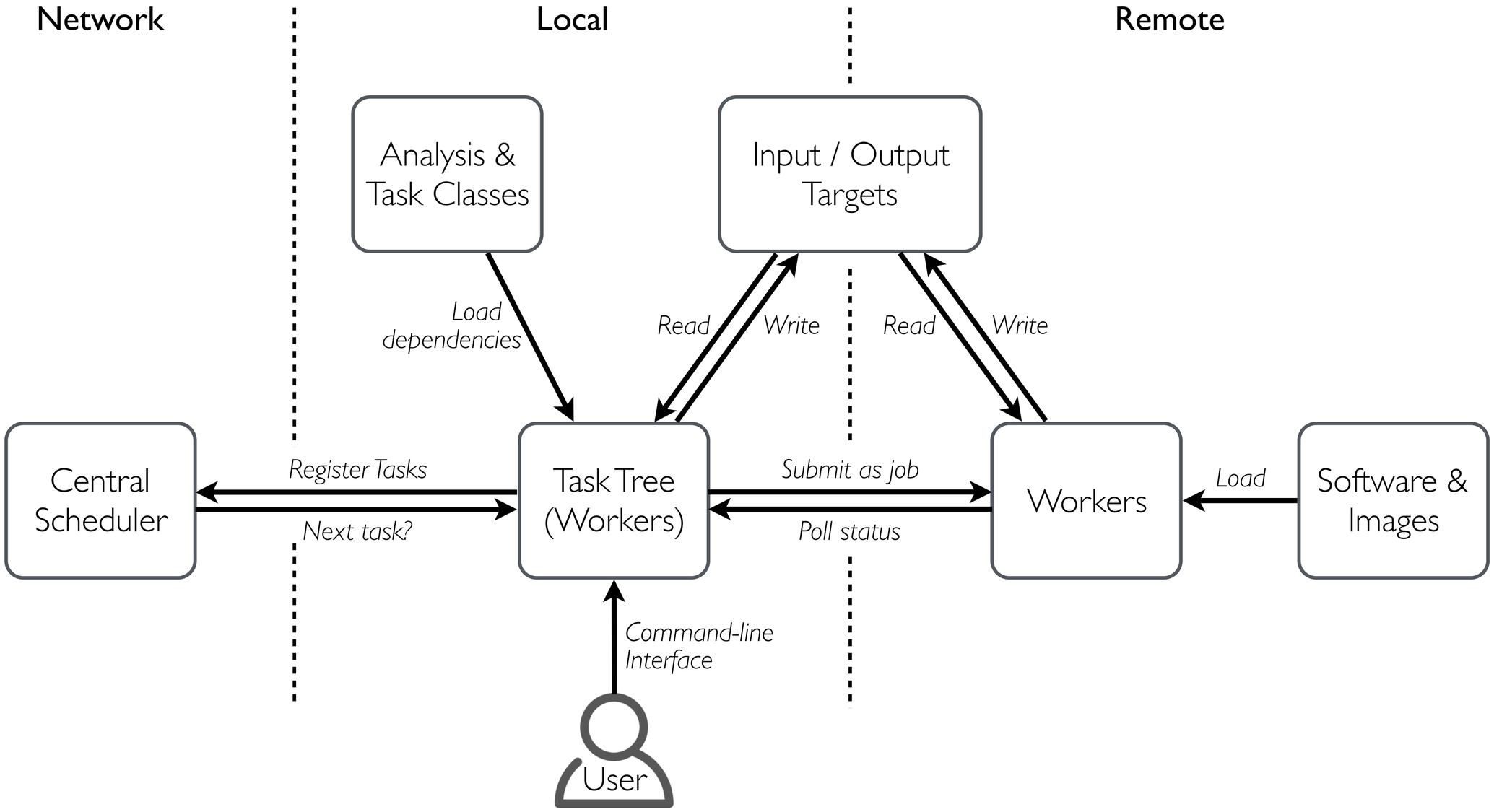
Job interface
- Job file factory interface
- Job manager interface
- Generic remote job script

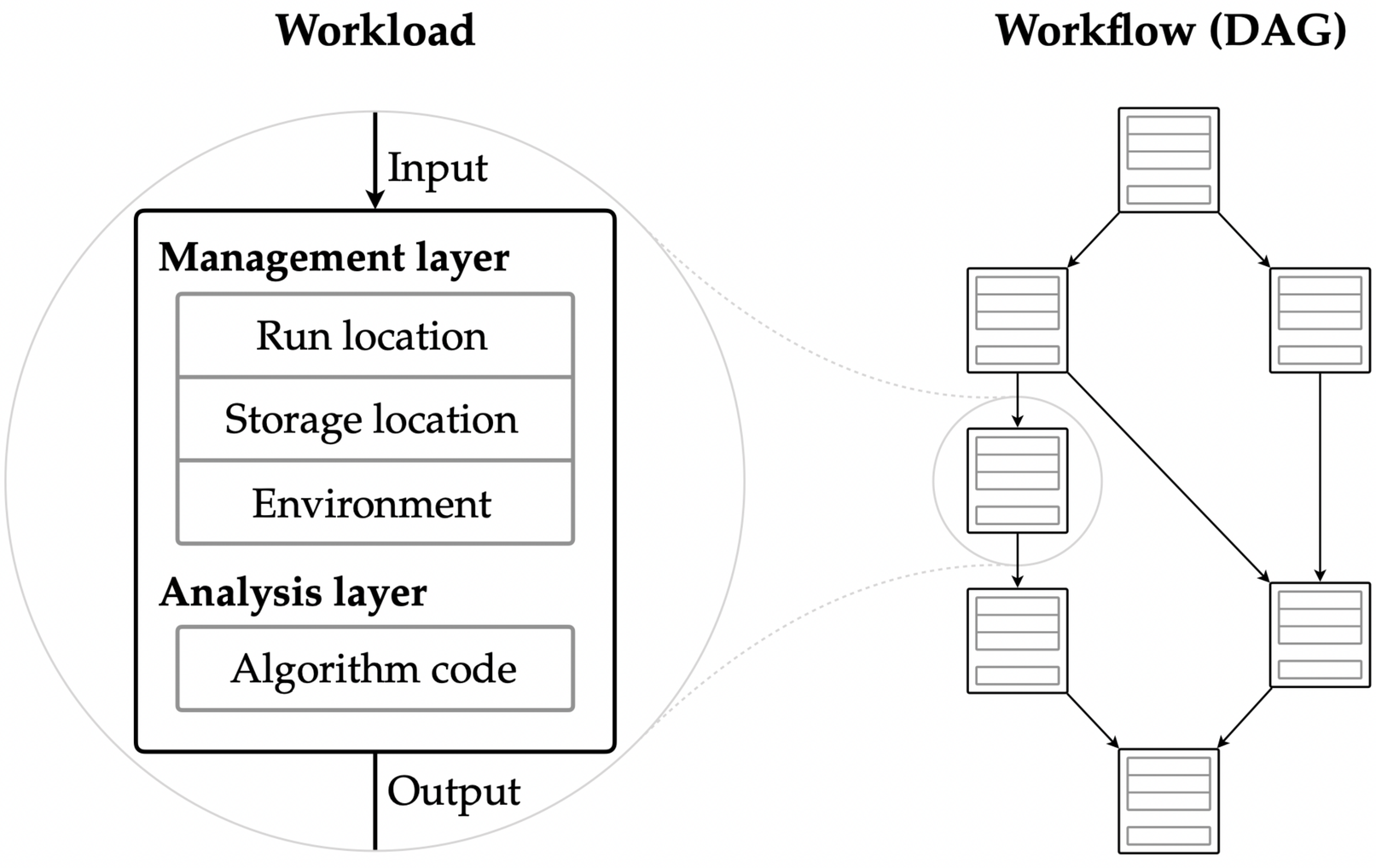
Target definitions
- Generic + file interace
- Local target impl.
- Remote target interfaces

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






Network
Local
Remote
Analysis & Task Classes
Input / Output Targets
Load dependencies
Read
Write
Read
Write
Central Scheduler
Register Tasks
Next task?
Task Tree (Workers)
Submit as job
Poll status
Workers
Load
Software & Images
Command-line Interface
User


Workload
Input
Management layer
Run location
Storage location
Environment
Analysis layer
Algorithm code
Output
Workflow (DAG)

- *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