“Agile Analysis”

hep_tables

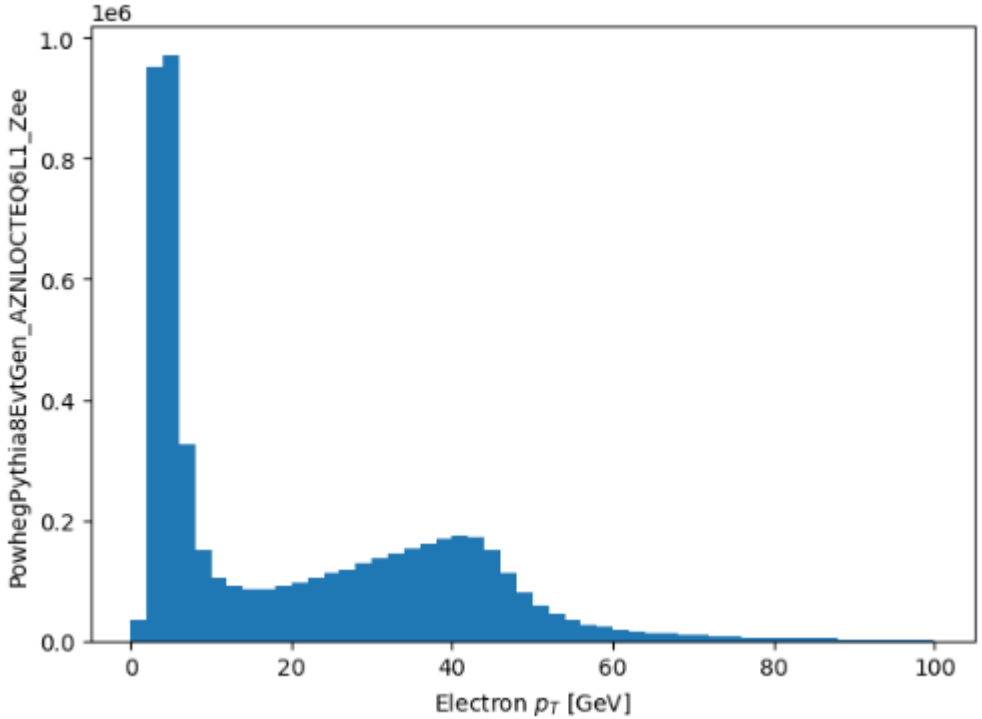
G. Watts (UW/Seattle)

Analysis Facilities Blueprint Meeting

Oct 26, 2020

Make a mass plot of $Z \rightarrow ee$ electron p_T

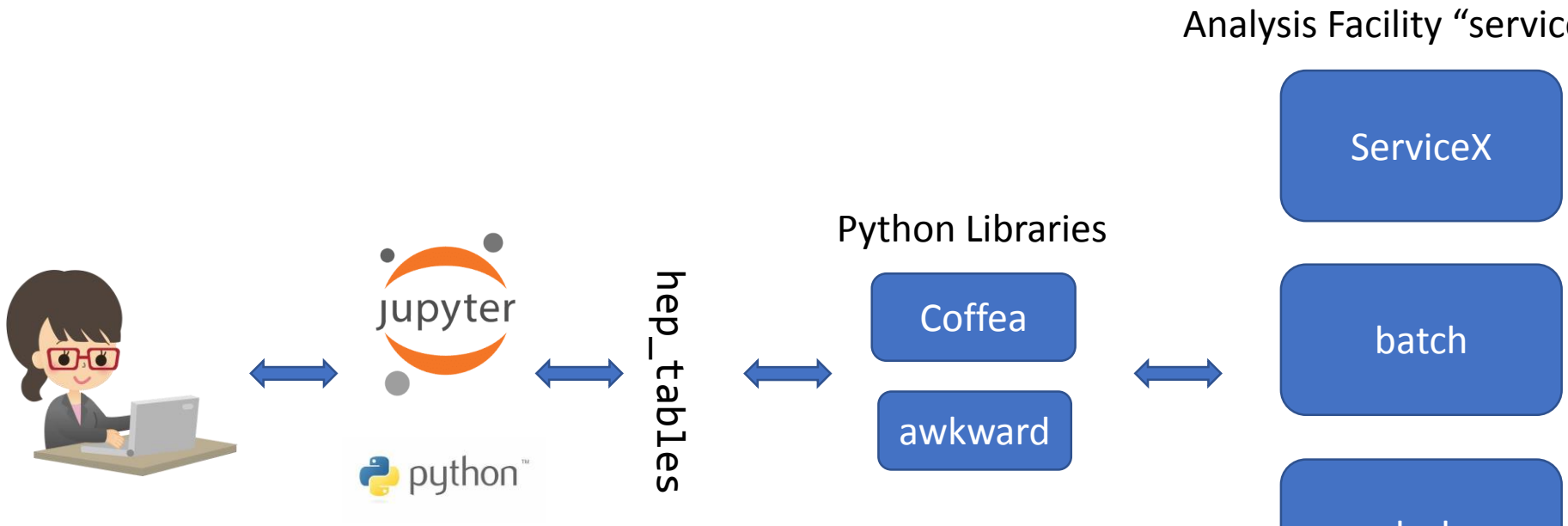
```
1 dataset = EventDataset('mc15_13TeV:… ATLAS XAOD production dataset')
  df      = xaod_table(dataset)
2 pts     = df.Electrons('Electrons').pt/1000.0
3 np_pts  = make_local(pts)

4 plt.hist(np_pts.flatten(), range=(0, 100), bins=50)
  plt.xlabel('Electron $p_T$ [GeV]')
  _ = plt.ylabel('PowhegPythia8EvtGen_AZNLOCTEQ6L1_Zee')
```



1. Onetime boilerplate to indicate the dataset you want
2. Extract the electrons from the "Electrons" xAOD bank, access the p_T property of the object, and turn it from MeV to GeV
3. Get the column of electron p_T from the dataset using ServiceX.
4. Plot it. np_pts is an JaggedArray when it comes back.

Jupyter Notebook Demo

A thin layer



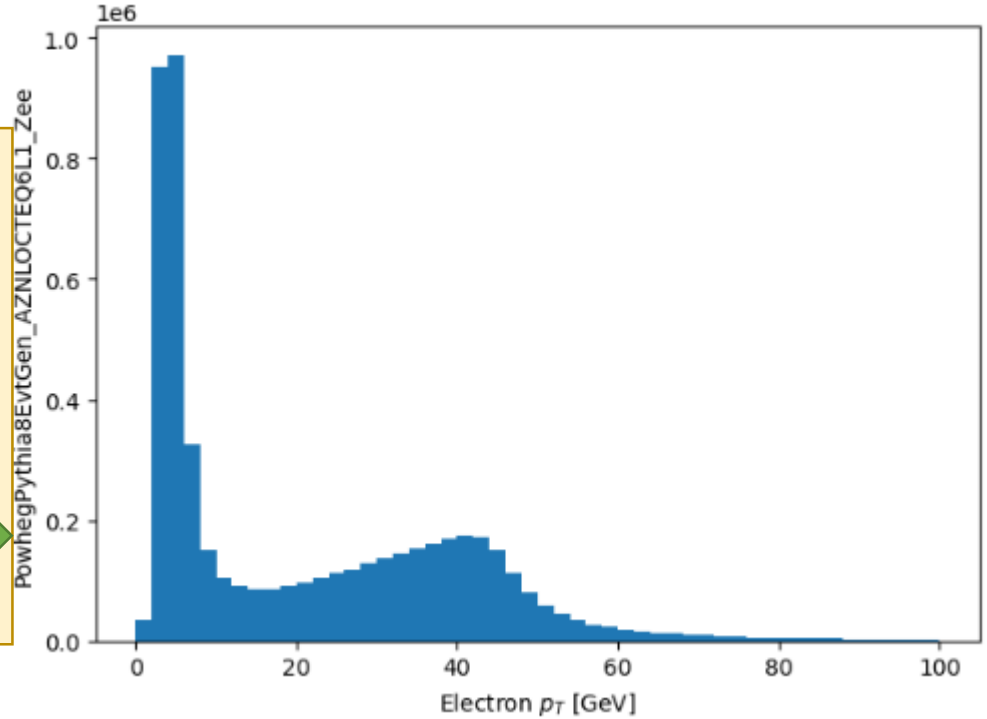
- Hide access to data infrastructure
- Use array programming with multi-object extensions
- Form DAG with everyone can manipulate
- Support histogramming and dataset creation
- Decides where to do what

This diagram isn't totally accurate: `hep_tables` does not mean to hide/abstract away libraries and services…

Make a mass plot of $Z \rightarrow ee$ electron p_T

```
1 dataset = EventDataset('mc15_13TeV:… ATLAS XAOD production dataset')
  df = xaod_table(dataset)
2 pts = df.Electrons('Electrons').pt/1000.0
3 np_pts = make_local(pts)
```

- Dataset located on the GRID
  - ServiceX uses rucio to fetch it
- Code sent to ServiceX to extract the p_T of the electrons
- Locally execute the divide by 1000
- And return an awkward array
- Histogram proceeds as would normally be expected.



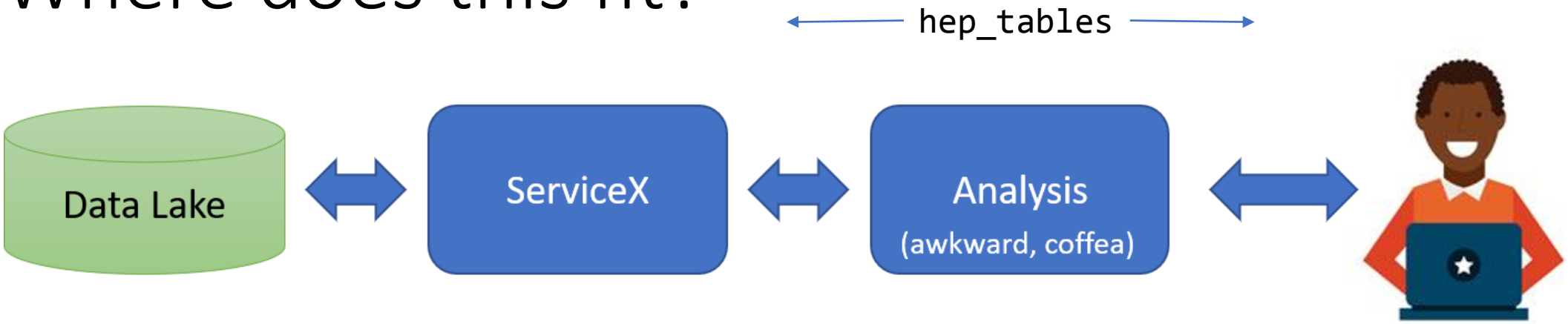
2 Extract the electrons from the "Electrons" xAOD bank, access the p_T property of the object, and turn it from MeV to GeV

3 Get the column of electron p_T from the dataset using ServiceX.

4 Plot it. np_pts is an JaggedArray when it comes back.

Jupyter Notebook Demo

Where does this fit?



What is it good for?

- Jupyter Notebook style analysis
- Quick plots
- A world where columns exist on demand

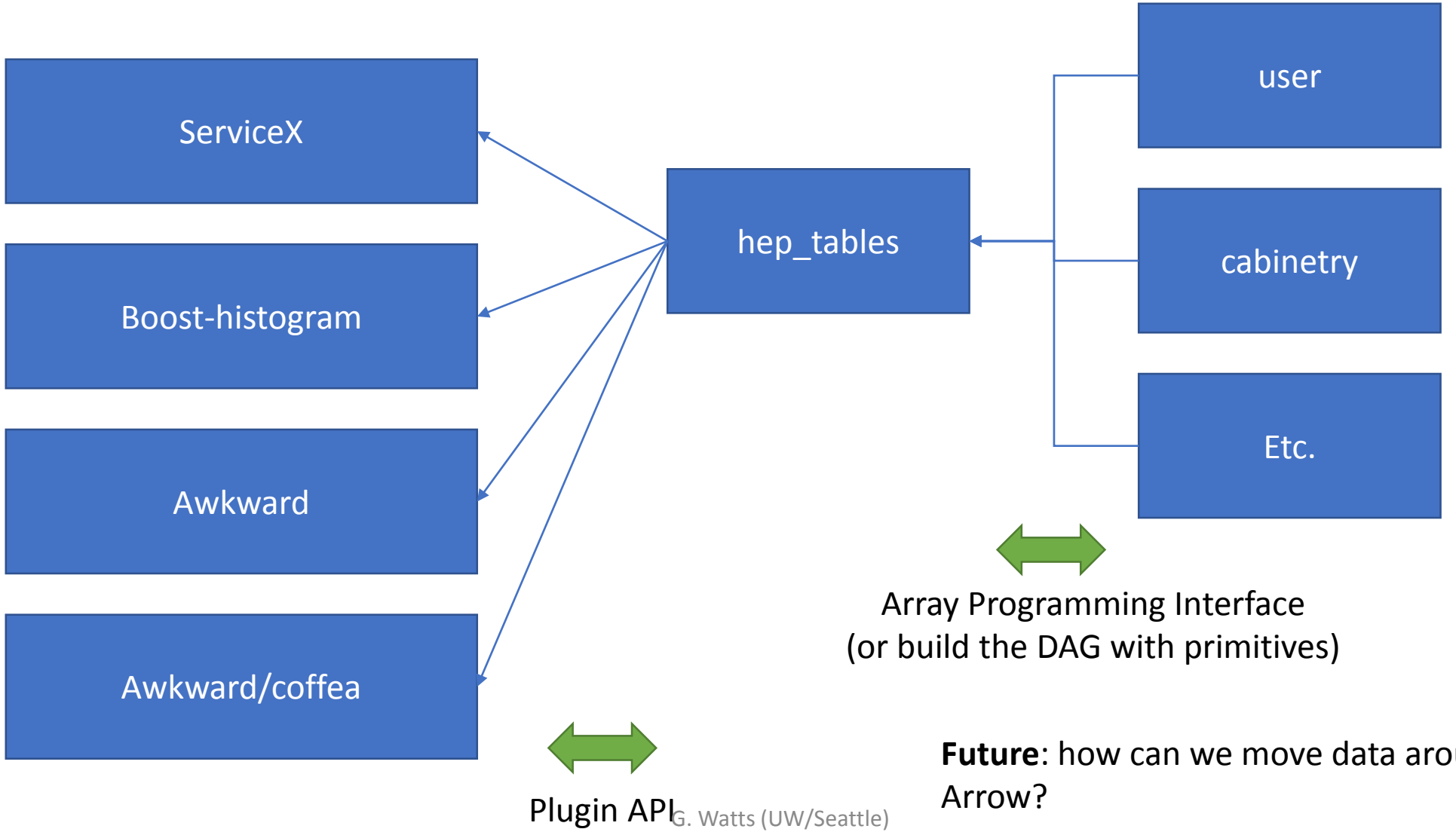
Not so good?

- A job to make 1000 plots
- Careful tuning of backend and how things work

Or pyhf/cabinetry, etc.

Get away from user making decisions like how much to ask of ServiceX, how to run on DASK or on Spark to fill histograms, etc.

How will it interface with other systems?



Future: how can we move data around internally? Arrow?

Current State

Prototype on github:

dataframe_expressions, hep_tables, hl_tables, some example docs

- Prototype was used in ATLAS MC Study for an Exotics Analysis
  - Plug-ins:
    - ServiceX
    - single-threaded awkward
    - and histogramming
- Upon turning to actual analysis, heuristics used in prototype proved too fragile

Rewrite under way now:

- Plug-in system better designed
- Uses python type hints to steer it through ambiguous situations (if available)
- Much more capable ServiceX interface than before

Rewrite has not yet left the stable…

Big Questions

How to naturally include full fidelity access to the python eco-system

- For example – boost.histogram
- And yet fill histograms on multiple backends using coffea.

Eco-system has produced a huge amount of work: need to make sure to take advantage of it!

Just discovered some overlap in functionality with coffea's nanoEvents

- Will need to be sorted out

More forward thinking

- User libraries in C++/C
- Training and inference
- Differentiable Programming

How does this fit in an Analysis Facility?

Presumably, an analysis facility will have some best practices for running large analysis jobs:

- A ServiceX instance
- A local data cache
- A particular way to spawn workers to fill histograms, etc

A personality

I hope this will be discoverable!

So this code will use services already provided by the facility – it doesn't necessarily require new services.

- Some infrastructure includes the proper front-end code
- Then one can move your code from facility to facility and it will "run"

(minus leaky abstractions)

Backup!

Easy things should be easy…

…Hard things should be possible

hep_tables is a user-facing library → With a plug-in model to orchestrate the work

Library is based on delayed execution and building a DAG

Array programming:
- Standard access to properties, like the p_T of a jet.
- Standard access to methods of objects
- Filtering on object properties and event-level quantities

Some SQL-like extensions to make things more readable
- Object associations (like jet-track matching)
- Think per-event, not per-column as much as possible

Event Data Model is Extendable
- New properties are just expressions
- Encourages sharing within a group
- xAOD/miniAOD and PHYSLITE/nanoAOD type formats

Builds a DAG

Easy things should be easy…

…Hard things should be possible

hep_tables is a user-facing library → With a plug-in model to orchestrate the work

Builds a DAG → Plug-ins analyze the DAG and execute the parts they are best suited to

- ServiceX plug-in fetches the data
- Awkward plug in executes higher level functions, histogramming, etc.
- Other plugins – run in coffea processors, etc.

Though not tested, it should be possible to transmit the DAG over the wire to a facility if desired