

columnflow

— Refactoring of Task Array Functions —

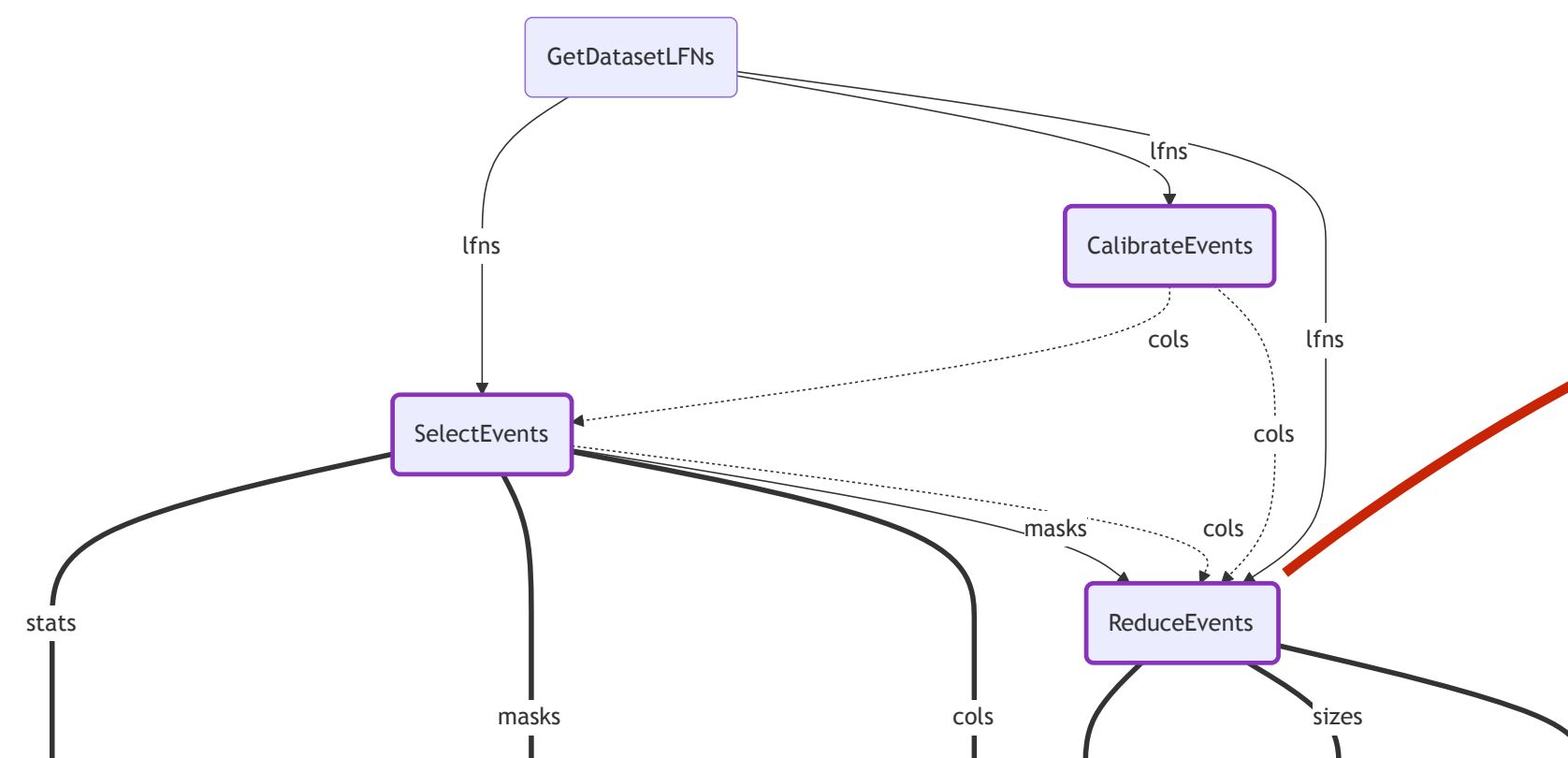
Marcel Rieger

UHH Framework Meeting

28.2.2025

2 Suggestion: reducer

- Our usual "interface" for defining columns to keep: produces set defined by task array functions
- **Only difference:** columns saved by ReduceEvents task
 - Columns are defined in config under keep_columns field



```

1  cfg.x.keep_columns = DotDict.wrap({
2    "cf.ReduceEvents": {
3      # mandatory
4      ColumnCollection.MANDATORY_COFFEA,
5      # object info
6      "Jet.{pt,eta,phi,mass,hadronFlavour,puId,hbtag,btagPNet*,btagDeep*}",
7      "Electron.*",
8      "Muon.*",
9      "Tau.*",
10     "PV.npvs",
11     # keep all columns added during selection, but skip cutflow feature
12     ColumnCollection.ALL_FROM_SELECTOR,
13   },
14 })

```

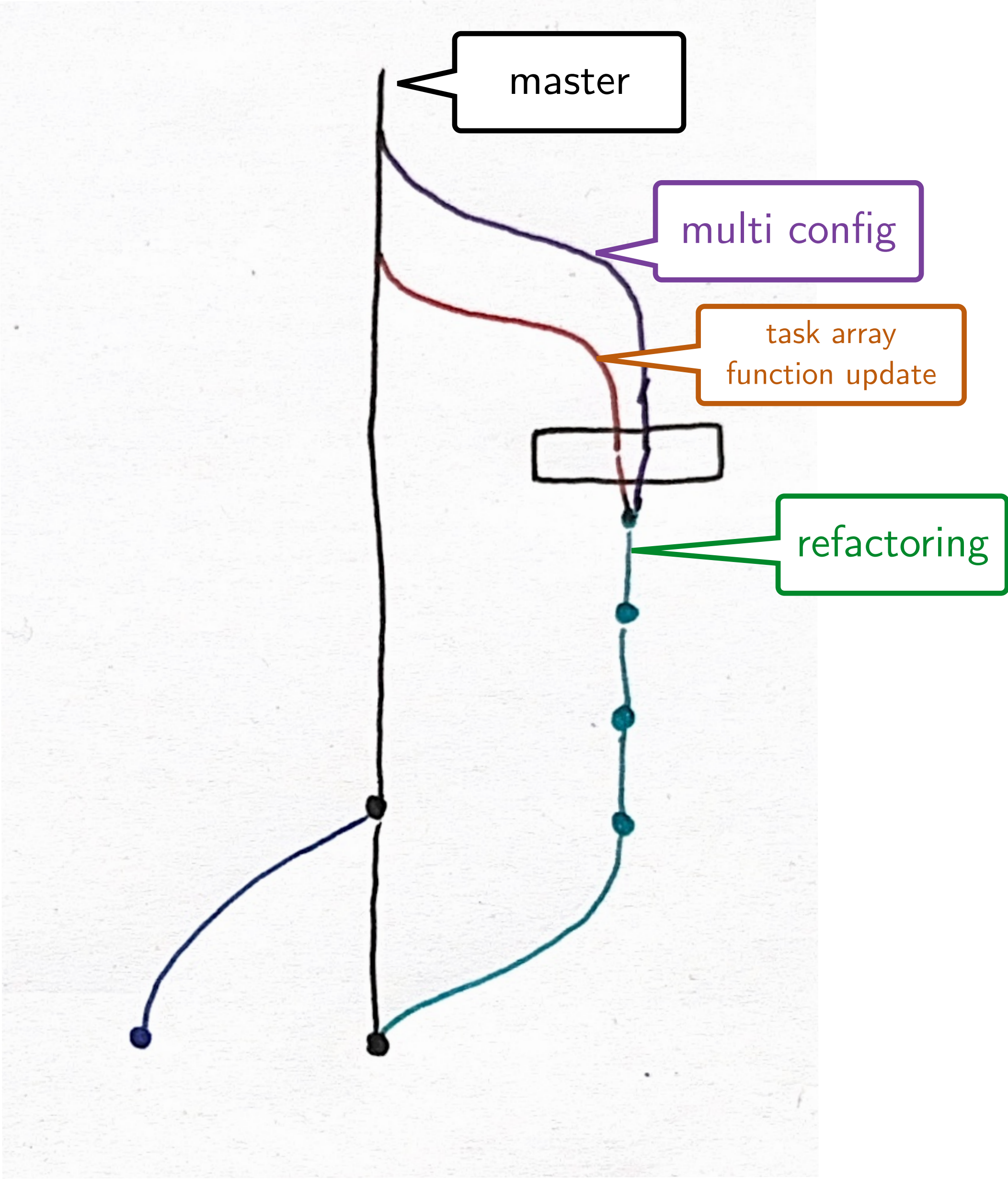
- Lead to misunderstandings in the past, but wasn't too unusual
- The need for **an additional feature** changes this
 - Use case: save only specific GenPart's for use in e.g. ProduceColumns
 - Right now we need to save them in output columns of SelectEvents
 - Waste of resources since only selected events will be stored
- **Idea:** add a new reducer, which is actually just a producer
 - On the cli: `law run cf.PlotVariables1D --selector foo --reducer bar ...`
 - Name be encoded into output paths
 - Add new `@reducer` or just create them via `@producer`? Name clashes?

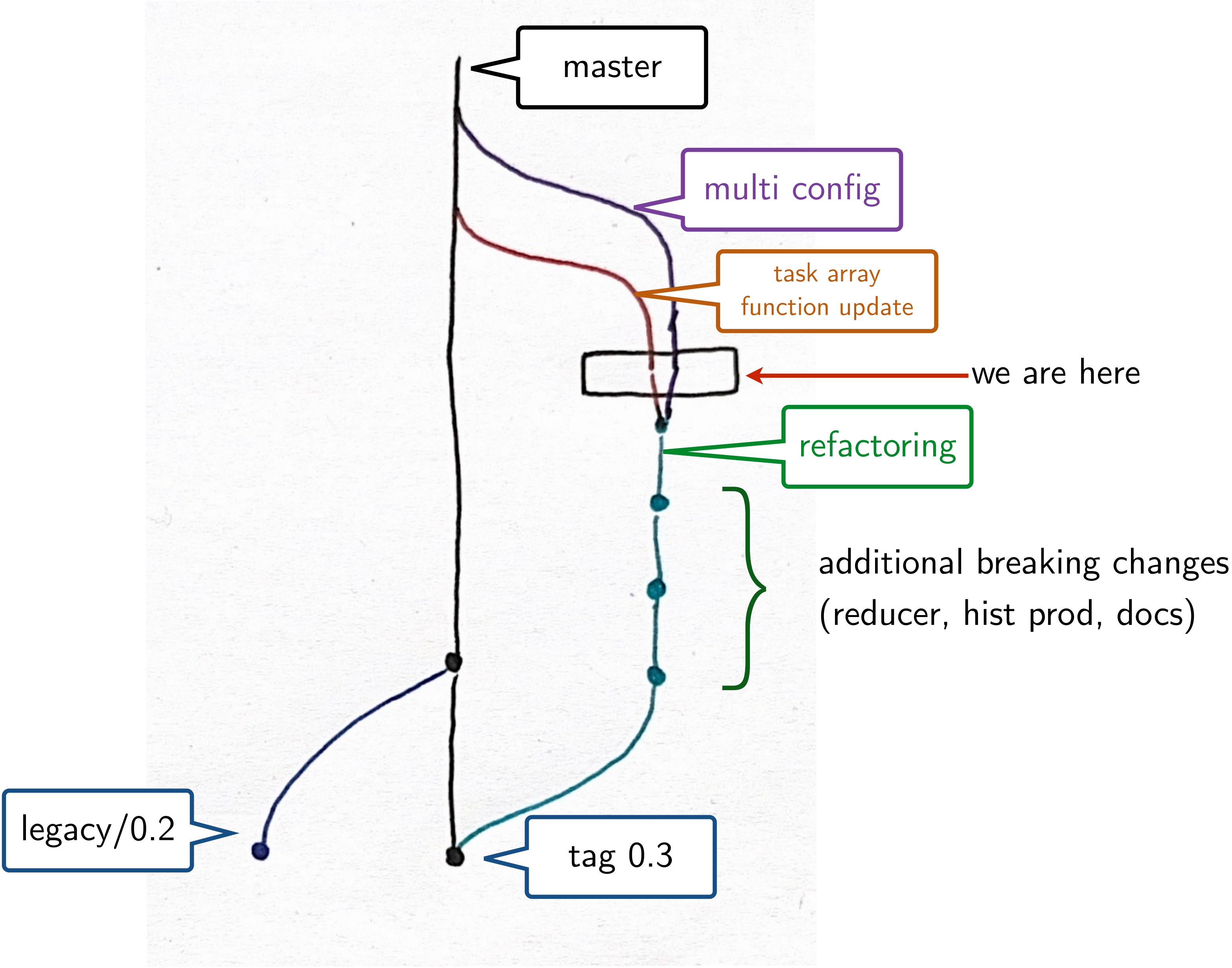
3 Suggestion: hist producer

- We have a `weight_producer` in place that
 - computes the event weight to be applied, and
 - can also impose a late-stage selection

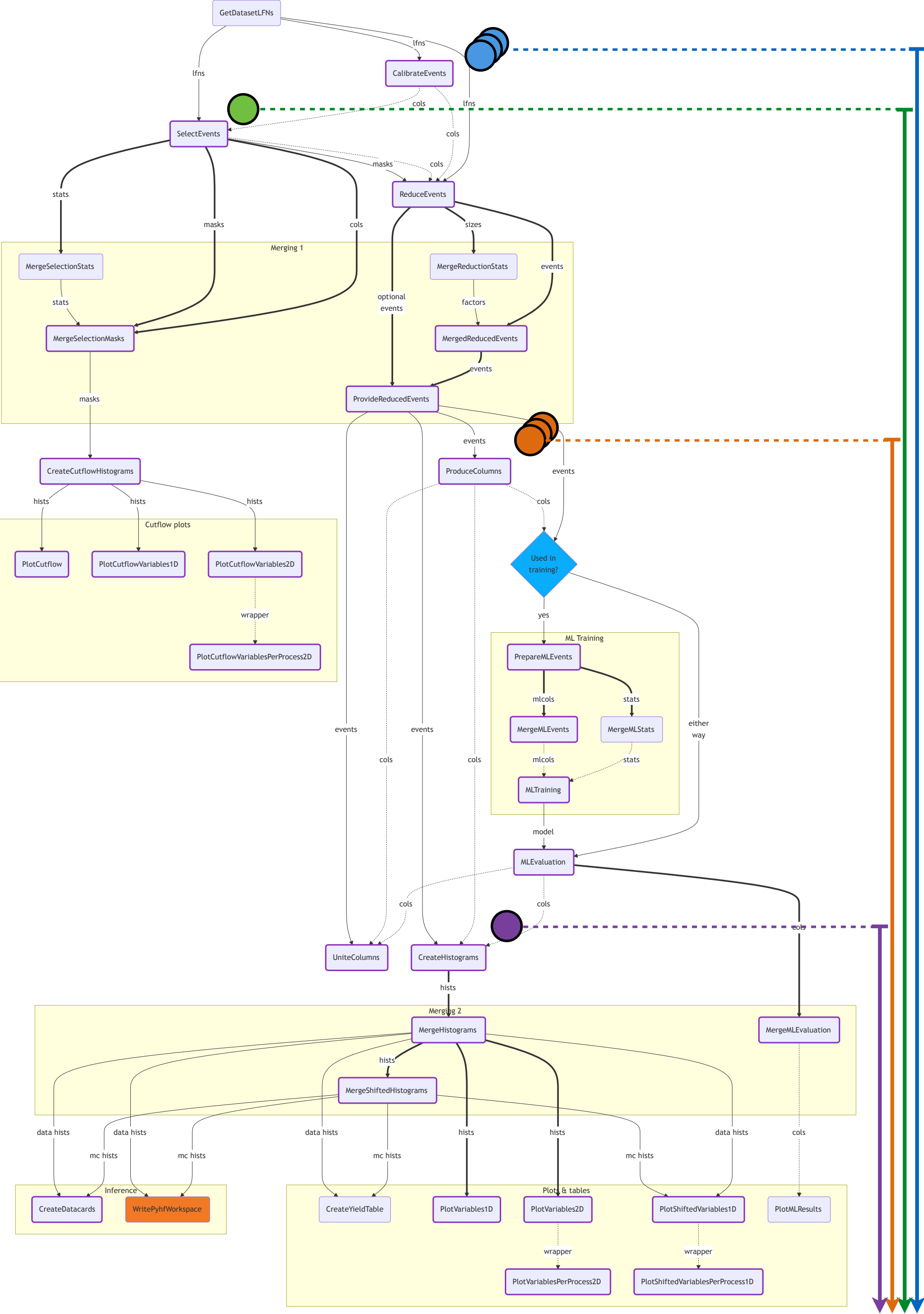


- **However**, multiple analyses could profit from more flexibility for customizing
 - histogram axes
 - different hist storage (int, double, float*) and weight types
 - ! the way that events and weights are filled
 - ▷ many use cases
- Generalizing `weight_producer` to `hist_producer` (?) might come in handy



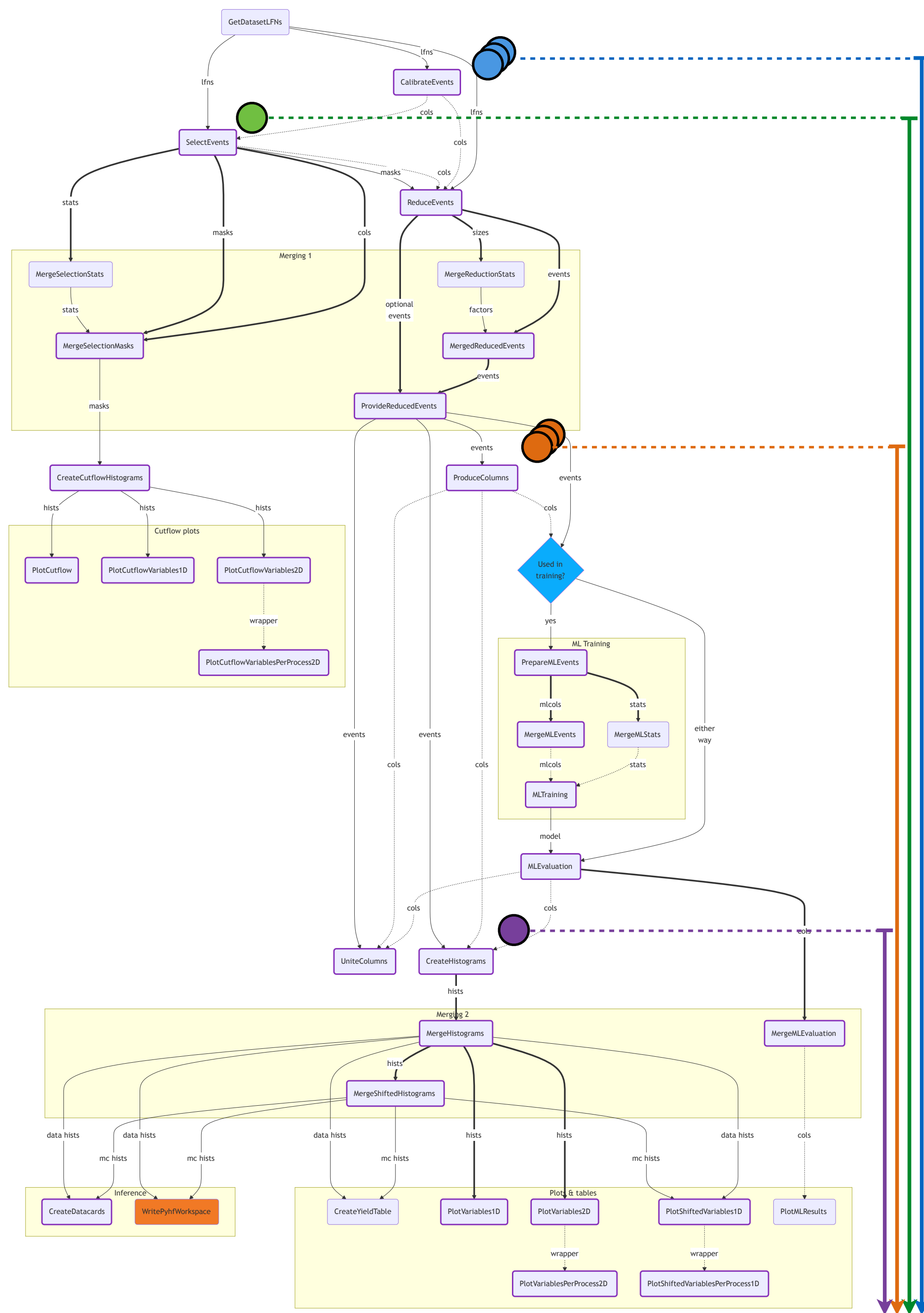


Backup



- Task array functions are **actually invoked** at four locations ●
 - Calibrators
 - Selector
 - Producers
 - Weight producer
- Parameters like --calibrators known by all downstream tasks →

6 Issues with current implementation



- Task array functions are **actually invoked** at four locations ●

- Calibrators
- Selector
- Producers
- Weight producer

- Parameters like `--calibrators` known by all downstream tasks →

- ! Right now, **each** TAF and all dependencies are instantiated

- in **each** downstream task
- by **each** workflow and branch task
- unnecessarily **multiple times**
 - Tens of thousands of redundant calls

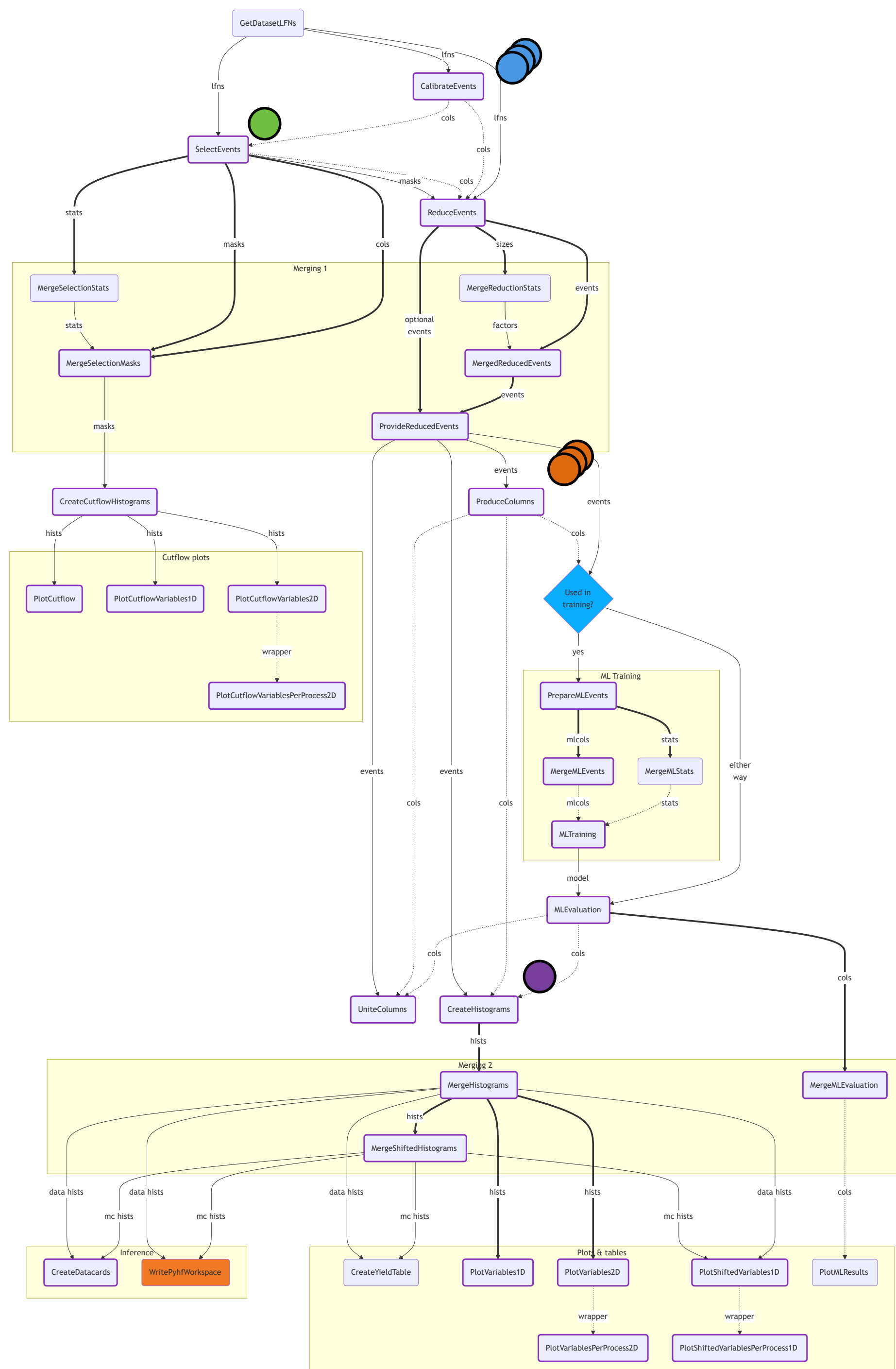
- ! Weird situations emerge

- Unclear when (e.g.) `init` is called and which attributes are available
- "It's called everywhere, so it hopefully works at *some point!*?"

```

1  @producer(...)
2  def all_weights(self, events, ...):
3      ...
4
5  @all_weights.init
6  def all_weights_init(self: WeightProducer) -> None:
7      if not getattr(self, "dataset_inst", None):
8          return
9
10     # code that needs a dataset_inst ...
  
```


7 Paradigms & thoughts on shifts



- Task array functions are bound to **three objects**

- analysis instance
 - config instance
 - dataset instance
- } **"constants"**

➤ They will **never change** throughout its lifetime

- Depending on these, TAFs can have dynamic behavior

- Used / produced columns
- Other TAFs they depend on

! Which **shifts** they yield to the analysis

➤ Example: once JEC is invoked, your analysis *can* depend on JEC shifts

➤ Highly important for tasks to understand where

! Decisions can depend on the **three "constants"** above and even runtime conditions

➤ Example: era-dependent number of DY weight uncertainties

- My main take-away*

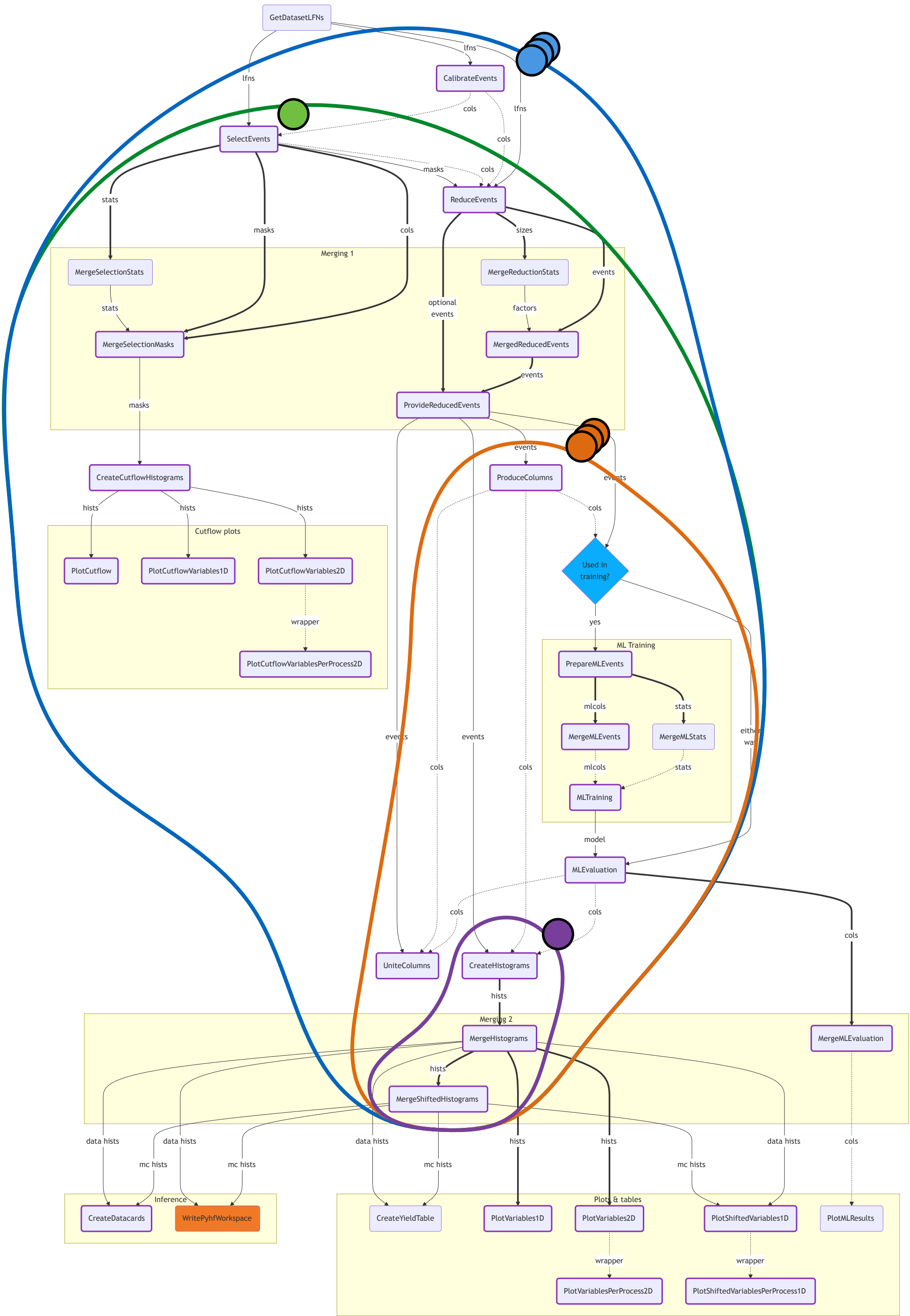
- Shifts cannot be part of these **"constants"**
- Currently they are and this causes a lot of headaches
- Fixing this could lead to a heavily improved TAF handling



8 Refactored task array function interface

- Upon creation, {analysis,config,dataset}_inst are passed to the TAF as members → they define the *state*
 - Hooks called thereafter in various places
 - @pre_init(self)
 - ▷ **New**, called **before** dependency creation, can be used to control `deps_kwargs`, fixes current duplication issue
 - @init(self)
 - ▷ Controls used / produced columns and other TAFs as dependencies, **as well as shifts**
 - @skip(self)
 - ▷ Called during init, can decide whether TAF should be removed from dependencies
-
- @post_init(self, **task**)
 - ▷ **New**, can control used / produced columns (using task info and **resolved shift**), **but no additional TAF deps**
 - @requires(self, **task**, reqs)
 - ▷ Allows adding extra task requirements
 - @setup(self, **task**, reqs, inputs, reader_targets)
 - ▷ Allows setting up objects needed for actual function calls
 - __call__(self, events, **task**, **kwargs)
 - ▷ Actual events chunk processing
 - @teardown(self, **task**)
 - ▷ **New**, called after processing, **but potentially before chunk merging**, allows reducing memory footprint

before task instantiation

after task instantiation



- Task array functions can be created **once** and passed to upstream tasks within 
- Tasks outside these bubbles don't need access to TAF instances, but just they class
- Shifts in "overarching" tasks like plotting can be simply gathered through upstream tasks
 Still under development, ETA next week
- Objects like `{ML,Inference}Model` can be treated similar, however, implications not as deep