Functional Data Query and other things at 

M. PROFFITT, E. TORRO, G. WATTS

UW/SEATTLE

AS WORKSHOP – NYU – JUNE 19-20 2019

Intellectual Hub Efforts

G. Watts (UW/Seattle)

API & User Stories

- Collect a series of high-level (complex) analysis user stories
- Grew out of work on one of our milestones
- as-user-facing on GitHub
- Use those to break down tools and API's we need
- Lots of work by Kyle, also Mason, Emma, Jim
- Discussion/Breakout later today

Benchmarks

- How me how your data query language works!
- Common set of Challenges, and each language implementor or designer writes a repro that gets linked to this one.
- 8 challenges so far, ranging in complexity.
- adl-benchmarks-index on GitHub (RDataFrame and nail are linked already).
- Grew out of a CHEP presentation in Bulgaria

HSF Interactions

- Discussions on many topics
- Fascinating one on what will an analysis system look like (hardware, university cluster, etc.)

It is good to start these discussions when meaningful – there are a lot of new ideas, and a lot of active people on these mailing lists.

Express a query over data in a clear, concise, and unambiguous way.

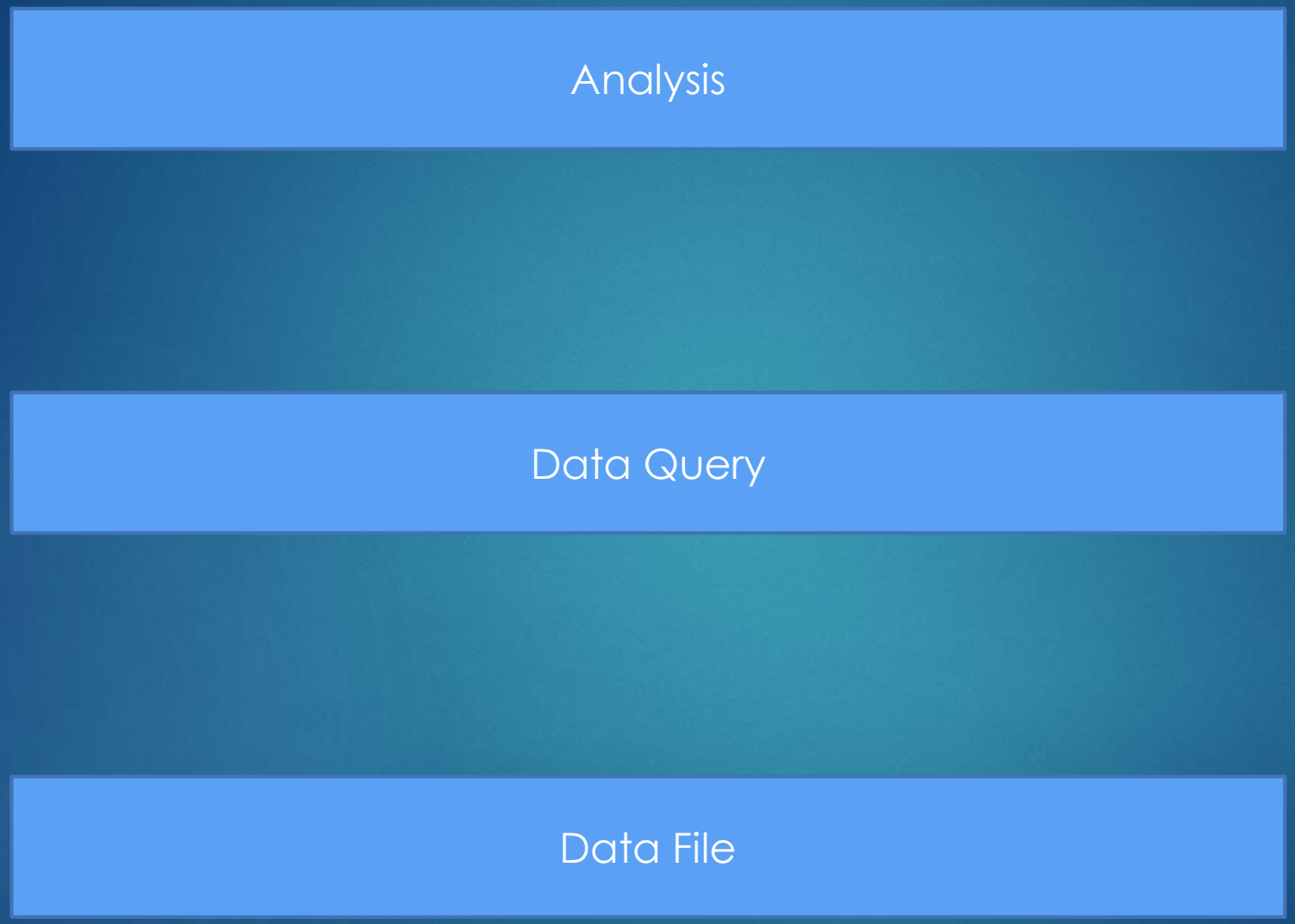
Mason Proffitt
Graduate Student
(ATLAS, MATHUSLA)

Enna Toro
Post-doc
(ATLAS, MATHUSLA)

Gordon Watts
Prof
(ATLAS, MATHUSLA)

All of us are physicists (experts in Searches for Long Lived Particles)

We use this to drive our IRIS-HEP work


Analysis
Data Query
Data File

Analysis

Data Query

Data File

- Various Binary Formats
- Lots of files
- Blobs
- Hopefully hidden from user

Analysis

Data Query

- Aggregate the data into histograms, counts, or even tables

Data File

- Various Binary Formats
- Lots of files
- Blobs
- Hopefully hidden from user

Analysis

- Ratios of histograms
- Fitting, Likelihoods
- Limit Plots

Data Query

- Aggregate the data into histograms, counts, or even tables

Data File

- Various Binary Formats
- Lots of files
- Blobs
- Hopefully hidden from user

Analysis

- Ratios of histograms
- Fitting, Likelihoods
- Limit Plots

Q: Where does ML belong?

Data Query

- Aggregate the data into histograms, counts, or even tables

Data File

- Various Binary Formats
- Lots of files
- Blobs
- Hopefully hidden from user

"Host Language"

Analysis

- Ratios of histograms
- Fitting, Likelihoods
- Limit Plots

"Query Language"

Data Query

- Aggregate the data into histograms, counts, or even tables

"Wild West"

Data File

- Various Binary Formats
- Lots of files
- Blobs
- Hopefully hidden from user

Individual tracks, electrons, jets, calorimeter clusters, etc. Unfiltered.

Jets, with near by tracks, and near by MC particle information. Very filtered.

Histograms of jet p_T with $p_T > 50$ GeV, and 2 tracks near by with $\Delta R < 0.4$, etc.

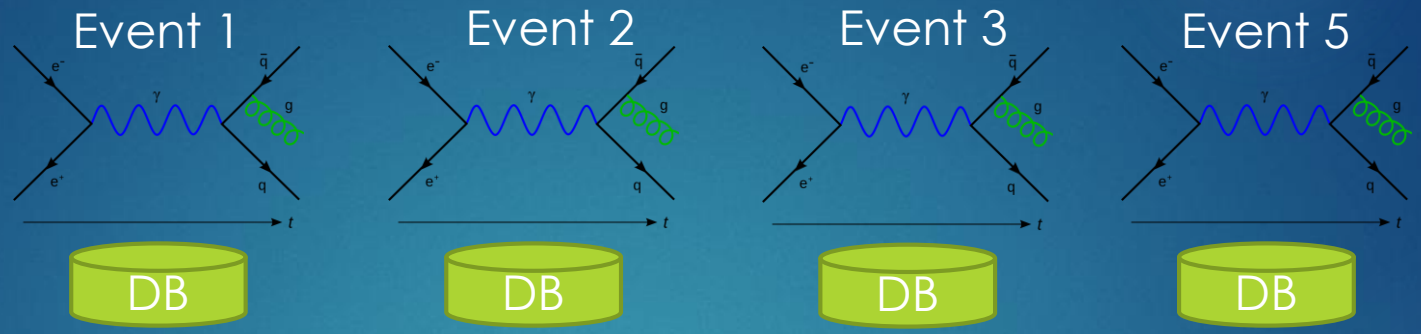
Building this relationship is the focus of this work

Individual tracks, electrons, jets, calorimeter clusters, etc. Unfiltered.

Jets, with near by tracks, and near by MC particle information. Very filtered.

Histograms of jet p_T with $p_T > 50$ GeV, and 2 tracks near by with $\Delta R < 0.4$, etc.

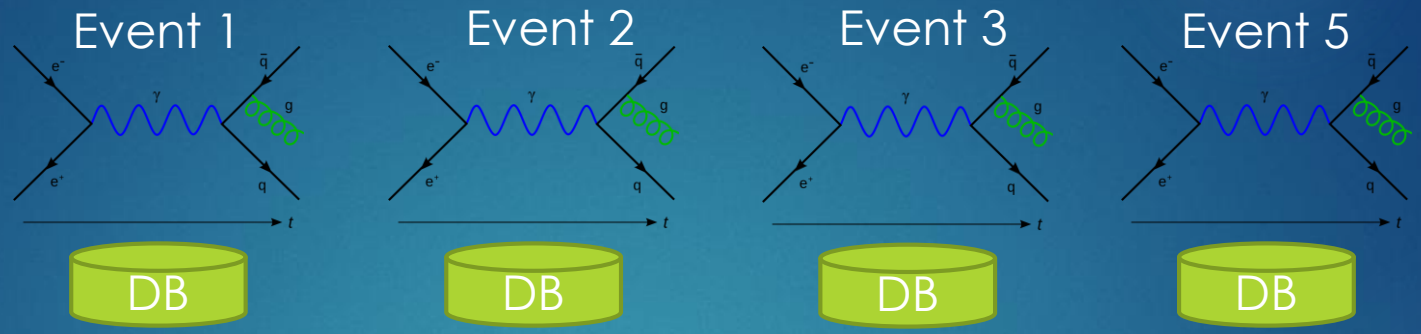
Reconstructed Data



Each event is a small structured DataBase

- The principles behind SQL have a firm theoretical underpinning
- Copy their semantics

Reconstructed Data



Each event is a small structured DataBase

- The principles behind SQL have a firm theoretical underpinning
- Copy their semantics

Is it complete?

1. Start with reconstructed data, unfiltered
2. Generate training data for a LLP Calorimeter Decay:
   - Train on jets with $p_T > 40$ and $|\eta| < 2.5$
   - Jet kinematics
   - MC Particle if a LLP is near ($\Delta R < 0.4$)
   - Fraction of energy that is in each layer of the calorimeter

```
e.TruthParticles('TruthParticles').Where(lambda tp1: tp1.pdgId() == 35)
```

Represents all the data in an event (the DB)

Select out just the Truth Particles (e.g. just that table)

Limit to rows where there is a LLP particle only

(this is all valid python, btw)

This is a tuple of event info (run number, etc.), the jets, and the LLP particles.

```
event_info = events \
.Select("lambda e: (e.EventInfo('EventInfo'),
                    e.Jets('AntiKt4EMTopoJets'),
                    e.TruthParticles('TruthParticles').Where(lambda tp1: tp1.pdgId() == 35))")
```

We have built a new per-event database consisting of only this information.

Need a row per-jet.
- Need to flatten the array

```
jet_info = event_info \
.SelectMany('lambda ev: ev[1].Select(lambda j1:
    (ev[0],
     j1,
     ev[2].Where(lambda tp2: DeltaR(tp2.eta(), tp2.phi(), j1.eta(), j1.phi()) < 0.4)))')
```

- For each jet, filter out only the MC LLPs that are close.
- The result might be 0, 1, or 2 of these particles (no more due to physics)

Syntax allows for no ambiguity
- Operating over each jet, looking at all MC particles

```
jet_info = event_info \
.SelectMany('lambda ev: ev[1].Select(lambda j1:
    (ev[0],
     j1,
     ev[2].Where(lambda tp2: DeltaR(tp2.eta(), tp2.phi(), j1.eta(), j1.phi()) < 0.4)))')


# Build us a list of columns
tc = track_columns()
tc.add_col('RunNumber', 'ji[0].runNumber()')
tc.add_col('EventNumber',
'ji[0].eventNumber()')

tc.add_col('JetPt', 'ji[1].pt()/1000.0')
tc.add_col('JetEta', 'ji[1].eta()')
tc.add_col('JetPhi', 'ji[1].phi()')
```

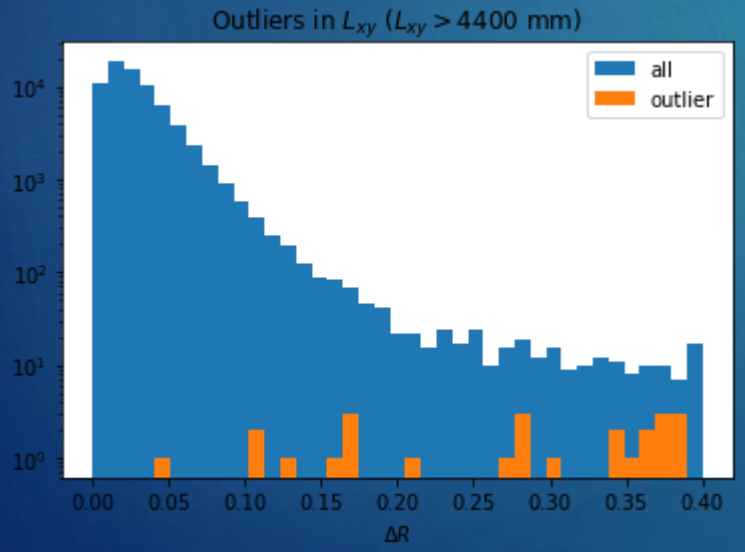
Building a flat table-like structure (numpy) for training input

There are a few other tricks

Like how do you calculate the layer fractions given just the energies

```
ji[1].getAttributeVectorFloat("EnergyPerSampling")['EMM_BL0']
    / ji[1].getAttributeVectorFloat("EnergyPerSampling").Sum()
```

Built in aggregate operation!



(see source on github)

How does this work?

Why AST?

- Python has built in *ast* library to manipulate it.
- Transformation turns *Sum()* into *Aggregate(0, lambda i: i+1)*.
- Transformation turns *.TruthParticles('TruthParticles')* into some C++ leaky-abstraction code
- Captures complete meaning
- Can be sent across the wire

If we had a DAG/AST language – we could interoperate:

- Front End Languages in your favorite language
- Write the complex distributed backend code once

Back Ends

1. ATLAS binary xAOD
   - Most complete implementation
   - 30 seconds for 40K events (15 to compile, 10 to initialize)
2. RDataFrame
   - Complex ROOT TTree's that don't need extra libraries
3. Columnar
   - Works with awkward arrays as back end

xAOD to Jet p_T's

In [4]:
```
%%time
jet_pts = events \
            .SelectMany("lambda e: e.Jets('AntiKt4EMTopoJets')") \
            .Select("lambda j: j.pt()")
```

```
Wall time: 0 ns
```

And finally turn it into a pandas data frame. Each row has a single jet pt.

At this point no execution happens - a "future" is now setup with the pandas dataframe.

In [5]:
```
%%time
training_df = jet_pts.AsPandasDF(columns=['JetPt'])
```

```
Wall time: 0 ns
```

Finally, we turn it into something real. The following steps occur:

1. Some C++ code is written to access, read the xAOD jet's, create a ttree, and write it out.
2. A docker container with the ATLAS envrionment is started. The code and the data file location are mapped into it.
3. The code is compiled, and then run.
4. The output ROOT file is loaded with uproot and a DF is created.
5. All that temporary code and data is removed.

There is a fair amount of log file information that appears in the notebook engine window. At the moement that is not correctly redirected here - so you can see evidence of it running there.

In [6]:
```
%%time
df = training_df.value()
```

```
Wall time: 29.7 s
```

(notebook)

Simple Columnar Selection

Create query to just pull the first column out:

```
In [9]: simple_col1_query = simple_array_stream.Select("lambda e: e.col1")
```

Then actually run the query:

```
In [10]: simple_output = simple_col1_query.value()
```

Output array contents:

```
In [11]: simple_output
Out[11]: array([0., 2., 4., 6., 8.], dtype=float32)
```

(example)

RDF to stream of Event Numbers


```
f = EventDataSet("data16_iso_728.root")
events = f.AsRDFEvents()
output = events.Select('lambda e: e.eventNumber').value()
print(output)
```

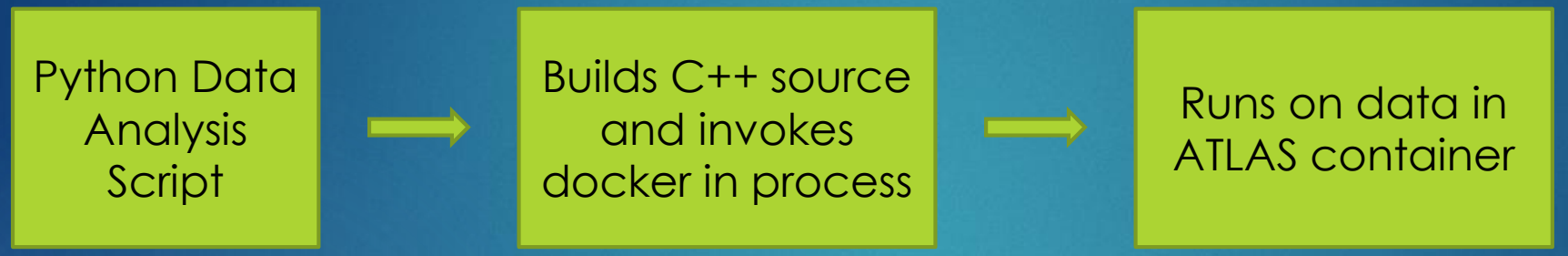
(python source)

Back End Flights Of Fancy (xAOD)

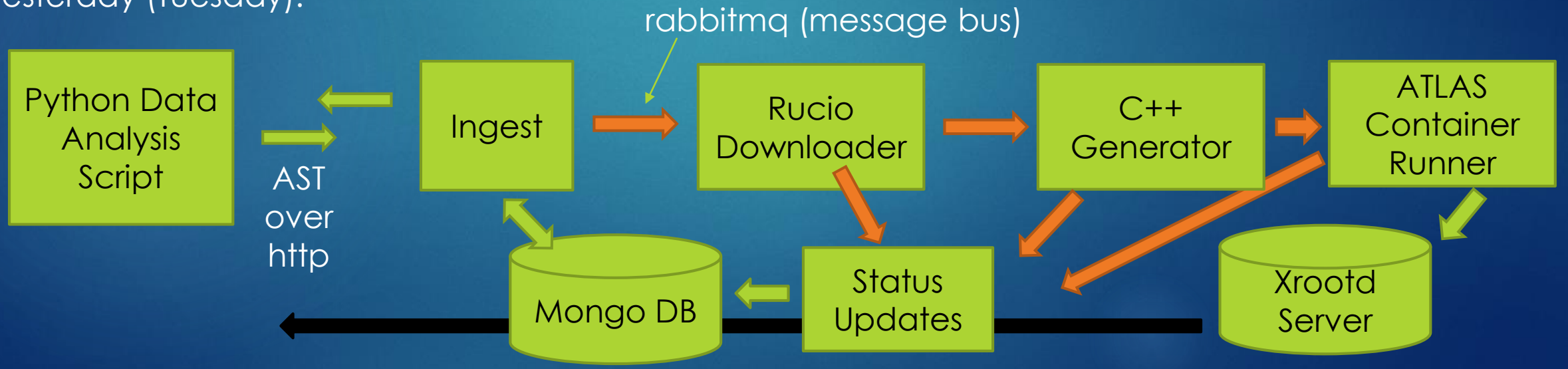
G. Watts (UW/Seattle)

Last Friday:

Monolithic



Yesterday (Tuesday):



An Aside…

- Converting monolithic approach to fully containerized: 1.5 days 
- Converting this to Kubernetes and Helm: 2 days (and not quite done yet) 

Helm Chart (containers are not published yet)

(keep in mind I have no idea what I'm doing)
Run's on my Windows laptop!

Repos

BDTTrainingAnalysisWork

- Has most of the recent work for RDF and Columnar backends
- Has lots of examples in Jupyter Notebooks
- Repo is for experimenting, not well laid out.

functional_adl

- Most recent work for the xAOD backend
- 90% copied and re-organized from the BDT.
- Layout still isn't pythonic (we are learning)
- Has no examples yet

calratio-perjet-training

- Uses functional_adl to build the training
- Doesn't use the container back-end yet

Repos

desktop_rucio	• Container that will cache files • Built to run on a laptop (e.g. loss of connectivity, shut downs by user due to power, etc). • Does require valid cert.
func_adl_cpp_runner	• Support for containerized ATLAS runner
func_adl_request_broker	• Support for containerized ATLAS runner
func_adl_server	• Helm chart to run everything in a k8 cluster • Will need some config eventually (e.g. where to get certs, where to write downloaded large files, etc.)

Risk: The Eco System

Very large number of tools that work with numpy

- How many can we keep with awkward?
- What about distributing the calculation across machines?

Individual tracks, electrons, jets, calorimeter clusters, etc. Unfiltered.

Jets, with near by tracks, and near by MC particle information. Very filtered.

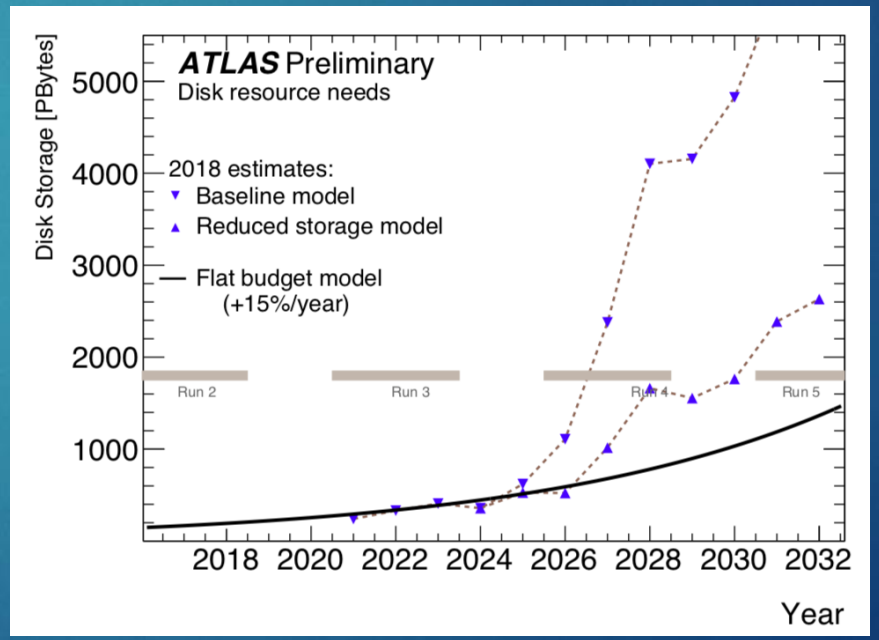
Histograms of jet p_T with $p_T > 50$ GeV, and 2 tracks near by with $\Delta R < 0.4$, etc.

An Aside

Disk space for the caches in these systems is making me very nervous!

There is no need to be nervous!

We have real numbers!



- ATLAS has a complete calculation that goes into these plots.
- Can extract sizes of xAOD's, etc.
- That plus local tests…

(my selection cuts turn 6 GB of data into 25 MB)

Conclusions

- Risk Factors
  - Not rewriting the eco system
  - How does this work fit in with ServiceX?
  - Cleaning up the Syntax
  - How does all of our lower-level work fit in with new analysis tools
- What is next
  - Stabilize the xAOD backend as a distributed tool
  - Bring the other two backends up to the same place
  - The challenges
  - Try to remove strings from front-end language
  - Evaluation