Data analysis tools from within HEP and from industry

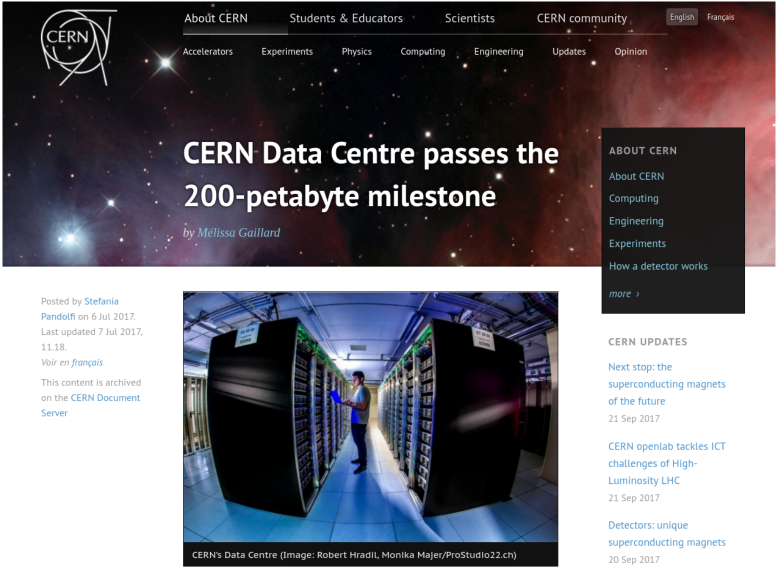
Jim Pivarski

Princeton University – DIANA-HEP

July 12, 2018

I'm going to start with a dumb comparison, to make a point.

We measure globally distributed data in hundreds of PB



CERN

About CERN | Students & Educators | Scientists | CERN community | English | Français

Accelerators | Experiments | Physics | Computing | Engineering | Updates | Opinion

CERN Data Centre passes the 200-petabyte milestone

by Mélissa Gaillard

Posted by Stefania Pandolfi on 6 Jul 2017. Last updated 7 Jul 2017, 11.18.

Voir en français

This content is archived on the CERN Document Server

CERN's Data Centre (Image: Robert Hradil, Monika Majer/ProStudio22.ch)

ABOUT CERN

- About CERN
- Computing
- Engineering
- Experiments
- How a detector works

more ›

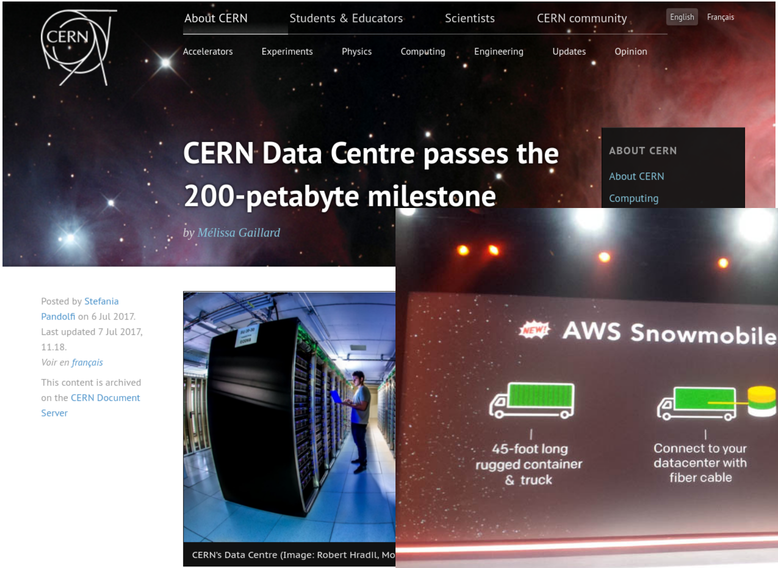
CERN UPDATES

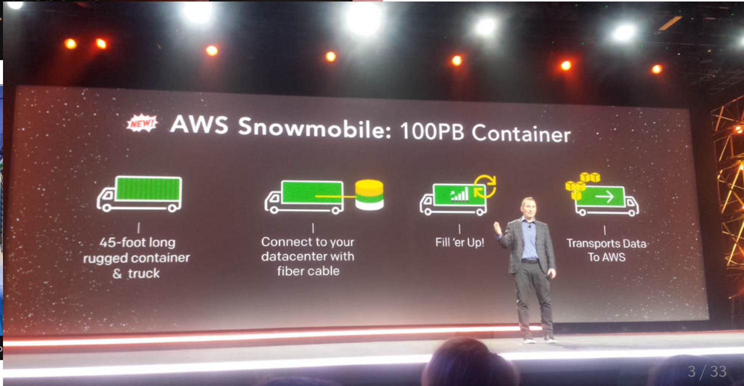
Next stop: the superconducting magnets of the future
21 Sep 2017

CERN openlab tackles ICT challenges of High-Luminosity LHC
21 Sep 2017

Detectors: unique superconducting magnets
20 Sep 2017

But for “web scale” companies, 100 PB = 1 truck





The point?

HEP is no longer the main developer or user in this problem space.

The point?

HEP is no longer the main developer or user in this problem space.

A better metric, which unfortunately I can't quantify:

- x FTEs in HEP developing open source analysis tools
- y FTEs outside of HEP developing open source analysis tools (not sure of x and y, but $x \ll y$)

The point?

HEP is no longer the main developer or user in this problem space.

A better metric, which unfortunately I can't quantify:

- x FTEs in HEP developing open source analysis tools
- y FTEs outside of HEP developing open source analysis tools (not sure of x and y, but $x \ll y$)

$\rightarrow$ There's a lot of good data analysis software out there!

The point?

HEP is no longer the main developer or user in this problem space.

A better metric, which unfortunately I can't quantify:

- x FTEs in HEP developing open source analysis tools
- y FTEs outside of HEP developing open source analysis tools (not sure of x and y, but $x \ll y$)

$\rightarrow$ There's a lot of good data analysis software out there!

$\rightarrow$ Could adopting it reduce in-house maintenance burdens?

The point?

HEP is no longer the main developer or user in this problem space.

A better metric, which unfortunately I can't quantify:

- x FTEs in HEP developing open source analysis tools
- y FTEs outside of HEP developing open source analysis tools (not sure of x and y, but $x \ll y$)

→ There's a lot of good data analysis software out there!
  → Could adopting it reduce in-house maintenance burdens?
    → More training examples and career options for users?

Show of hands: are you currently using data analysis software created outside of HEP?

Show of hands: are you currently using data analysis software created outside of HEP?

Are you planning to or want to?

On the other hand. . .

Another important metric: experience!

- High-energy physicists have been performing big data analytics (i.e. reducing large datasets to statistical inferences with computers) for about 50 years.

Another important metric: experience!

- High-energy physicists have been performing big data analytics (i.e. reducing large datasets to statistical inferences with computers) for about 50 years.
- Web-scale companies have been doing it for about 10 years.

Another important metric: experience!

- High-energy physicists have been performing big data analytics (i.e. reducing large datasets to statistical inferences with computers) for about 50 years.
- Web-scale companies have been doing it for about 10 years.

HEP analyses have grown sophisticated— there are certain things we expect but don't find in industry-grade software.

Another important metric: experience!

- High-energy physicists have been performing big data analytics (i.e. reducing large datasets to statistical inferences with computers) for about 50 years.
- Web-scale companies have been doing it for about 10 years.

HEP analyses have grown sophisticated— there are certain things we expect but don't find in industry-grade software.

The simple prescription of "just use Spark" would leave analyzers without some necessary tools.

Option #1

All of our needs are specialized.

Continue developing our own everything.

Option #1

All of our needs are specialized.

Continue developing our own everything.

Option #2

Modern big data software has some good ideas; integrate those *ideas* into our stack.

Option #1

All of our needs are specialized.

Continue developing our own everything.

Option #2

Modern big data software has some good ideas; integrate those *ideas* into our stack.

Option #3

Narrow our scope to HEP-specific tools, what no one else is developing, and make them interoperate with non-HEP tools for the common parts.

Option #1

All of our needs are specialized.

Continue developing our own everything.

Option #2

Modern big data software has some good ideas; integrate those *ideas* into our stack.

Option #3

Narrow our scope to HEP-specific tools, what no one else is developing, and make them interoperate with non-HEP tools for the common parts.

Option #4

Convince the world to start using HEP analysis techniques so that they will develop solutions for these, too.

Option #1

All of our needs are specialized.

Continue developing our own everything.

Option #2

Modern big data software has some good ideas; integrate those *ideas* into our stack.

Option #3

Narrow our scope to HEP-specific tools, what no one else is developing, and make them interoperate with non-HEP tools for the common parts.

Option #4

Convince the world to start using HEP analysis techniques so that they will develop solutions for these, too.

#3 is my opinion, but it begs the question: what's HEP-specific and what's not?

Three examples each:

What they've got

1. Distributed DAG processing
2. Indexed analysis
3. Machine learning

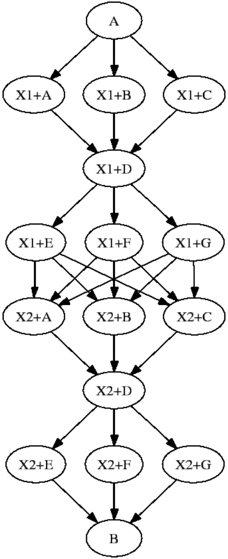
What we'd need

1. Nested data structures
2. Advanced histogramming
3. Ansatz fitting

Distributed DAG processing

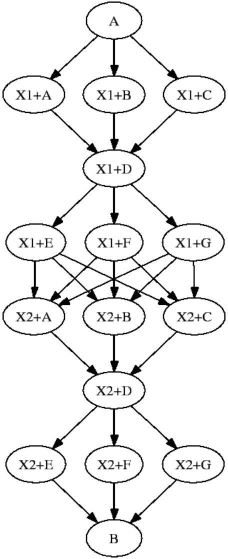
not HEP-specific

Distributed DAG processing



DAG: Directed Acyclic Graph of dependencies between subtasks.
Some would say this is what big data processing *is*.

Distributed DAG processing

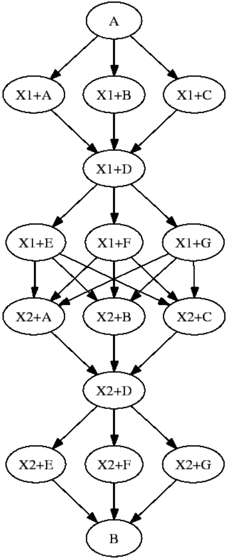


DAG: Directed Acyclic Graph of dependencies between subtasks.
Some would say this is what big data processing *is*.

Many frameworks distribute work this way:

Spark (JVM), Dask, Joblib, Parsl (Python), Storm (continuous), Thrill (C++), DAGMan (HTCondor), TensorFlow (fitting)...

Distributed DAG processing



DAG: Directed Acyclic Graph of dependencies between subtasks. Some would say this is what big data processing *is*.

Many frameworks distribute work this way:

Spark (JVM), Dask, Joblib, Parsl (Python), Storm (continuous), Thrill (C++), DAGMan (HTCondor), TensorFlow (fitting)...

To use these frameworks, one must

- express user tasks as DAG nodes (e.g. ROOT RDataFrame);
- serialize user functions on the driver and load user data on the workers in accordance to the framework's way of doing things.

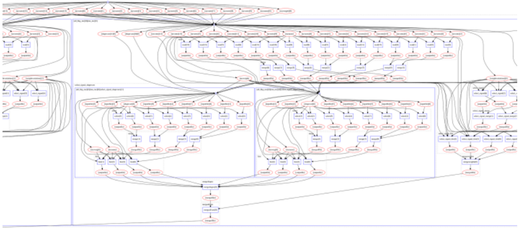
From "REANA: A System for Reusable Research Data Analyses"

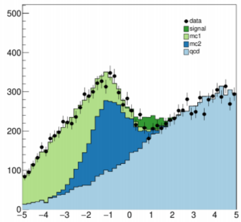
reana

Reproducible research data analysis platform

http://www.reana.io/

Example: BSM search

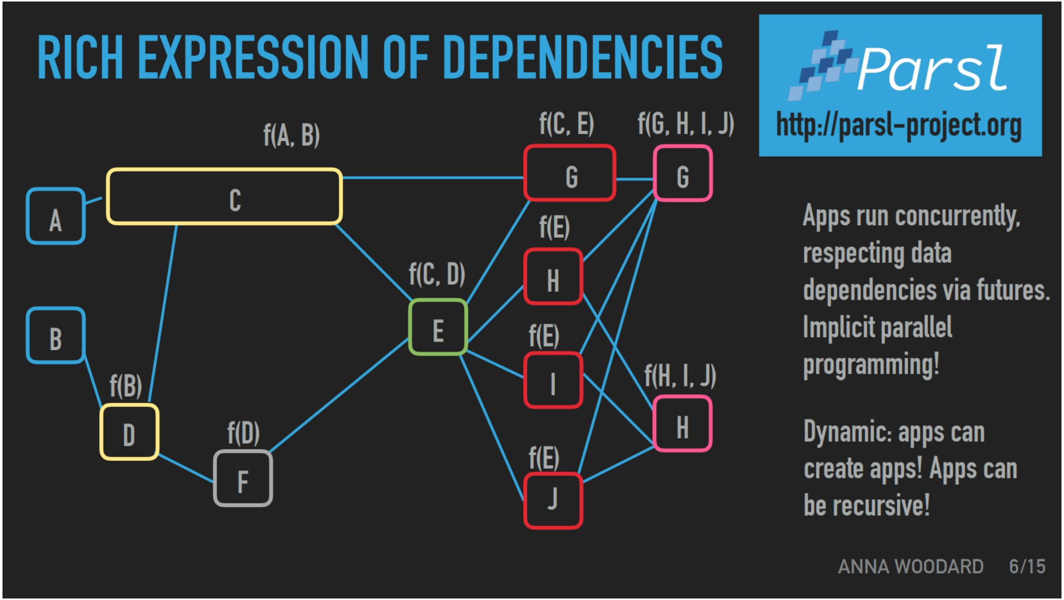




https://github.com/reanahub/reana-demo-bsm-search/

Complex computational workflows typical in particle physics analyses.

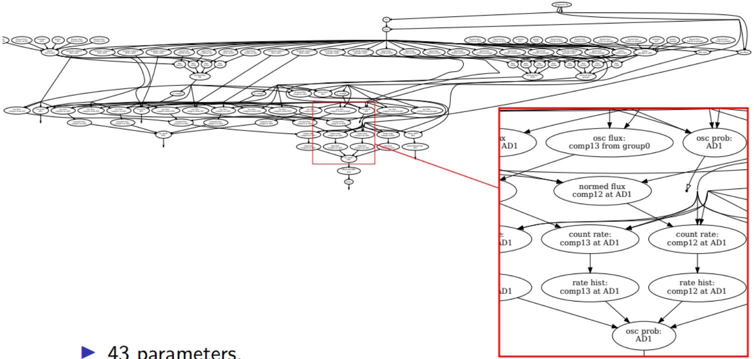
From “Interactive, scalable, reproducible data analysis with con. . .



From "GNA: new framework for statistical data analysis"

Computational graph example

The whole JUNO graph



- 43 parameters.
- The JUNO graph contains 110 nodes and 174 edges.
- It produces a histogram of 280 bins.

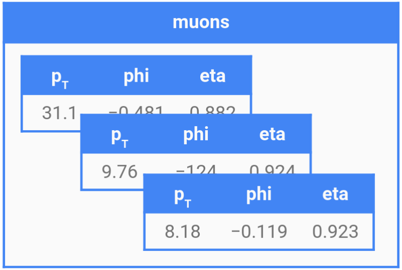
HEP has adopted the idea of DAGs, but will we be developing our own DAG-processors or using what we find?

(*Can* we? Why or why not?)

Nested data structures

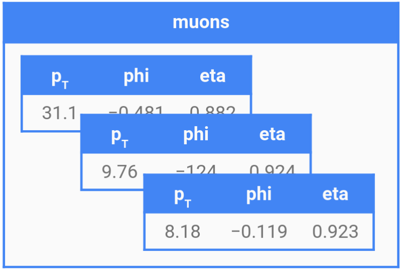
strangely HEP-specific

Nested data structures



mu1 p_T	mu1 phi	mu1 eta	mu2 p_T	mu2 phi	mu2 eta
31.1	−0.481	0.882	9.76	−0.124	0.924
5.27	1.246	−0.991	n/a	n/a	n/a
4.72	−0.207	0.953	n/a	n/a	n/a
8.59	−1.754	−0.264	8.714	0.185	0.629

Nested data structures

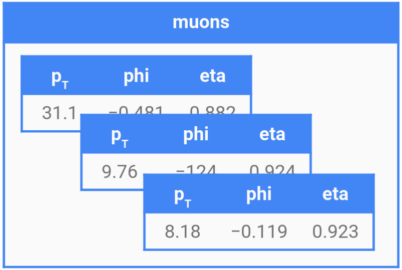


mu1 p_T	mu1 phi	mu1 eta	mu2 p_T	mu2 phi	mu2 eta
31.1	−0.481	0.882	9.76	−0.124	0.924
5.27	1.246	−0.991	n/a	n/a	n/a
4.72	−0.207	0.953	n/a	n/a	n/a
8.59	−1.754	−0.264	8.714	0.185	0.629

Objects are essential in HEP analysis.

Many physicists consider TTrees with `std::vector<float>` branches to be “minimal” or “flat.”

Nested data structures



mu1 p_T	mu1 phi	mu1 eta	mu2 p_T	mu2 phi	mu2 eta
31.1	−0.481	0.882	9.76	−0.124	0.924
5.27	1.246	−0.991	n/a	n/a	n/a
4.72	−0.207	0.953	n/a	n/a	n/a
8.59	−1.754	−0.264	8.714	0.185	0.629

Objects are essential in HEP analysis.

Many physicists consider TTrees with `std::vector<float>` branches to be “minimal” or “flat.”

Most data analysis tools have an SQL mindset, with rectangular data tables.

Objects $\to$ rectangular tables is lossy!

Performance claims often start the stopwatch after this “data cleaning.”

Nested data structures

Spark/Parquet/Arrow/HDF5/Pandas
has nested objects!

Nested data structures

Spark/Parquet/Arrow/HDF5/Pandas has nested objects!

Nested data are in these projects' scope, but as a second-class citizen.

Nested data structures

Spark/Parquet/Arrow/HDF5/Pandas has nested objects!

Nested data are in these projects' scope, but as a second-class citizen.

- Spark DataFrames allow arrays of structs, but using them involves a cumbersome explode-groupby or "drop to RDDs," giving up performance.

Nested data structures

Spark/Parquet/Arrow/HDF5/Pandas has nested objects!

Nested data are in these projects' scope, but as a second-class citizen.

- Spark DataFrames allow arrays of structs, but using them involves a cumbersome explode-groupby or "drop to RDDs," giving up performance.
- Parquet and Arrow specifications define lists of records, but they haven't been implemented in C++ and therefore Python yet (last time I checked).

Nested data structures

Spark/Parquet/Arrow/HDF5/Pandas has nested objects!

Nested data are in these projects' scope, but as a second-class citizen.

- Spark DataFrames allow arrays of structs, but using them involves a cumbersome explode-groupby or "drop to RDDs," giving up performance.
- Parquet and Arrow specifications define lists of records, but they haven't been implemented in C++ and therefore Python yet (last time I checked).
- HDF5 has lists of compounds, but they're rowwise ("unsplit").

Nested data structures

Spark/Parquet/Arrow/HDF5/Pandas has nested objects!

Nested data are in these projects' scope, but as a second-class citizen.

- Spark DataFrames allow arrays of structs, but using them involves a cumbersome explode-groupby or "drop to RDDs," giving up performance.
- Parquet and Arrow specifications define lists of records, but they haven't been implemented in C++ and therefore Python yet (last time I checked).
- HDF5 has lists of compounds, but they're rowwise ("unsplit").
- Pandas can put arbitrary Python objects in DataFrames, but most operations only apply to numbers.

Nested data structures

```
>>> import uproot
>>> t = uproot.open("tests/samples/HZZ.root")["events"]
>>> t.pandas.df(["MET_px", "Muon_Px", "Electron_Px"], entrystart=-20, flatten=False)

           MET_px                 Muon_Px             Electron_Px
2401     2.998099             [-1.492689]                      []
2402    27.944883             [-4.560287]                      []
2403     3.787466             [-9.715589]                      []
2404     9.378232            [-31.072098]                      []
2405   -17.310106  [47.484627, 4.6953125]                      []
2406   -81.965927  [74.75617, -20.911081]                      []
2407    -9.059591 [25.786427, -29.265024]                      []
2408    25.649775                      []                      []
2409    29.691553             [-24.7368]                      []
2410   -25.754967 [53.005814, -30.208649] [-37.681973, 18.453588]
2411    -2.426847   [55.7203, -26.914448]                      []
2412   -15.611773             [14.896802]                      []
2413    18.921183            [-24.158083]                      []
2414   -11.730723             [-9.204197]                      []
2415   -10.648725  [34.506527, -31.56778]                      []
2416   -14.607650            [-39.285824]                      []
2417    22.208313             [35.067146]                      []
2418    18.101646            [-29.756786]                      []
2419    79.875191             [1.1418698]                      []
2420    19.713749             [23.913206]                      []
```

In some cases, maybe we're using the wrong idiom: instead of working with structured values, Pandas prefers structured indexes.

Nested data structures

```
>>> import uproot
>>> t = uproot.open("tests/samples/HZZ.root")["events"]
>>> t.pandas.df(["MET_px", "Muon_Px", "Electron_Px"], entrystart=-20, flatten=True)

                   MET_px    Muon_Px  Electron_Px
entry subentry
2401  0          2.998099  -1.492689          NaN
2402  0         27.944883  -4.560287          NaN
2403  0          3.787466  -9.715589          NaN
2404  0          9.378232 -31.072098          NaN
2405  0        -17.310106  47.484627          NaN
      1               NaN   4.695312          NaN
2406  0        -81.965927  74.756172          NaN
      1               NaN -20.911081          NaN
2407  0         -9.059591  25.786427          NaN
      1               NaN -29.265024          NaN
2408  0         25.649775        NaN          NaN
2409  0         29.691553 -24.736799          NaN
2410  0        -25.754967  53.005814   -37.681973
      1               NaN -30.208649    18.453588
2411  0         -2.426847  55.720299          NaN
      1               NaN -26.914448          NaN
2412  0        -15.611773  14.896802          NaN
2413  0         18.921183 -24.158083          NaN
2414  0        -11.730723  -9.204197          NaN
2415  0        -10.648725  34.506527          NaN
      1               NaN -31.567780          NaN
2416  0        -14.607650 -39.285824          NaN
2417  0         22.208313  35.067146          NaN
2418  0         18.101646 -29.756786          NaN
2419  0         79.875191   1.141870          NaN
2420  0         19.713749  23.913206          NaN
```

In some cases, maybe we're using the wrong idiom: instead of working with structured values, Pandas prefers structured indexes.

Nested data structures

But that shouldn't be the only way: we *should* be able to use our data models and algorithms, even if we run them in non-HEP frameworks.

Nested data structures

But that shouldn't be the only way: we *should* be able to use our data models and algorithms, even if we run them in non-HEP frameworks.

This is my main project now: fast manipulation of columnar data.

General programming model

```
@numba.jit      # LLVM-compiled Python
def deltaphi(event):
    metphi = event.MET.phi
    for jet in event.jets:
        yield metphi - jet.phi
```

Numpy-like broadcasting

Awkward Array

```
# one per event          one per particle
event["MET"]["phi"] - event["jet"]["phi"]
```

Nested data structures

But that shouldn't be the only way: we *should* be able to use our data models and algorithms, even if we run them in non-HEP frameworks.

This is my main project now: fast manipulation of columnar data.

<u>General programming model</u>

```
@numba.jit     # LLVM-compiled Python
def deltaphi(event):
    metphi = event.MET.phi
    for jet in event.jets:
        yield metphi - jet.phi
```

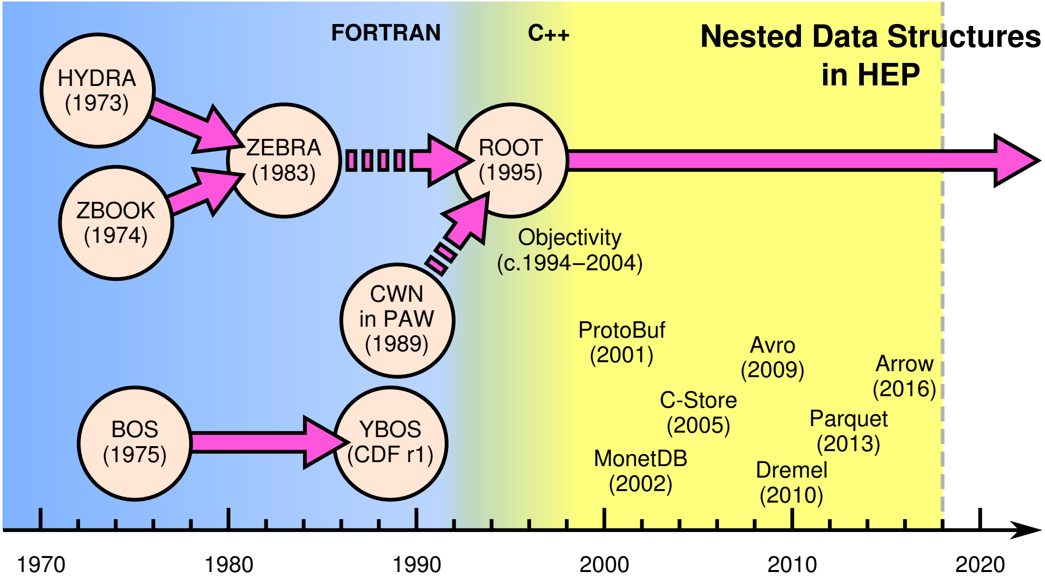
<u>Numpy-like broadcasting</u>

Awkward Array

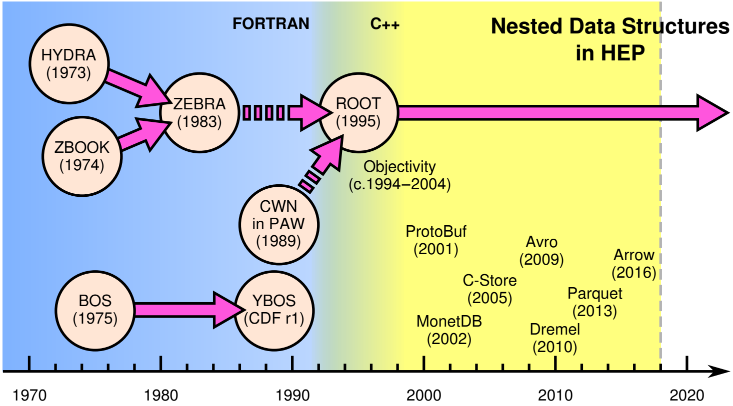
```
# one per event           one per particle
event["MET"]["phi"] - event["jet"]["phi"]
```

Also, this *should* be of wider interest than HEP: developers of Arrow, Dask, and XND (∼Numpy 2.0) are curious about it.

Missed opportunity



Missed opportunity



Google Dremel paper (2010):
(inspired Parquet)

storage and reduce CPU cost due to cheaper compression. Column stores have been adopted for analyzing relational data [1] but to the best of our knowledge have not been extended to nested data models. The columnar storage format that we present is supported by

Indexed analysis

not well-known in HEP

To understand what I mean by "indexed analysis," consider analysis with *less advanced indexing* than modern HEP.

Indexed analysis

```
h/cr/1d 201 'd0miss' 100 -0.5e-3 0.5e-3
h/cr/1d 202 'z0miss' 100 -0.015 0.015
h/cr/1d 203 'pxmiss' 100 -0.076 0.076
h/cr/1d 204 'pymiss' 100 -0.076 0.076
h/cr/1d 205 'pzmiss' 100 -0.076 0.076
nt/plot 2.d0 ! ! ! ! ! 201
nt/plot 2.z0 ! ! ! ! ! 202
nt/plot 2.px ! ! ! ! ! 203
nt/plot 2.py ! ! ! ! ! 204
nt/plot 2.pz ! ! ! ! ! 205

h/cr/1d 301 'normalized d0miss' 100 -10 10
h/cr/1d 302 'normalized z0miss' 100 -10 10
h/cr/1d 303 'normalized pxmiss' 100 -10 10
h/cr/1d 304 'normalized pymiss' 100 -10 10
h/cr/1d 305 'normalized pzmiss' 100 -10 10
nt/plot 2.d0/sqrt(ed0) ! ! ! ! ! 301
nt/plot 2.z0/sqrt(ez0) ! ! ! ! ! 302
nt/plot 2.px/sqrt(epx) ! ! ! ! ! 303
nt/plot 2.py/sqrt(epy) ! ! ! ! ! 304
nt/plot 2.pz/sqrt(epz) ! ! ! ! ! 305

h/cr/1d 401 'd0miss after constraint' 100 -0.1e-16 0.1e-16
h/cr/1d 402 'z0miss after constraint' 100 -0.1e-15 0.1e-15
h/cr/1d 403 'pxmiss after constraint' 100 -0.01 0.01
h/cr/1d 404 'pymiss after constraint' 100 -0.01 0.01
h/cr/1d 405 'pzmiss after constraint' 100 -0.01 0.01
nt/plot 2.ad0 ! ! ! ! ! 401
nt/plot 2.az0 ! ! ! ! ! 402
nt/plot 2.apx ! ! ! ! ! 403
nt/plot 2.apy ! ! ! ! ! 404
nt/plot 2.apz ! ! ! ! ! 405
```

To the left is a PAW script (pre-ROOT), creating and filling histograms.

Histograms were indexed by numbers because they didn't have *names* back then.

Indexed analysis

```
h/cr/1d 201 'd0miss' 100 -0.5e-3 0.5e-3
h/cr/1d 202 'z0miss' 100 -0.015 0.015
h/cr/1d 203 'pxmiss' 100 -0.076 0.076
h/cr/1d 204 'pymiss' 100 -0.076 0.076
h/cr/1d 205 'pzmiss' 100 -0.076 0.076
nt/plot 2.d0 ! ! ! ! ! 201
nt/plot 2.z0 ! ! ! ! ! 202
nt/plot 2.px ! ! ! ! ! 203
nt/plot 2.py ! ! ! ! ! 204
nt/plot 2.pz ! ! ! ! ! 205

h/cr/1d 301 'normalized d0miss' 100 -10 10
h/cr/1d 302 'normalized z0miss' 100 -10 10
h/cr/1d 303 'normalized pxmiss' 100 -10 10
h/cr/1d 304 'normalized pymiss' 100 -10 10
h/cr/1d 305 'normalized pzmiss' 100 -10 10
nt/plot 2.d0/sqrt(ed0) ! ! ! ! ! 301
nt/plot 2.z0/sqrt(ez0) ! ! ! ! ! 302
nt/plot 2.px/sqrt(epx) ! ! ! ! ! 303
nt/plot 2.py/sqrt(epy) ! ! ! ! ! 304
nt/plot 2.pz/sqrt(epz) ! ! ! ! ! 305

h/cr/1d 401 'd0miss after constraint' 100 -0.1e-16 0.1e-16
h/cr/1d 402 'z0miss after constraint' 100 -0.1e-15 0.1e-15
h/cr/1d 403 'pxmiss after constraint' 100 -0.01 0.01
h/cr/1d 404 'pymiss after constraint' 100 -0.01 0.01
h/cr/1d 405 'pzmiss after constraint' 100 -0.01 0.01
nt/plot 2.ad0 ! ! ! ! ! 401
nt/plot 2.az0 ! ! ! ! ! 402
nt/plot 2.apx ! ! ! ! ! 403
nt/plot 2.apy ! ! ! ! ! 404
nt/plot 2.apz ! ! ! ! ! 405
```

To the left is a PAW script (pre-ROOT), creating and filling histograms.

Histograms were indexed by numbers because they didn't have *names* back then.

The ability to name stuff was as fundamental to HEP data analysis as handwashing was to medical science!

Indexed analysis

```
h/cr/1d 201 'd0miss' 100 -0.5e-3 0.5e-3
h/cr/1d 202 'z0miss' 100 -0.015 0.015
h/cr/1d 203 'pxmiss' 100 -0.076 0.076
h/cr/1d 204 'pymiss' 100 -0.076 0.076
h/cr/1d 205 'pzmiss' 100 -0.076 0.076
nt/plot 2.d0 ! ! ! ! ! 201
nt/plot 2.z0 ! ! ! ! ! 202
nt/plot 2.px ! ! ! ! ! 203
nt/plot 2.py ! ! ! ! ! 204
nt/plot 2.pz ! ! ! ! ! 205

h/cr/1d 301 'normalized d0miss' 100 -10 10
h/cr/1d 302 'normalized z0miss' 100 -10 10
h/cr/1d 303 'normalized pxmiss' 100 -10 10
h/cr/1d 304 'normalized pymiss' 100 -10 10
h/cr/1d 305 'normalized pzmiss' 100 -10 10
nt/plot 2.d0/sqrt(ed0) ! ! ! ! ! 301
nt/plot 2.z0/sqrt(ez0) ! ! ! ! ! 302
nt/plot 2.px/sqrt(epx) ! ! ! ! ! 303
nt/plot 2.py/sqrt(epy) ! ! ! ! ! 304
nt/plot 2.pz/sqrt(epz) ! ! ! ! ! 305

h/cr/1d 401 'd0miss after constraint' 100 -0.1e-16 0.1e-16
h/cr/1d 402 'z0miss after constraint' 100 -0.1e-15 0.1e-15
h/cr/1d 403 'pxmiss after constraint' 100 -0.01 0.01
h/cr/1d 404 'pymiss after constraint' 100 -0.01 0.01
h/cr/1d 405 'pzmiss after constraint' 100 -0.01 0.01
nt/plot 2.ad0 ! ! ! ! ! 401
nt/plot 2.az0 ! ! ! ! ! 402
nt/plot 2.apx ! ! ! ! ! 403
nt/plot 2.apy ! ! ! ! ! 404
nt/plot 2.apz ! ! ! ! ! 405
```

To the left is a PAW script (pre-ROOT), creating and filling histograms.

Histograms were indexed by numbers because they didn't have *names* back then.

The ability to name stuff was as fundamental to HEP data analysis as handwashing was to medical science!

But we don't have to stop there. There's more to indexing than name-value pairs.

From “Pandas DataFrames for F.A.S.T. binned analysis at CMS”

Example Cut-flow Dataframe

component	depth	class	name	pass	total
data	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	16208	469384
		LambdaStr	ev: ev.triggerIsoMu24[0]	16208	16208
		LambdaStr	ev: ev.Muon_Pt[0] > 25	15995	16208
dy	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	37559	77729
		LambdaStr	ev: ev.triggerIsoMu24[0]	37559	37559
		LambdaStr	ev: ev.Muon_Pt[0] > 25	37263	37559
qcd	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	0	142
		LambdaStr	ev: ev.triggerIsoMu24[0]	0	0
		LambdaStr	ev: ev.Muon_Pt[0] > 25	0	0
single_top	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	111	5684
		LambdaStr	ev: ev.triggerIsoMu24[0]	111	111
		LambdaStr	ev: ev.Muon_Pt[0] > 25	110	111
ttbar	1	LambdaStr	ev: len(ev.		
		LambdaStr	ev:		
		LambdaStr	ev:		
wjets	1	LambdaStr	ev: len(ev.		
		LambdaStr	ev:		
		LambdaStr	ev:		
ww	1	LambdaStr	ev: len(ev.		
		LambdaStr	ev:		
		LambdaStr	ev:		
wz	1	LambdaStr	ev: len(ev.		
		LambdaStr	ev:		
		LambdaStr	ev:		
zz	1	LambdaStr	ev: len(ev.		
		LambdaStr	ev: ev.triggerIsoMu24[0]	1235	1235
		LambdaStr	ev: ev.Muon_Pt[0] > 25	1232	1235

```
# Apply an event selection
Selection:
    Selection:
        All:
            - len(ev.Muon_Iso_Idx) >= 2
            - ev.triggerIsoMu24[0]
            - ev.Muon_Pt[0] > 25
```


component	depth	class	name	pass	total
data	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	16208	469384
		LambdaStr	ev: ev.triggerIsoMu24[0]	16208	16208
		LambdaStr	ev: ev.Muon_Pt[0] > 25	15995	16208
dy	1	LambdaStr	ev: len(ev.Muon_Iso_Idx) >= 2	37559	77729
		LambdaStr	ev: ev.triggerIsoMu24[0]	37559	37559
		LambdaStr	ev: ev.Muon_Pt[0] > 25	37263	37559

Pandas DataFrames filled by AlphaTwirl, analyzed by F.A.S.T.

From "Pandas DataFrames for F.A.S.T. binned analysis at CMS"

Manipulating DFs: Long to wide form

```python
# Convert variance --> error
df["err"] = np.sqrt(df.nvar)

# Switch to long-form
df2 = df.pivot_table(index="dimu_mass", columns="component", values=["n", "err"])
df2 = df2.sort_index(axis=1, ascending=False)

# Sort components to match tutorial
order = ["data", "ttbar", "wjets", "dy", "ww", "wz", "zz", "qcd", "single_top"]
df2 = df2.reindex(order, axis=1, level="component")

# Show first 10 rows
df2.head(10)
```

	n								err			
component	**data**	**ttbar**	**wjets**	**dy**	**ww**	**wz**	**zz**	**single_top**	**data**	**ttbar**	**wjets**	**d**
dimu_mass												
-inf	993.0	11.392980	0.311917	655.570771	3.600221	0.320914	0.360053	1.741041	31.511903	1.752727	0.311917	3
60.000000	38.0	0.840432	0.000000	23.963227	0.063284	0.053328	0.000000	0.065288	6.164414	0.486302	0.000000	:
61.000000	25.0	0.319709	NaN	25.572841	0.102053	0.000000	NaN	0.005831	5.000000	0.275655	NaN	:
62.000000	22.0	0.274432	NaN	29.271624	0.068484	0.038697	NaN	0.000000	4.690416	0.274432	NaN	:
63.000000	28.0	0.000000	NaN	22.941727	0.194258	0.000000	0.009475	NaN	5.291503	0.000000	NaN	:
64.000000	29.0	0.847224	NaN	20.534599	0.065338	0.081642	0.009540	NaN	5.385165	0.490427	NaN	:
65.000000	17.0	0.352667	NaN	29.464412	0.130224	0.000000	0.004153	0.093700	4.123106	0.282423	NaN	:
66.000000	37.0	0.570011	NaN	27.861013	0.128668	0.059988	0.015375	0.000000	6.082763	0.403615	NaN	:
67.000000	34.0	0.817704	NaN	34.173523	0.063818	0.000000	0.017707	0.000652	5.830952	0.475827	NaN	.
68.000000	31.0	0.753107	NaN	26.971645	0.024008	0.042326	0.000000	0.000000	5.567764	0.440761	NaN	:

Depending on task, "wide-form" tables can be easier to work with

Pandas DataFrames filled by AlphaTwirl, analyzed by F.A.S.T.

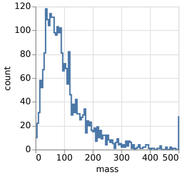
I had the same thought: our primary examples of indexable data are histograms and systems of related histograms. Rich indexing would let us project/rebin/cut/transform histograms more fluidly.

```
>>> from histbook import *
>>> multihist = Hist(bin("mass", 100, 0, 500), cut("q1*q2 < 0"),
...                  split("mt1", [0.2, 0.5]), split("mt2", [0.2, 0.5]), fill=df)
>>> multihist.pandas()
                                                   count()  err(count())
mass           q1*q2 < 0 mt1         mt2
[-inf, 0.0)    fail      [-inf, 0.2) [-inf, 0.2)       0.0           0.0
                                     [0.2, 0.5)        0.0           0.0
                                     [0.5, inf)        0.0           0.0
                         [0.2, 0.5)  [-inf, 0.2)       0.0           0.0
                                     [0.2, 0.5)        0.0           0.0
                                     [0.5, inf)        0.0           0.0
                         [0.5, inf)  [-inf, 0.2)       0.0           0.0
                                     [0.2, 0.5)        0.0           0.0
                                     [0.5, inf)        0.0           0.0
               pass      [-inf, 0.2) [-inf, 0.2)       0.0           0.0
                                     [0.2, 0.5)        0.0           0.0
                                     [0.5, inf)        0.0           0.0
                         [0.2, 0.5)  [-inf, 0.2)       0.0           0.0
                                     [0.2, 0.5)        0.0           0.0
                                     [0.5, inf)        0.0           0.0
                         [0.5, inf)  [-inf, 0.2)       0.0           0.0
                                     [0.2, 0.5)        0.0           0.0
```

I had the same thought: our primary examples of indexable data are histograms and systems of related histograms. Rich indexing would let us project/rebin/cut/transform histograms more fluidly.

histbook

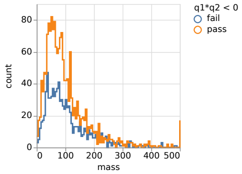
```
>>> from histbook import *
>>> multihist = Hist(bin("mass", 100, 0, 500), cut("q1*q2 < 0"),
...                  split("mt1", [0.2, 0.5]), split("mt2", [0.2, 0.5]), fill=df)
>>> multihist.step("mass")
```



Indexed analysis

I had the same thought: our primary examples of indexable data are histograms and systems of related histograms. Rich indexing would let us project/rebin/cut/transform histograms more fluidly.

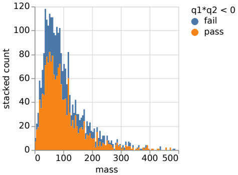
```
>>> from histbook import *
>>> multihist = Hist(bin("mass", 100, 0, 500), cut("q1*q2 < 0"),
...                  split("mt1", [0.2, 0.5]), split("mt2", [0.2, 0.5]), fill=df)
>>> multihist.overlay("q1*q2 < 0").step("mass")
```



I had the same thought: our primary examples of indexable data are histograms and systems of related histograms. Rich indexing would let us project/rebin/cut/transform histograms more fluidly.

histbook

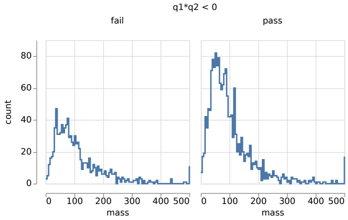
```
>>> from histbook import *
>>> multihist = Hist(bin("mass", 100, 0, 500), cut("q1*q2 < 0"),
...                  split("mt1", [0.2, 0.5]), split("mt2", [0.2, 0.5]), fill=df)
>>> multihist.stack("q1*q2 < 0").area("mass")
```



Indexed analysis

I had the same thought: our primary examples of indexable data are histograms and systems of related histograms. Rich indexing would let us project/rebin/cut/transform histograms more fluidly.

```
>>> from histbook import *
>>> multihist = Hist(bin("mass", 100, 0, 500), cut("q1*q2 < 0"),
...                  split("mt1", [0.2, 0.5]), split("mt2", [0.2, 0.5]), fill=df)
>>> multihist.beside("q1*q2 < 0").step("mass")
```

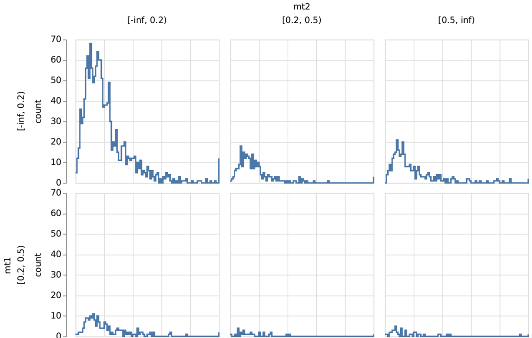


Indexed analysis

I had the same thought: our primary examples of indexable data are histograms and systems of related histograms. Rich indexing would let us project/rebin/cut/transform histograms more fluidly.

histbook

```
>>> from histbook import *
>>> multihist = Hist(bin("mass", 100, 0, 500), cut("q1*q2 < 0"),
...                  split("mt1", [0.2, 0.5]), split("mt2", [0.2, 0.5]), fill=df)
>>> multihist.below("mt1").beside("mt2").step("mass")
```

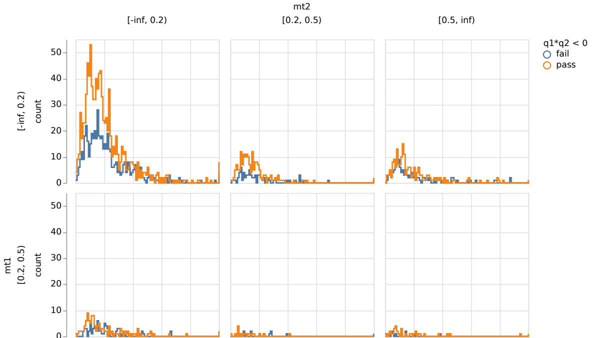


Indexed analysis

I had the same thought: our primary examples of indexable data are histograms and systems of related histograms. Rich indexing would let us project/rebin/cut/transform histograms more fluidly.

histbook

```
>>> from histbook import *
>>> multihist = Hist(bin("mass", 100, 0, 500), cut("q1*q2 < 0"),
...                  split("mt1", [0.2, 0.5]), split("mt2", [0.2, 0.5]), fill=df)
>>> multihist.below("mt1").beside("mt2").overlay("q1*q2 < 0").step("mass")
```



Indexed analysis

Using tools with rich indexing systemizes what we're already doing with naming conventions, splitting names on underscores, etc.

Pandas is not a TTree replacement— if anything, it's a histogram organizer!

Advanced histogramming

very HEP-specific

Advanced histogramming

The histograms themselves, however, are more sophisticated in HEP than elsewhere.

Advanced histogramming

The histograms themselves, however, are more sophisticated in HEP than elsewhere.

- As far as I have found, *only* HEP histogramming tools (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled, merged, and accessed programmatically.

Advanced histogramming

The histograms themselves, however, are more sophisticated in HEP than elsewhere.

- As far as I have found, *only* HEP histogramming tools (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled, merged, and accessed programmatically.
- In many non-HEP packages, "histogram" is more of a display option than an analysis tool, with no way to access contents or control binning.

Advanced histogramming

The histograms themselves, however, are more sophisticated in HEP than elsewhere.

- As far as I have found, *only* HEP histogramming tools (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled, merged, and accessed programmatically.
- In many non-HEP packages, “histogram” is more of a display option than an analysis tool, with no way to access contents or control binning.
- “Profile” plots are only in HEP. Robust log scales are hard to find, too.

The histograms themselves, however, are more sophisticated in HEP than elsewhere.

- As far as I have found, *only* HEP histogramming tools (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled, merged, and accessed programmatically.
- In many non-HEP packages, "histogram" is more of a display option than an analysis tool, with no way to access contents or control binning.
- "Profile" plots are only in HEP. Robust log scales are hard to find, too.

These features aren't difficult, but they're our responsibility.

Machine learning versus ansatz fitting

Machine learning versus ansatz fitting

My take on machine learning: **it's fitting.**

My take on machine learning: **it's fitting.**

It's fitting with thousands of free parameters, where the goal is not to find a global minimum or understand the limiting value of those parameters, but to generate, recognize, or classify patterns.

My take on machine learning: **it's fitting.**

It's fitting with thousands of free parameters, where the goal is not to find a global minimum or understand the limiting value of those parameters, but to generate, recognize, or classify patterns.

Ansatz fitting, however, optimizes a theory-driven function of few parameters, and the exact shape of the minimum has implications for the theory.

Machine learning versus ansatz fitting

My take on machine learning: **it's fitting.**

It's fitting with thousands of free parameters, where the goal is not to find a global minimum or understand the limiting value of those parameters, but to generate, recognize, or classify patterns.

Ansatz fitting, however, optimizes a theory-driven function of few parameters, and the exact shape of the minimum has implications for the theory.

Qualitatively different purposes: **both needed.**

My take on machine learning: **it's fitting.**

It's fitting with thousands of free parameters, where the goal is not to find a global minimum or understand the limiting value of those parameters, but to generate, recognize, or classify patterns.

Ansatz fitting, however, optimizes a theory-driven function of few parameters, and the exact shape of the minimum has implications for the theory.

Qualitatively different purposes: **both needed.**

We can look to industry for machine learning innovations, but the best ansatz fitters are in HEP: RooFit, RooStats, GooFit, HistFitter, HistFactory, pyhf...

Areas of overlap

What they've got

1. Distributed DAG processing
2. Indexed analysis
3. Machine learning

What we'd need

1. Nested data structures
2. Advanced histogramming
3. Ansatz fitting

Areas of overlap

What they've got

1. Distributed DAG processing
2. Indexed analysis
3. **Machine learning**

What we'd need

1. **Nested data structures**
2. Advanced histogramming
3. Ansatz fitting

Nearly all ML techniques require flattened or sequences of flattened data, but we have real problems that need nested data: e.g. classifying N_i jets per event (nested, unordered sets). RNNs and LSTMs (for non-nested, ordered sequences) are designed for a different data type!

Areas of overlap

<u>What they've got</u>

1. Distributed DAG processing
2. **Indexed analysis**
3. Machine learning

<u>What we'd need</u>

1. Nested data structures
2. **Advanced histogramming**
3. Ansatz fitting

F.A.S.T. and histbook are incorporating Pandas indexing into advanced histogramming.

Areas of overlap

What they've got

1. **Distributed DAG processing**
2. Indexed analysis
3. Machine learning

What we'd need

1. Nested data structures
2. Advanced histogramming
3. **Ansatz fitting**

As fits get bigger, they may need to be distributed, for instance with iterative map-reduce.

Conclusions

Data analysis tools outside of HEP are mature but not a perfect fit to our needs.

- Some of what we need is available now: can we use it?
- Some exists only as HEP software: can it interoperate?
- Some of what's available is unlike anything we do now: an opportunity to do better physics?

Conclusions

Data analysis tools outside of HEP are mature but not a perfect fit to our needs.

- Some of what we need is available now: can we use it?
- Some exists only as HEP software: can it interoperate?
- Some of what's available is unlike anything we do now: an opportunity to do better physics?
- The door swings both ways: we have things to teach the world!