

Tooling for Differentiable HEP

Current Status and Open Questions

Introduction

Point of this talk and upcoming session

- We've heard about concepts, motivation and challenges for AD in HEP
- Rest of today and first session tomorrow will be about the tools for this
- The 'standard' software ecosystem is quite mature
- Multiple efforts of introducing AD into this ecosystem
 - What is the status of these AD extensions?
 - What are fundamental roadblocks vs technical challenges?

Underlying AD engines

The actual autodiff bit

- The HEP analysis software stack can be split into: **python**-based & **ROOT**-based
- Most python AD tools (HEP & beyond) use Jax under the hood
 - Essentially numpy with gradients, hardware accelerators & JIT compile
 - Great introduction by Peter in last week's stats workshop ([contribution](#))
- AD for C++: Clad is a source-transformation compiler plugin
 - Used for differentiable extensions in the ROOT ecosystem
 - Vassil and Jonas will talk about this tomorrow ([contribution](#))
- Essentially all AD HEP tools build on top of one of these



Clad

Typical Analysis Pipeline

Closer look at the Scikit-HEP Ecosystem

- The typical HEP data analysis pipeline consists of several steps
- “Analysis” refers to steps from columnar input data to final measurement result

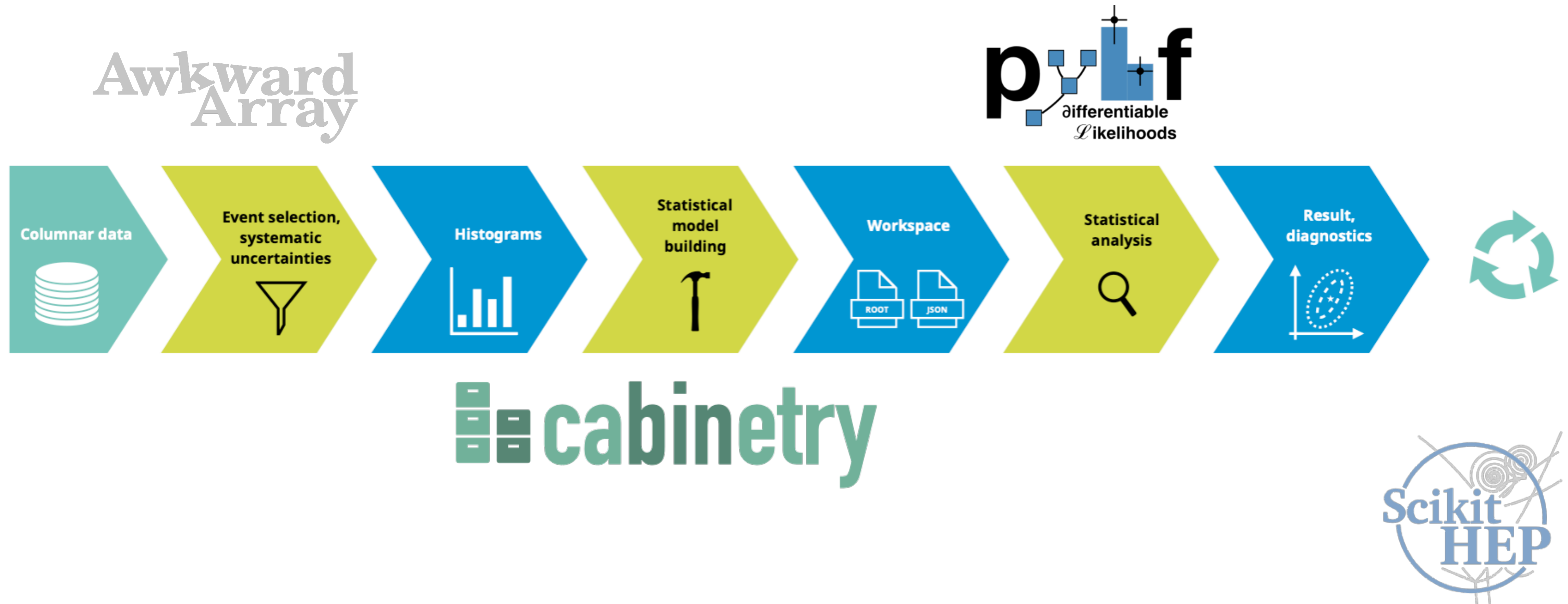


- Scikit-HEP provides an entire ecosystem of python tools for various steps



Typical Differentiable Analysis Pipeline

Closer look at the Scikit-HEP Ecosystem



Typical Analysis Pipeline

Closer look at the Scikit-HEP Ecosystem

Manipulate nested, variables-sized data w/ numpy-like idioms

Awkward
Array

pure-Python HistFactory implementation with tensors and autodiff

py_lf
differentiable
Likelihoods



cabinetry

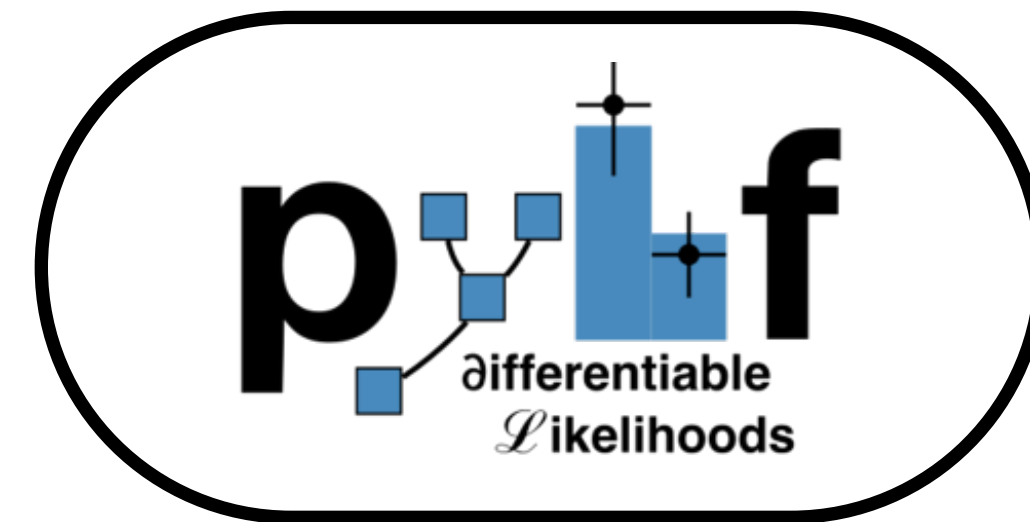
building and steering binned template fits



Typical Differentiable Analysis Pipeline

Making existing tools differentiable

Comes w/ autodiff by default
(via e.g. jax backend)



Awkward
Array

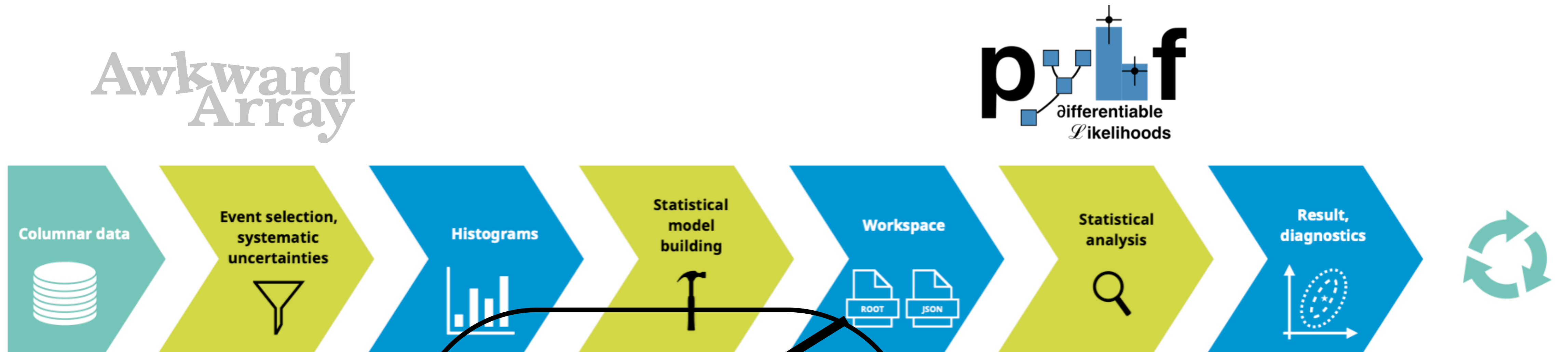


cabinetry



Typical Differentiable Analysis Pipeline

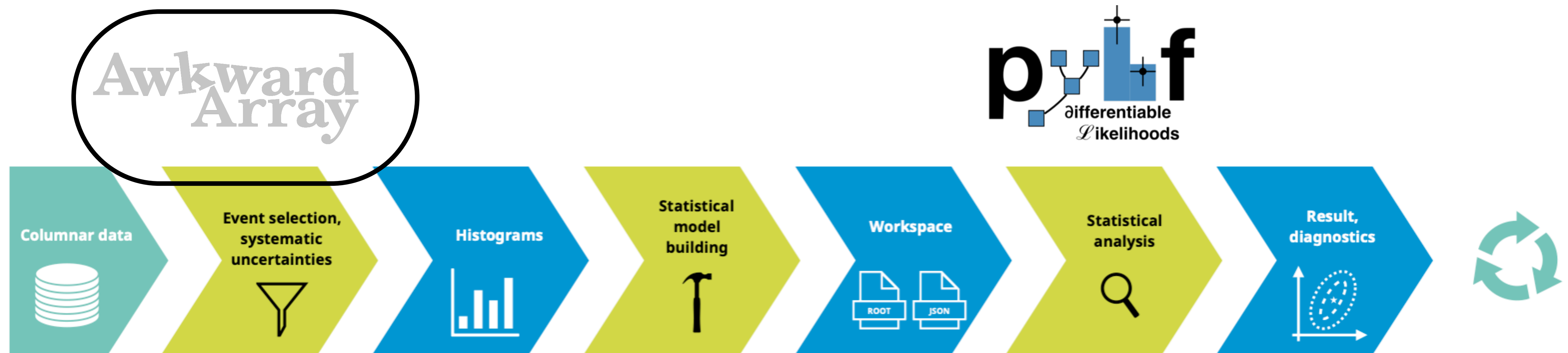
Making existing tools differentiable



Currently a road block:
Creates static, intermediate result
blocks gradient flow
typically omitted for diff'able approaches

Typical Differentiable Analysis Pipeline

Making existing tools differentiable

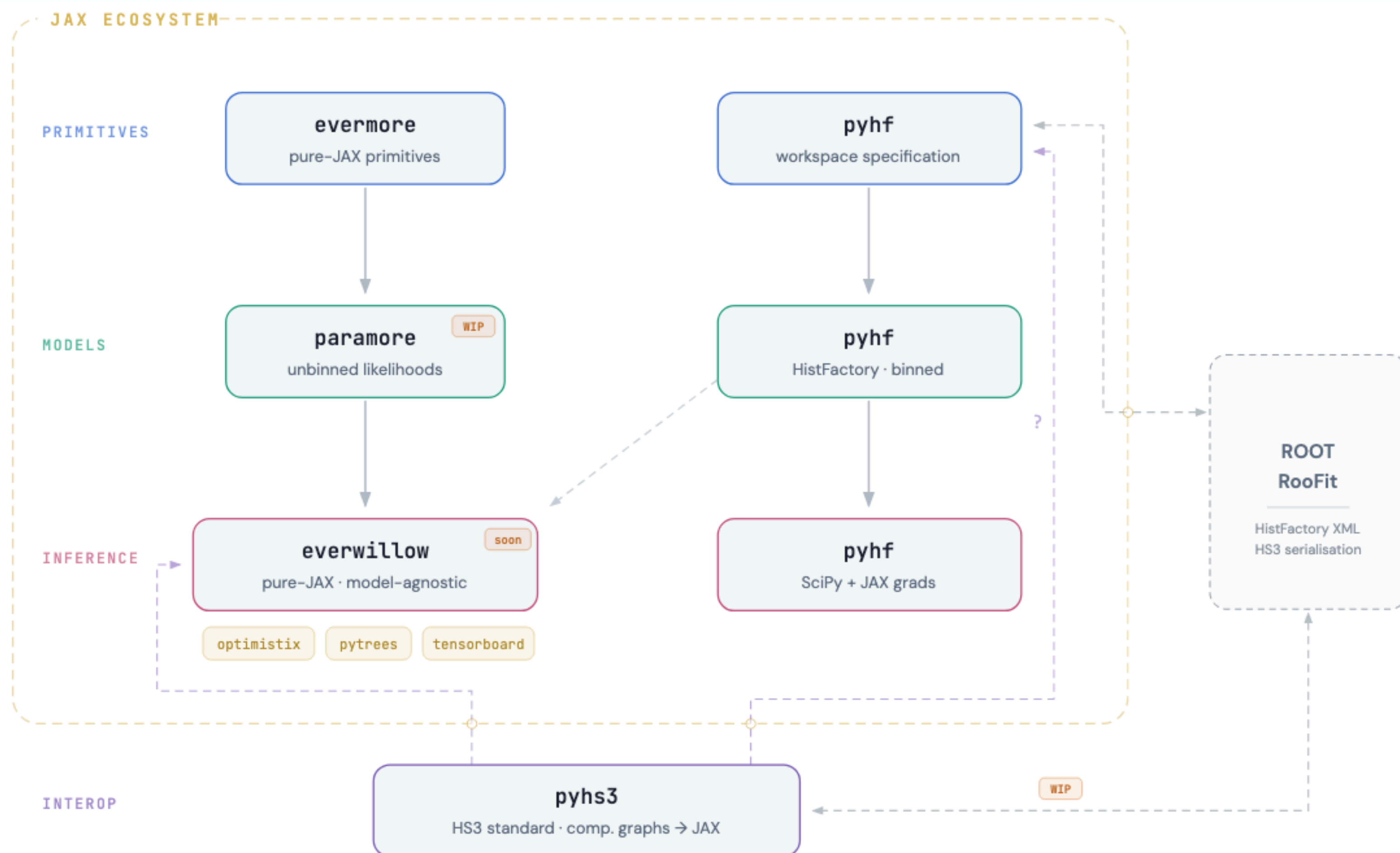


Has a jax backend, but some operations not yet supported
See Ianna's talk later today ([contribution](#))



JAX-based Stats analysis

Splitting Statistical Analysis into several tools

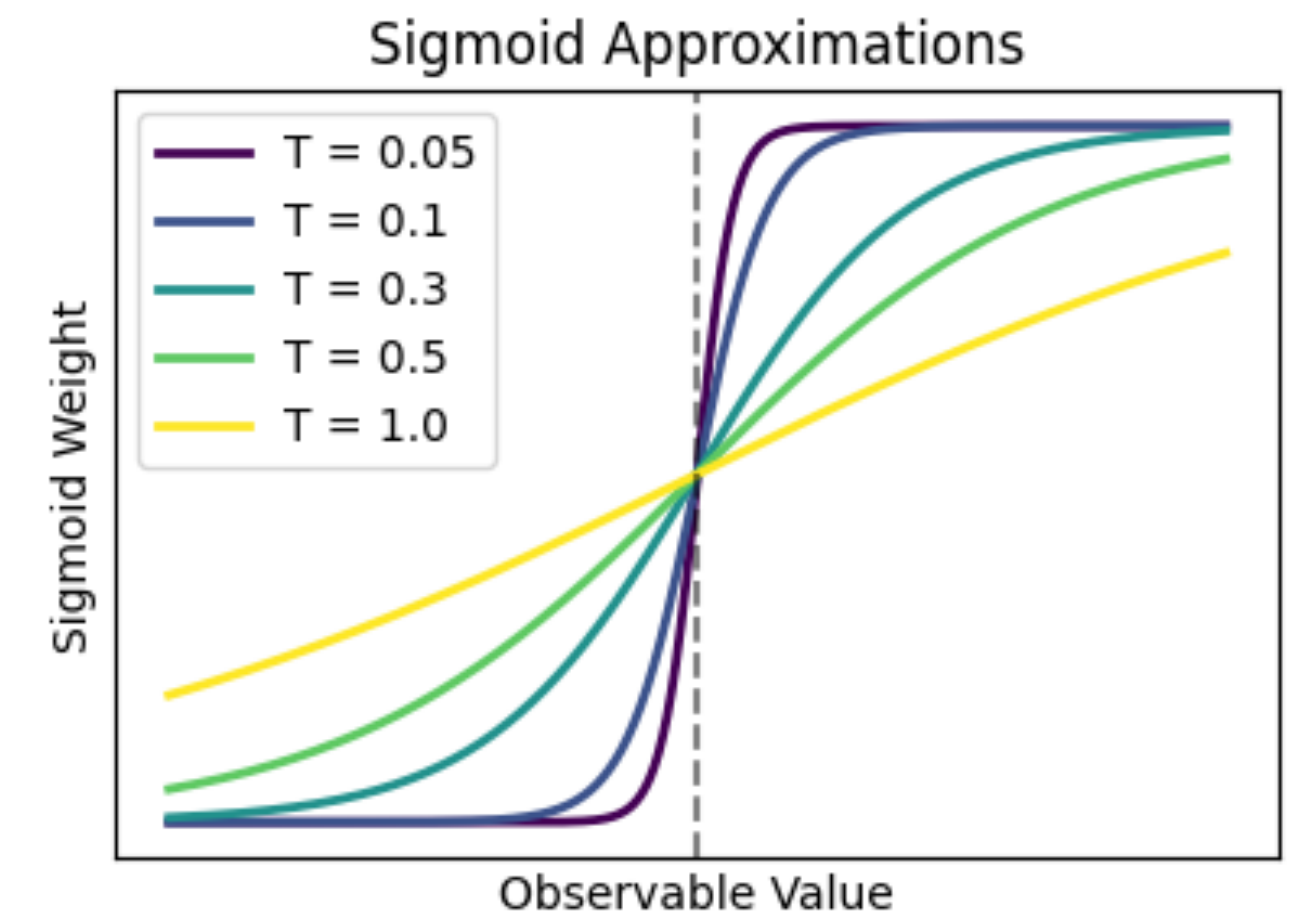


- Ongoing efforts on Jax-native stats ecosystem
- No other backends
- Stronger separation of concerns
- For more info, stay tuned for Peter's talk tomorrow ([contribution](#))

Relaxing Discrete Operations

Wholistic optimizations framework approaches

- Tracking gradients through all operations not yet enough
- We also need 'differentiable relaxations' of discrete operations
 - E.g. approximating cut as sigmoid, histogram as a KDE
 - ... and there is a package for this **relaxed** 😴
- Nitish will talk about differentiable bin optimization right after (contribution)
- Btw: statistical gradients, straight-through estimators, gumbel noise injection (see Annalena's talk earlier today) are not yet supported (but could be)

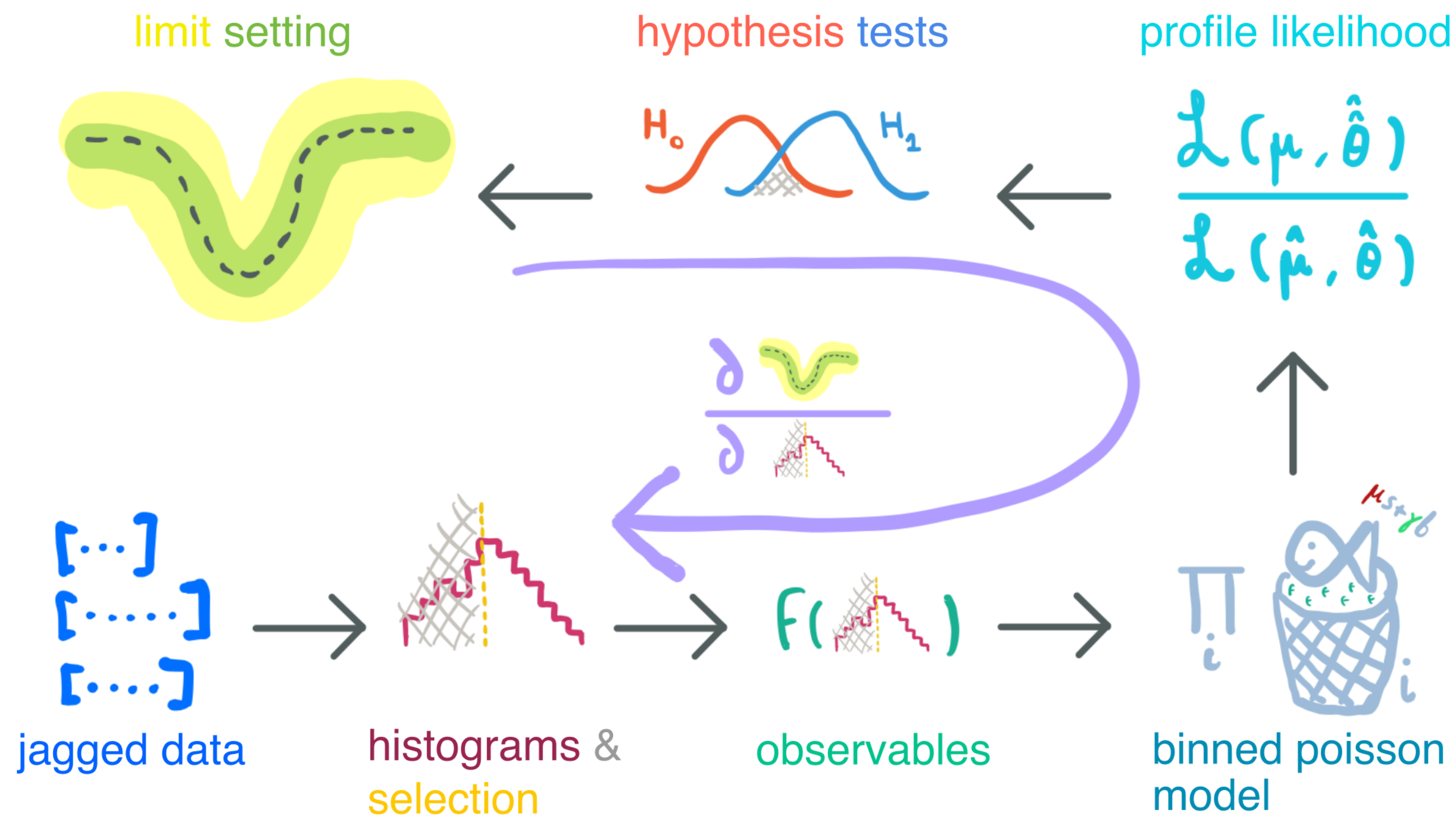


Putting it all together - Frameworks

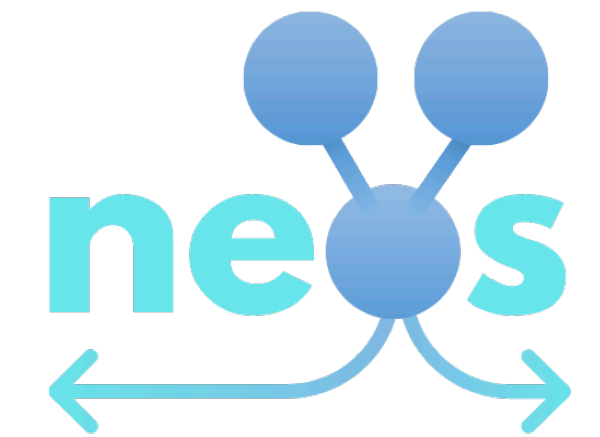
End-to-end optimization

From Nathan Simpson's talk

- Gradients for each step in the pipeline → can optimise all parameters to maximise significance



- This is what neos demonstrated on toy data



- Builds on pyhf
- Only handles rectangular data
- GRAEP does the same thing but with evermore & awkward
- Demonstrated on real CMS data



Recap - AD Tools for HEP

Where we stand

- There are various approaches to introduce AD in HEP
 - Making existing tools differentiable (e.g. awkward-jax or clad for ROOT)
 - Novel Jax-native developments (e.g. evermore)
 - Additional utility libraries (e.g. relaxed)
 - Frameworks putting it all together (e.g. neos, TOMATOS, GRAEP)
- Pros & Cons - and quite some overlap!

Open Questions

Things to discuss later this session

- Shall we support various backends for existing libraries or develop natively-differentiable tools?
 - Tradeoff between flexibility, maintainability and sheer number of packages
- Is there a case for end-to-end differentiable frameworks as a centrally managed packages?
 - In 'classical' analysis, groups of analyses tend to write their own stuff
- How do we scale differentiable approach to distributed computing?
 - Unlike distributed training of large ML models, HEP workflows are much more diverse (HPC vs HTC)
 - How do we pass gradients through tools like coffea-case, serviceX, etc?
 - This was discussed a few years ago: [google doc](#). Things have changed. Is there still interest?

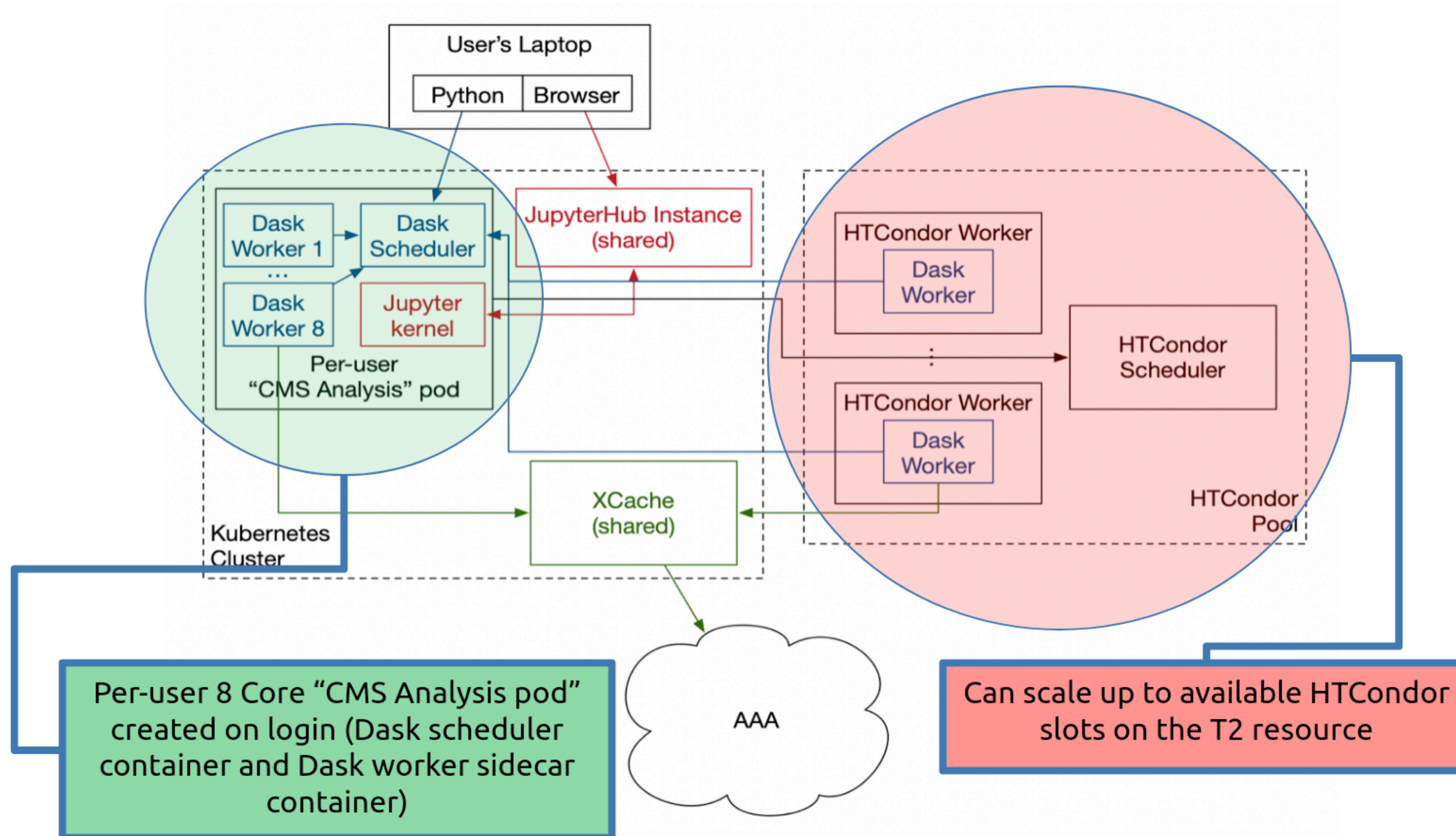
Open Questions II

Scaling to distributed computing

- Batching: Does not work out-of-the-box like in classic ML
 - In ML, the loss is defined for a single sample and can be averaged over a batch
 - In our case, we need a representative subset of the data to calculate the objective
 - E.g.: we cannot calculate the significance on 1000 background events alone
- Can we use a mixture of batched- and unbatched?
 - E.g. run the fit on running, global histograms, that are updated in batches through preselection

Backup

Coffea-casa

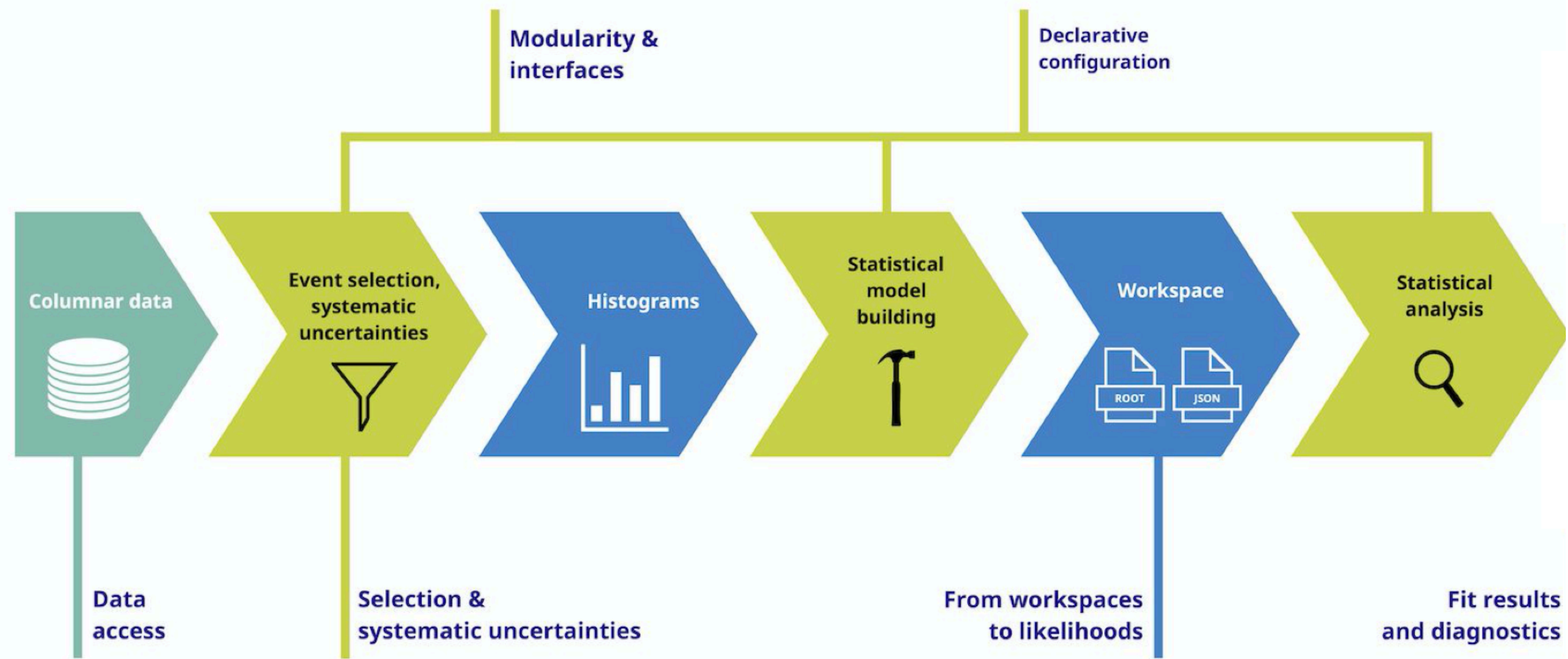


Typical Analysis Pipeline

Modified from [IRIS-HEP website](#)



func_adl
formulate



ServiceX

uproot

Awkward Array
VECTOR



coffea



differentiable Likelihoods

