



Me!

2

- Ph.D. University of Rochester, CDF (Fermilab!), Top Quark, SVX Interlock
- Post-doc at Brown University on DZERO, Single Top Quark, Higgs, DAQ
- Professor at University of Washington, DZERO/ATLAS/MATHUSLA/IRIS-HEP, Long Lived Particle/Hidden Sector Searches



The intersection between the physics, hardware, and computers is what drove me into this field, and, initially, DAQ

Working on Analysis Systems trying to make it possible for a professor to make a plot.

func-adl

Stream columns of data from an ATLAS xAOD to a Jupyter Notebook

- ▶ Declarative Data Query Language
 - ▶ What does the physicist interface look like?
- ▶ Python
- ▶ Feeds awkward array and numpy

[Demo on Binder](#) (click on binder badge)

Extracting Jet p_T

We define the dataset. The URI scheme `localds` tells the system that we want to go after a dataset from the GRID, but downloaded locally.

```
In [3]: ds = EventDataset('localds://mc16_13TeV.311309.MadGraphPythia8EvtGen_A14NNPDF31LO_HSS_LLP_mH125_mS5_ltlow.deriv.DAOD_EXOT15.4')
```

Next, we build the query. At the end we get a future that will contain the PandasDF with the jet p_T 's. In ATLAS jet p_T 's are in units of MeV, so we convert them to more sensible GeV.

```
In [4]: df_future = ds \
        .SelectMany('lambda e: e.Jets("Antikt4EMTopoJets")') \
        .Select('lambda j: j.pt()/1000.0') \
        .AsPandasDF('JetPt')
```

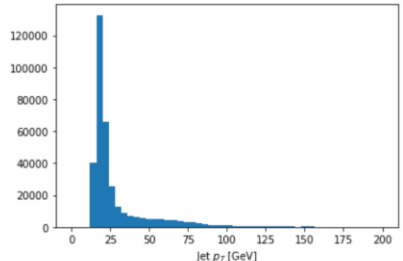
The final thing is to turn the future into something real. We tell it to use the `func-adl` server as a back end. This can take a little while:

- If the xAOD's haven't been downloaded, they must be.
- If the JetPt's have to be extracted from the xAOD's, then that must occur.

```
In [6]: df = await df_future.future_value(executor=lambda a: use_exe_func_adl_server(a, node=endpoint))
```

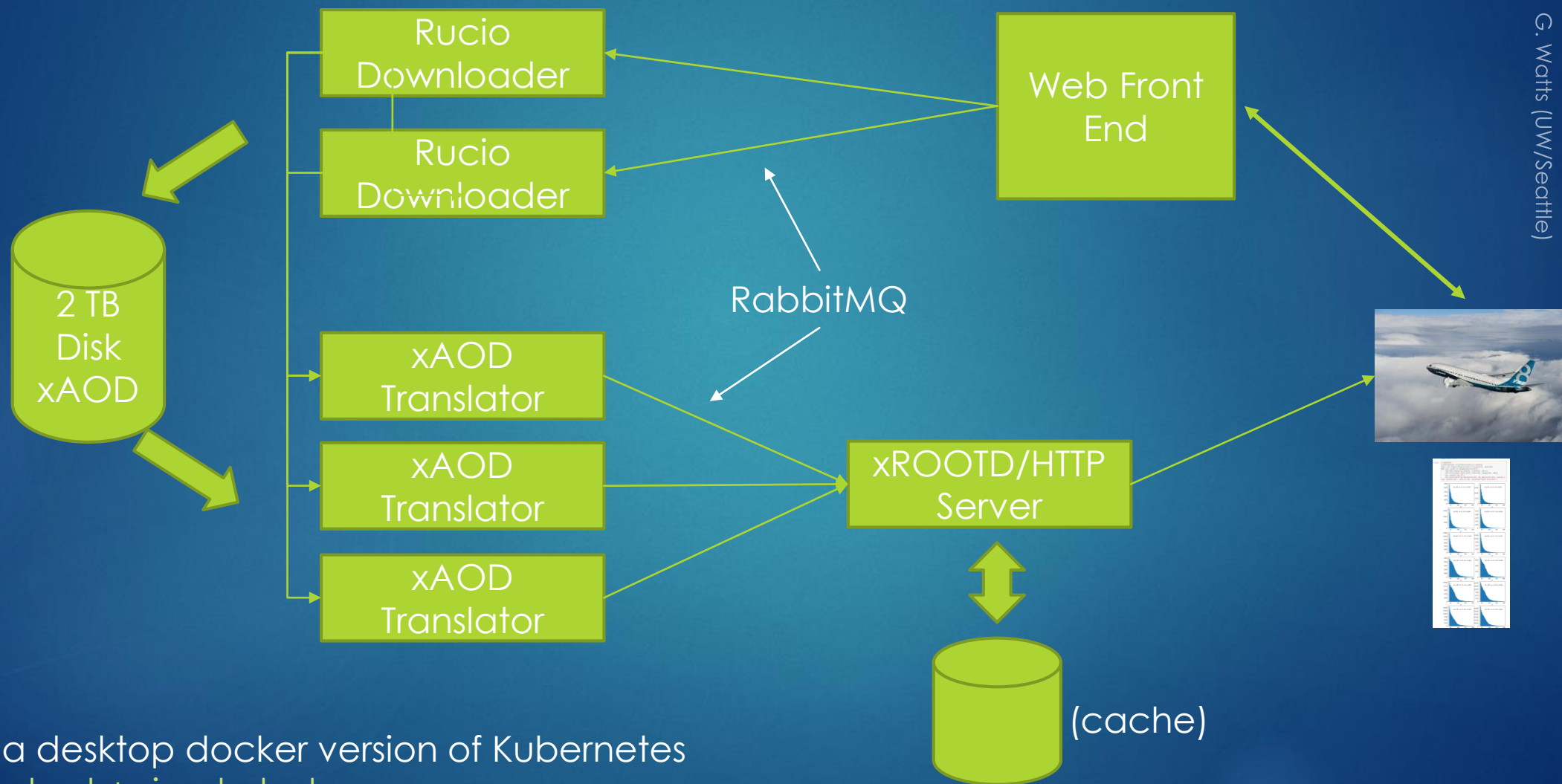
And plot the p_T .

```
In [7]: plt.hist(df.JetPt, bins=50, range=(0,200))
plt.xlabel('Jet $p_T$ [GeV]')
plt.show()
```



The histogram displays the distribution of jet transverse momentum (p_T) in GeV. The x-axis ranges from 0 to 200 GeV with major ticks every 25 units. The y-axis represents the number of events, ranging from 0 to 120,000 with major ticks every 20,000 units. The distribution is highly peaked at low p_T , with a maximum frequency of approximately 120,000 events occurring between 10 and 20 GeV. The frequency drops sharply as p_T increases, following a power-law-like decay, and reaches near zero by 100 GeV.

Design



G. Watts (UW/Seattle)

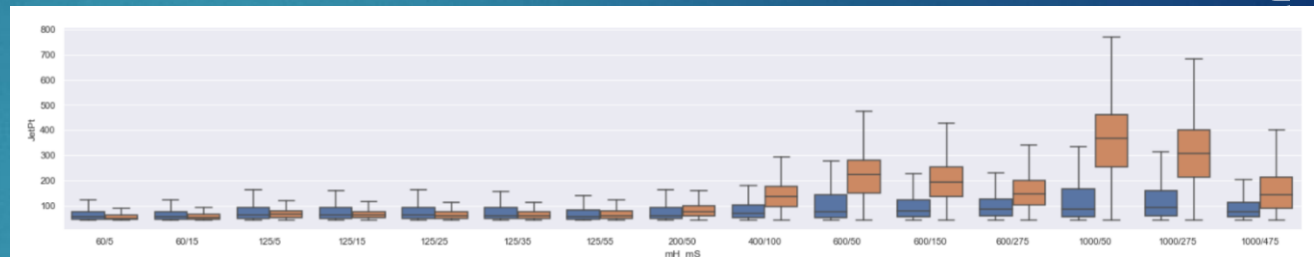
Used a desktop docker version of Kubernetes
Helm chart + simple tests

Good

Dogfooding:

- LLP Analysis Per-Jet Training Set
- 57 MC Datasets (~3 TB of files)
- ~200 M events?
- 20 Minutes to extract 25 columns on a core i5 single machine (5 yrs old)
- Compressed columns: 2 GB
- In core memory: ~8-10 GB
- “multiuser”

Analysis Plots Notebook



So little python code for plots with a huge amount of information!

$$Z \rightarrow e^+ e^-$$

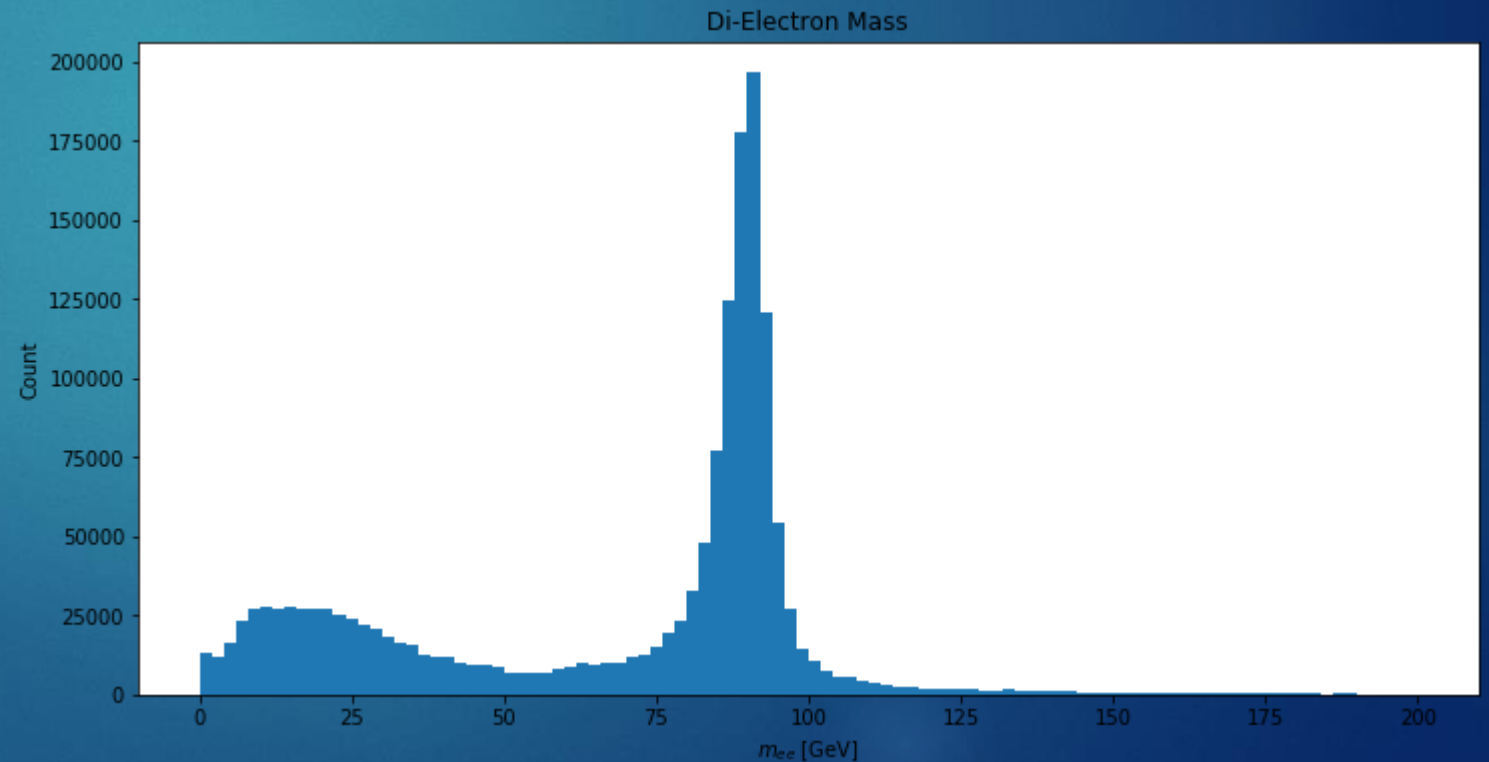
Simple physics example:

- Fetch electrons in each event
- Use the first two to form a mass
- Plot

[Jupyter Notebook](#)

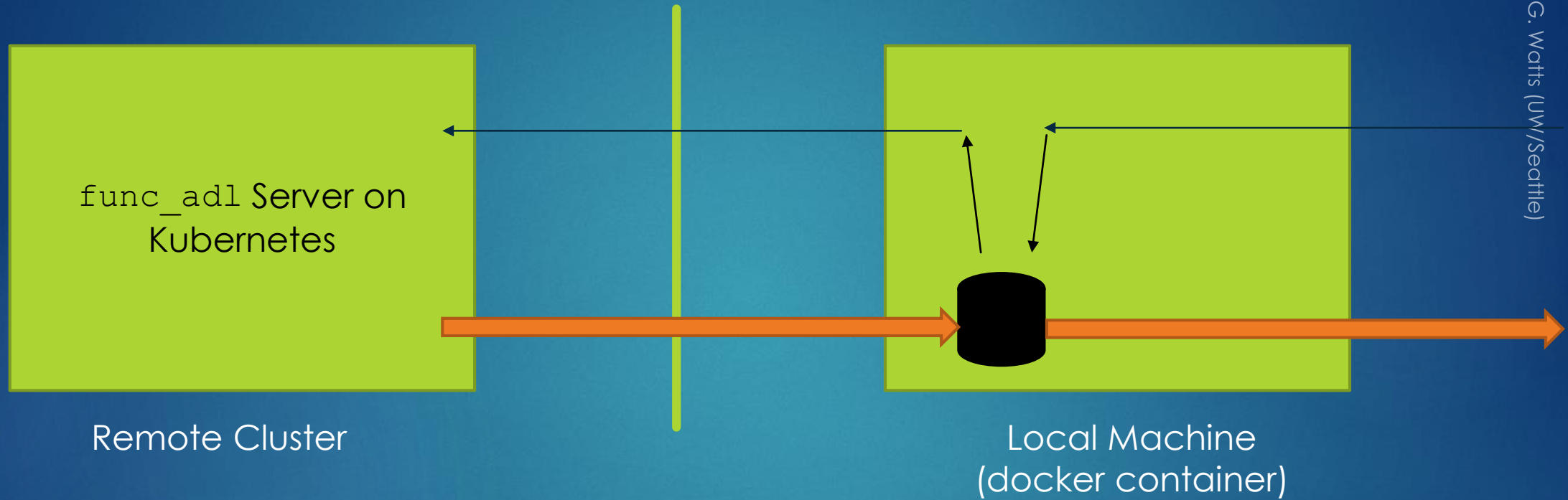
G. Watts (UW/Seattle)

- 70 GB dataset
- Takes about a minute for a single process to run through all these files.
- But without cuts, takes a long time before the data is transferred!! (not on binder)



Caching

7



I restart my Jupyter kernel 4-10 times a day, depending...

Running the Server

- K8 and helm charts means I could run a multi-container application without hand art.
- Crucial to be able to run the complete server on my laptop
 - And desktop
 - And server
 - Ran in *production* in all three locations at various times
- Rabbit MQ
- We should at least be familiar with these tools



I think this is the way we will deploy even on our laptops

Chart

```
helm install https://github.com/gordonwatts/func_adl_server/releases/download/0.4.1/func-adl-server-0.4.1.tgz
```

(you have to pass in certificates for full functionality)

Query

Caches Data Locally

- No waiting on cache for small repeated queries

Query is uniquely hashed

- Lookup is just a re-query

Output is ROOT

- Pretty much everything can read it
- Need it for all systems in the future

Cache is hard

- Keeping robust interactions when laptop is closed, open, loses wifi is hard!
- Lots of bugs

Python isn't the best language

- Not designed to deal with streams of data without modification

Python Async is a mess

- Buy you can run 57 requests at once with very little work.

What is Next?

10

G. Watts (UW/Seattle)

Backend is good enough to power my analysis

Backend is good enough to power my analysis

Align with ServiceX
Run on a different K8 cluster

Language Exploration