Orchestrated columnar analysis with columnflow

Mathis Frahm
on behalf of the
CMS collaboration

General idea

- Python-based framework flat ntuples as inputs
- End-to-end **orchestration** & **automation**
- **No reliance** on single local cluster or local storage
- Adapt to any remote cluster and storage system
  - ▻ HTCondor, Slurm, CMS-CRAB, LSF
  - ▻ Store via `file://`, `xrootd://`, `gsiftp://`, `webdav://`
- **Persistent intermediate outputs**
  - ▻ Debugging, reuse, sharing across groups

Key concepts

- **Experiment-agnostic** core
  - ▻ Organize experiment-specific recipes in extensions
- Use Awkward Array as interface, Parquet as file format
  - ▻ Give **users full control** over processing tools (NumPy, TensorFlow, coffea-nano-format, pandas, ...)
- Define **workflows** with luigi + law, metadata with order
- Control and execution via **CLI**, **scripts** and **notebooks**
- High degree of **code-reuse** and collaboration

Automation stack

Parallelization over ...

- Datasets
- Files
- Systematic uncertainties
- Data-taking periods

▻ Running **complete workflow** on standard resources in $\mathcal{O}$**(hours)**

▻ HTCondor, CRAB, ...

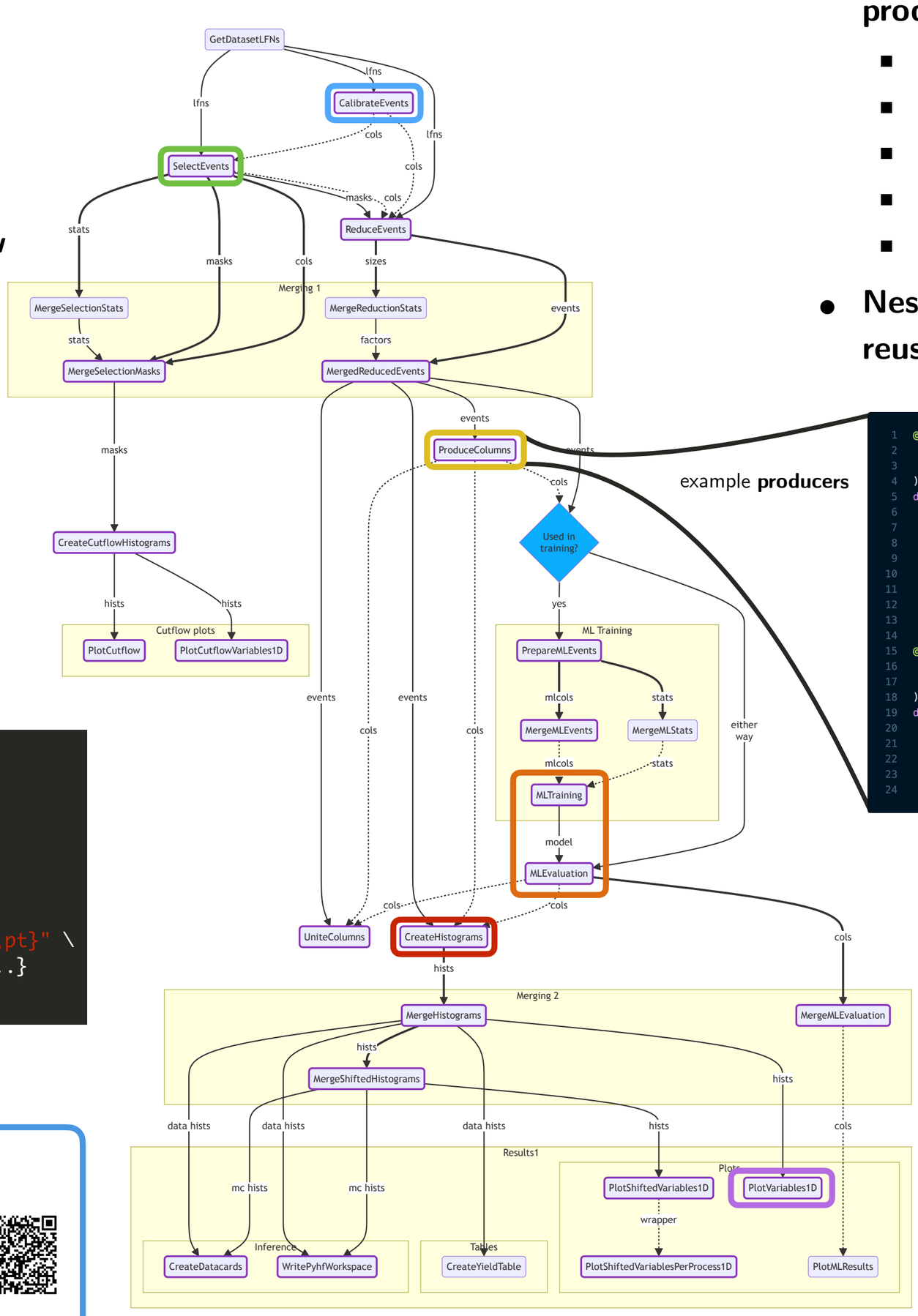
Graph execution

- **Single command** can trigger the full pipeline from **inputs to plots**
- **Example**

```
> law run cf.PlotVariables1D \
    --version dev1 \
    --datasets ttbar,dy \
    --calibrators jec,jer \
    --selector full \
    --producers features \
    --variables "m_jj,jet*_{eta,pt}" \
    --workflow {crab,htcondor,...}
```

Orchestrated workflow*

(Can be altered or amended by analyses)*



Simple customization

- Provide simple functions, **producers**, to create
  - **calibrated** (*updated*) **columns**
  - **selection masks**
  - **new columns**
  - **ML training & evaluation**
  - **histograms**
- **Nesting** enables for easy **reuse** and **capsulation**

example **producers**

```python
@producer(
    uses={"Jet.{pt,eta,phi,mass}"},
    produces={"m_jj"},
)
def dijet_mass(self: Producer, events: ak.Array, **kwargs) -> ak.Array:
    # add four vectors
    dijet = events.Jet[:, 0] + events.Jet[:, 1]

    # store the dijet mass in a new column
    events = set_ak_column(events, "m_jj", dijet.mass, value_type=np.float32)

    return events


@producer(
    uses={dijet_mass, event_weights},
    produces={dijet_mass, event_weights},
)
def features(self: Producer, events: ak.Array, **kwargs) -> ak.Array:
    # combine multiple producers via nesting
    events = self[dijet_mass](events, **kwargs)
    if self.dataset_inst.is_mc:
        events = self[event_weights](events, **kwargs)
    return events
```

- Using bare **awkward arrays**
- Implementation and **choice of tools** fully **up to user**

Documentation

github.com/columnflow
columnflow.readthedocs.io