Data analysis tools from within particle physics and from industry

Jim Pivarski

Princeton University – DIANA-HEP

September 18, 2018

The point I want to make

- Although nuclear and high energy physics once dealt with the world's largest datasets, this is no longer true. "Big Data" or (better) "Web Scale" analysis regularly deals with petabytes and exabytes, and they've developed software tools for it.

The point I want to make

- Although nuclear and high energy physics once dealt with the world's largest datasets, this is no longer true. "Big Data" or (better) "Web Scale" analysis regularly deals with petabytes and exabytes, and they've developed software tools for it.
- We can reduce maintenance costs and improve students' career options by mixing industry standard tools with our in-house tools, particularly for cases in which the purpose of the tool is the same.

The point I want to make

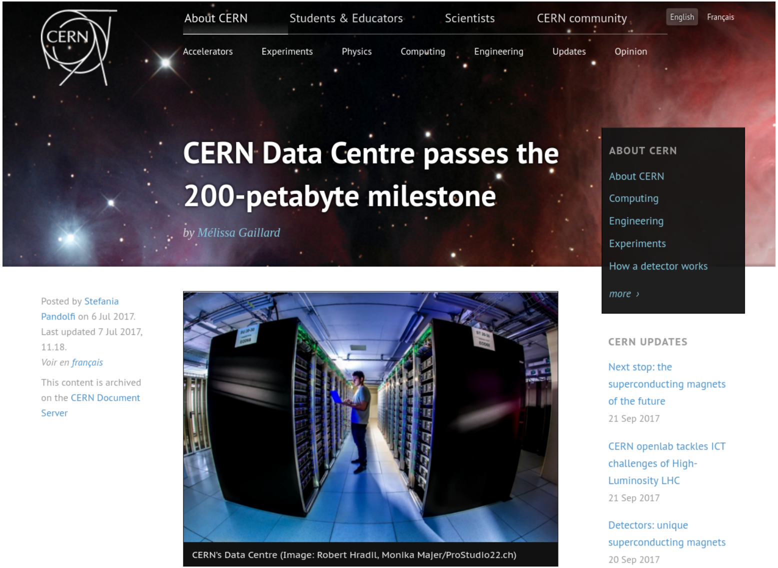
- Although nuclear and high energy physics once dealt with the world's largest datasets, this is no longer true. "Big Data" or (better) "Web Scale" analysis regularly deals with petabytes and exabytes, and they've developed software tools for it.
- We can reduce maintenance costs and improve students' career options by mixing industry standard tools with our in-house tools, particularly for cases in which the purpose of the tool is the same.
- This is in line with ROOT's new Python and TMVA interfaces, but broader: data should flow freely to the best tool for the job and back again, leaving the choice in the physicist's hands.

The point I want to make

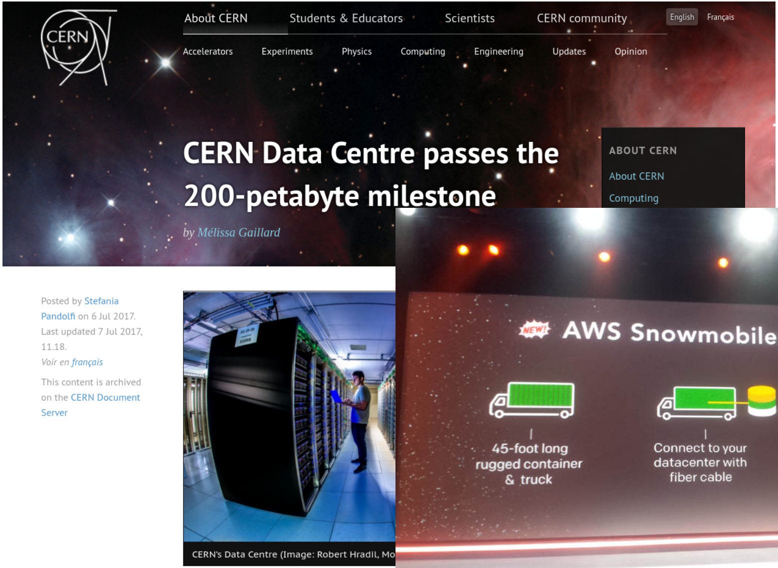
- Although nuclear and high energy physics once dealt with the world's largest datasets, this is no longer true. "Big Data" or (better) "Web Scale" analysis regularly deals with petabytes and exabytes, and they've developed software tools for it.
- We can reduce maintenance costs and improve students' career options by mixing industry standard tools with our in-house tools, particularly for cases in which the purpose of the tool is the same.
- This is in line with ROOT's new Python and TMVA interfaces, but broader: data should flow freely to the best tool for the job and back again, leaving the choice in the physicist's hands.

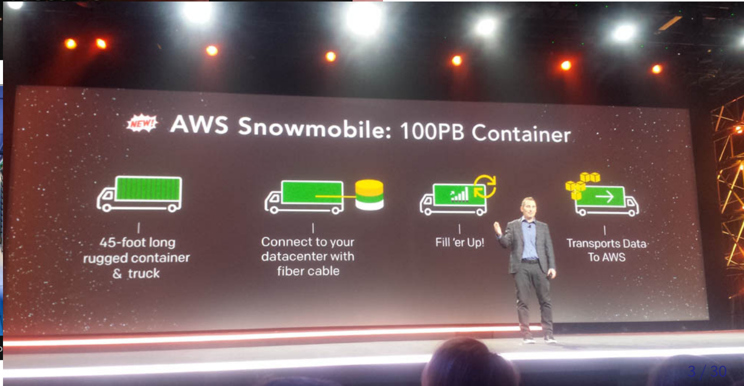
In short, we should become like other sciences, such as astronomy or biology: common libraries for common stuff and our own libraries for domain-specific stuff.

We measure globally distributed data in hundreds of PB

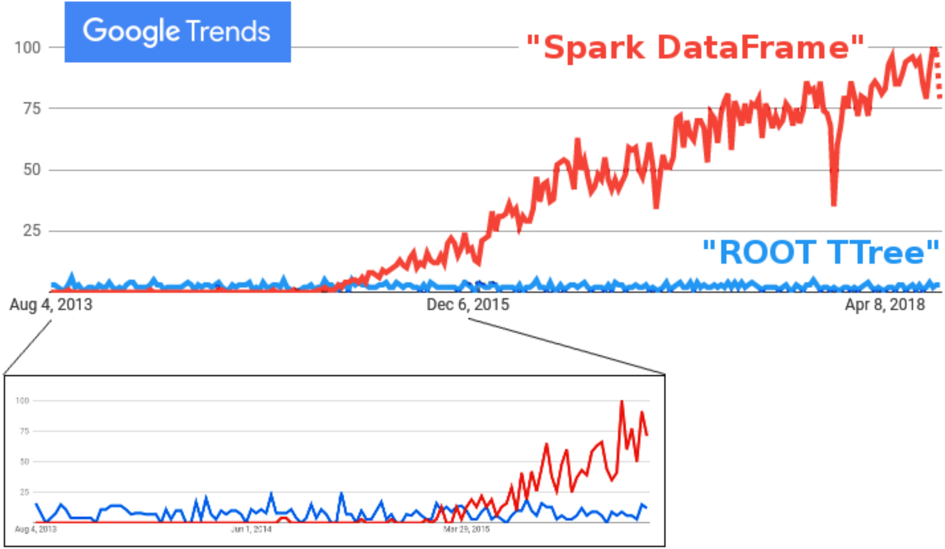


But for web scale companies, 100 PB = 1 truck

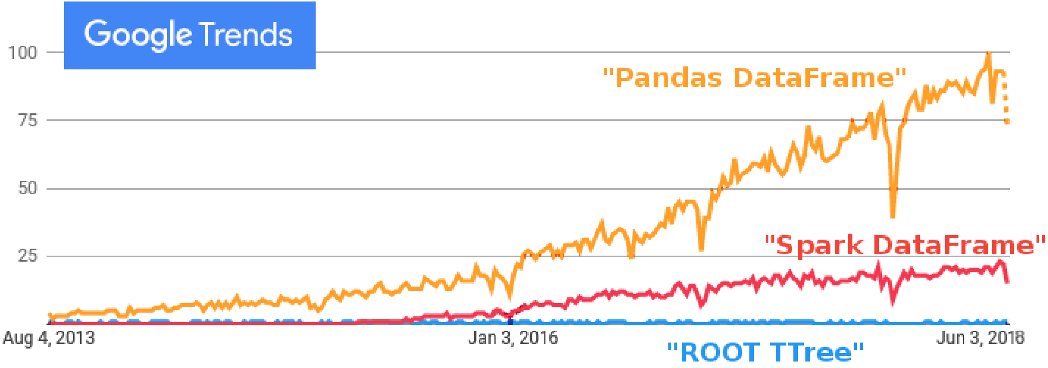




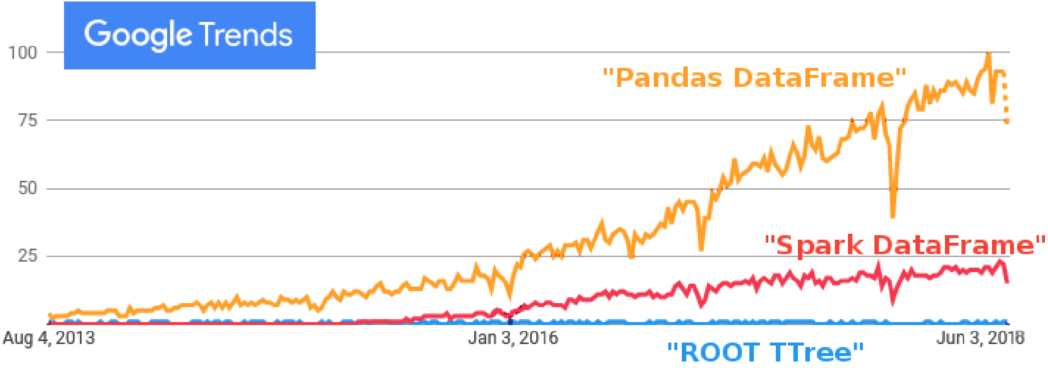
Number of people (users and developers) also dwarf our field



Number of people (users and developers) also dwarf our field



Number of people (users and developers) also dwarf our field



More users means more bug reports, more online help, more how-to blogs. . .

More developers means more bug-fixes, more features, more connectors. . .

Another important metric: experience!

- Physicists have been performing big data analytics (reducing large datasets to statistical inferences) for about 50 years.
- Web scale companies have been doing it for about 10 years.

Another important metric: experience!

- Physicists have been performing big data analytics (reducing large datasets to statistical inferences) for about 50 years.
- Web scale companies have been doing it for about 10 years.

Nuclear and high-energy physics analysis is specialized and sophisticated— many tools we'd call "basic" are not implemented in industry-grade software.

Another important metric: experience!

- Physicists have been performing big data analytics (reducing large datasets to statistical inferences) for about 50 years.
- Web scale companies have been doing it for about 10 years.

Nuclear and high-energy physics analysis is specialized and sophisticated— many tools we'd call "basic" are not implemented in industry-grade software.

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

What should we do?

Option #1

All of our needs are specialized.

Continue developing our own everything.

What should we do?

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.

What should we do?

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 domain-specific tools, what no one else is developing, and make them interoperate with non-physics tools for the common parts.

What should we do?

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 domain-specific tools, what no one else is developing, and make them interoperate with non-physics tools for the common parts.

Option #4

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

What should we do?

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 domain-specific tools, what no one else is developing, and make them interoperate with non-physics tools for the common parts.

Option #4

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

#3 is my opinion, but what's domain-specific and what's not?

Three examples each:

What web scale software's got

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

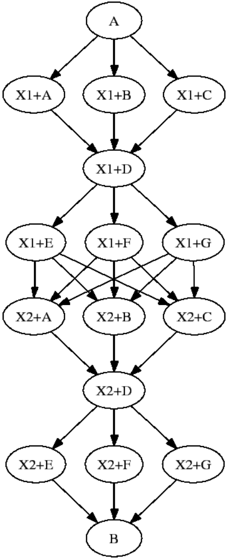
What we need that it hasn't got

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

Distributed DAG processing

not physics-specific

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:

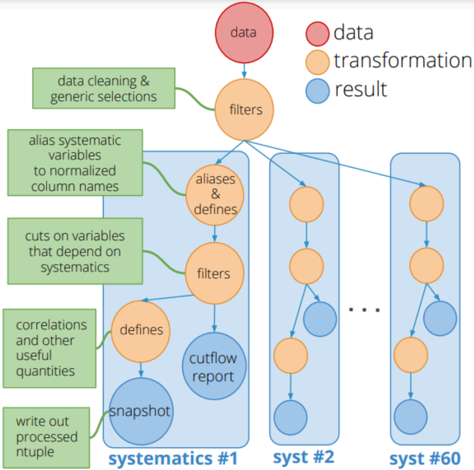
Spark (JVM), Dask, Joblib, Parsl (Python), Storm (continuous), Thrill (C++), DAGMan (HTCondor), TensorFlow (fitting)...

Physics software is embracing this approach:

- RDataFrame in ROOT
- Dozens of other examples at CHEP

RDataFrame from the ROOT Workshop (last week)

Case study: ATLAS SUSY ntuple → ntuple

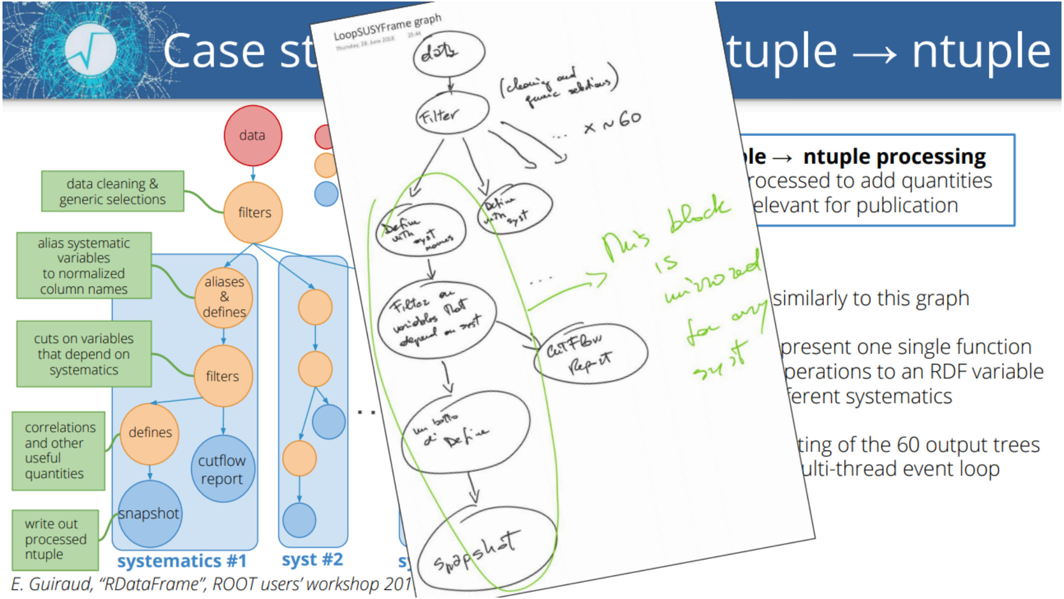


E. Guiraud, "RDataFrame", ROOT users' workshop 2018

Local ntuple → ntuple processing
MC data is processed to add quantities that are relevant for publication

- ➔ program's **main** reads similarly to this graph
- ➔ the large blue boxes represent one single function that applies the same operations to an RDF variable and is re-used for all different systematics
- ➔ cuts, calculations and writing of the 60 output trees all happen in the same multi-thread event loop

RDataFrame from the ROOT Workshop (last week)



But how will it develop?

Will RDataFrame be a *programming model* that *interfaces* with distributed processing systems such as Spark?

Or will it be *part* of a ROOT *implementation* of distributed DAG processing? A new PROOF, for instance?

But how will it develop?

Will RDataFrame be a *programming model* that *interfaces* with distributed processing systems such as Spark?

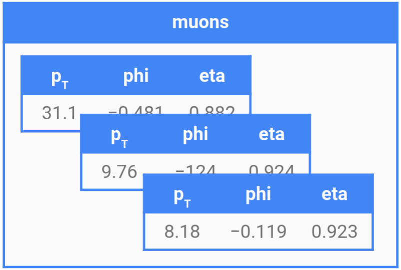
Or will it be *part* of a ROOT *implementation* of distributed DAG processing? A new PROOF, for instance?

Distributing computational tasks with dependencies is a good example of a non-domain-specific problem.

Nested data structures

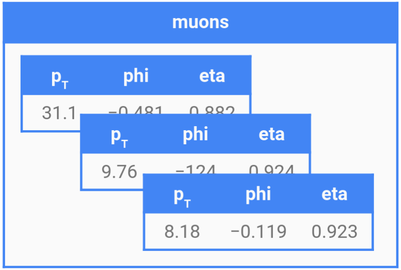
strangely physics-specific, but shouldn't be

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

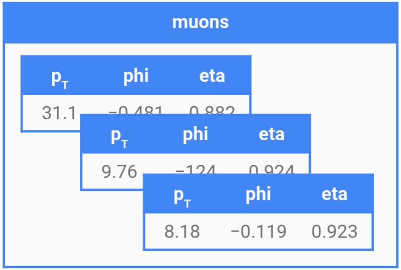


mu1 p_T	mu1 phi	mu1 eta	mu2 p_T	mu2 phi	mu2 eta
31.1	−0.481	0.882	9.76	−0.124	0.924
5.27	1.246	−0.991	n/a	n/a	n/a
4.72	−0.207	0.953	n/a	n/a	n/a
8.59	−1.754	−0.264	8.714	0.185	0.629

Objects are essential in physics 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 physics 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 → 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-physics 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-physics frameworks.

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

General programming model

```python
@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

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

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-physics 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
```

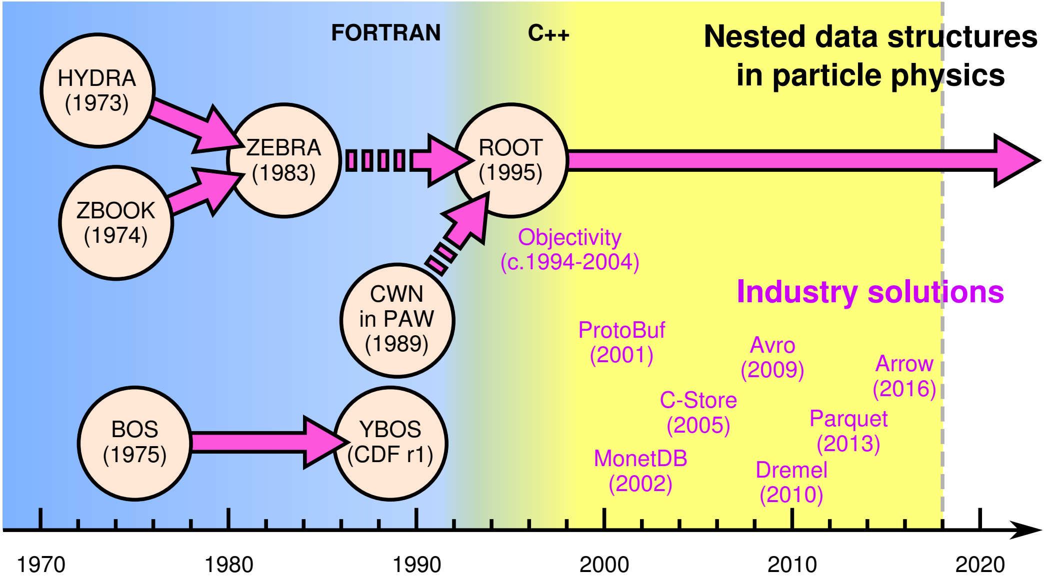
Numpy-like broadcasting

Awkward Array

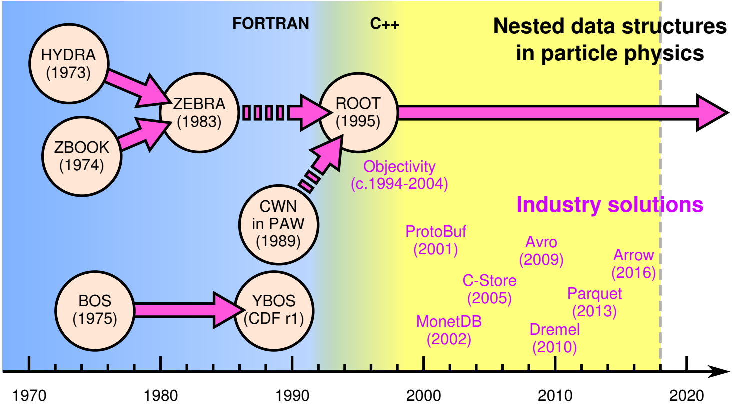
```
# one per event           one per particle
event["MET"]["phi"] - event["jet"]["phi"]
```

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

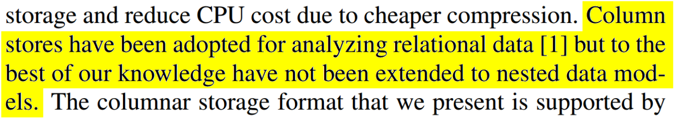
Missed opportunity



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 our field

What is indexing?

A way of organizing analysis:

- PAW/HBOOK histograms were indexed by *integer IDs*
- ROOT histograms are indexed by *string names*
- ROOT TTrees are indexed by *integer entry numbers*
- Excel spreadsheets are indexed by *integer row and letter column IDs*
- SQL tables are indexed by *unordered sets*
- Pandas DataFrames are indexed by *ordered, structured series*

Pandas is not a TTree replacement!

Most of Pandas's functionality is about manipulating data by index, and the whole DataFrame must fit in memory.

```python
result = left.join(right, how="inner")
```

left

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

right

		C	D
K0	Y0	C0	D0
K1	Y1	C1	D1
K2	Y2	C2	D2
K2	Y3	C3	D3

Result

		A	B	C	D
K0	Y0	A0	B0	C0	D0
K1	Y1	A1	B1	C1	D1
K2	Y2	A2	B2	C2	D2
K2	Y3	A2	B2	C3	D3

None (?) of TTree's functionality is about manipulating data by index, and it's focused on lazily loading large datasets.

Pandas DataFrames resemble histograms, not TTrees

Histograms are mappings from intervals to weighted counts and their errors.

Profiles are mappings from intervals to weighted means and standard deviations.

Pandas has an interval key type, as well as MultiIndexes for multiple dimensions.

Pandas DataFrames resemble histograms, not TTrees

Histograms are mappings from intervals to weighted counts and their errors.

Profiles are mappings from intervals to weighted means and standard deviations.

Pandas has an interval key type, as well as MultiIndexes for multiple dimensions.

Since the index is explicit, rather than by position in an array, there's no distinction between sparse and dense histograms. (Drop zero-valued rows and impute zero on merge.)

Pandas DataFrames resemble histograms, not TTrees

Histograms are mappings from intervals to weighted counts and their errors.

Profiles are mappings from intervals to weighted means and standard deviations.

Pandas has an interval key type, as well as MultiIndexes for multiple dimensions.

Since the index is explicit, rather than by position in an array, there's no distinction between sparse and dense histograms. (Drop zero-valued rows and impute zero on merge.)

Fluidly convert between MultiIndexes and columns with `pivot_table()`.

Pandas DataFrames resemble histograms, not TTrees

Histograms are mappings from intervals to weighted counts and their errors.

Profiles are mappings from intervals to weighted means and standard deviations.

Pandas has an interval key type, as well as MultiIndexes for multiple dimensions.

Since the index is explicit, rather than by position in an array, there's no distinction between sparse and dense histograms. (Drop zero-valued rows and impute zero on merge.)

Fluidly convert between MultiIndexes and columns with `pivot_table()`.

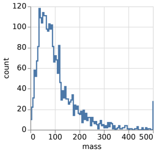
(Stop making arrays and `std::maps` of TH3!)

Indexed analysis

```
>>> 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()
                  MultiIndex key                         Columns
---------------------------------------------------+---------------------
                                                   | 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
```

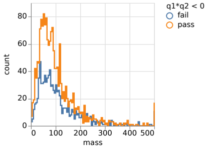
Indexed analysis

```
>>> 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")
```



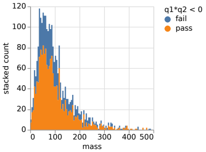
Indexed analysis

```
>>> 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")
```



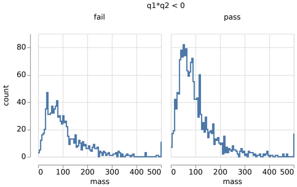
Indexed analysis

```
>>> 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")
```



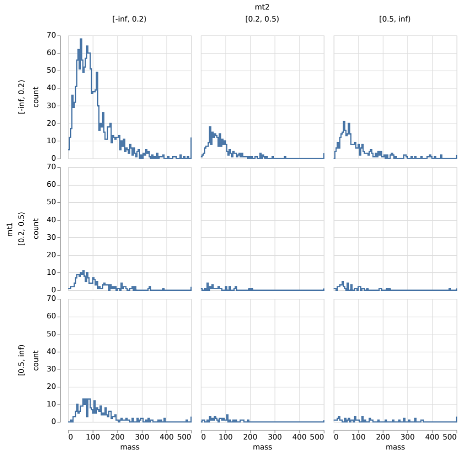
Indexed analysis

```
>>> 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")
```



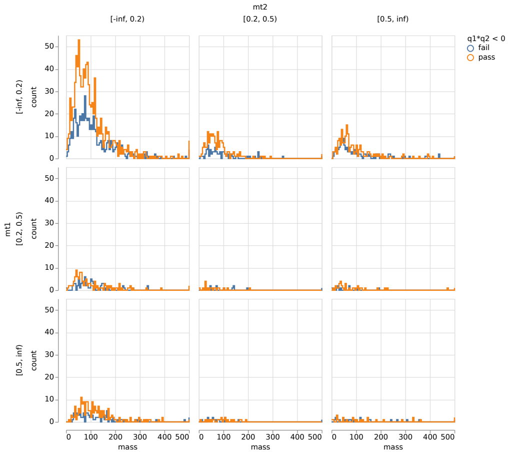
Indexed analysis

```
>>> 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

```
>>> 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")
```



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

			name	pass	total
component	depth	class			
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

Example Cut-flow Dataframe

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

			name	pass	total
component	depth	class			
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

10 July 2018, b.krikler@cern.ch 13 FAST: Pandas-based binned analysis

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**
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
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.

Advanced histogramming

very physics-specific

Advanced histogramming

The histograms themselves, however, are more sophisticated in particle physics software than elsewhere.

Advanced histogramming

The histograms themselves, however, are more sophisticated in particle physics software than elsewhere.

- As far as I have found, *only* particle physics packages (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled.

  Exception: Boost.Histogram, currently under review (Hans Dembinski, LHCb)

Advanced histogramming

The histograms themselves, however, are more sophisticated in particle physics software than elsewhere.

- As far as I have found, *only* particle physics packages (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled.

  Exception: Boost.Histogram, currently under review (Hans Dembinski, LHCb)

- In non-physics packages, "histogram" is more of a display option than an analysis tool, with no way to access contents or control binning.

The histograms themselves, however, are more sophisticated in particle physics software than elsewhere.

- As far as I have found, *only* particle physics packages (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled.

  Exception: Boost.Histogram, currently under review (Hans Dembinski, LHCb)

- In non-physics packages, "histogram" is more of a display option than an analysis tool, with no way to access contents or control binnning.
- Profile plots are only in our tools.

Advanced histogramming

The histograms themselves, however, are more sophisticated in particle physics software than elsewhere.

- As far as I have found, *only* particle physics packages (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled.

  Exception: Boost.Histogram, currently under review (Hans Dembinski, LHCb)

- In non-physics packages, "histogram" is more of a display option than an analysis tool, with no way to access contents or control binnning.
- Profile plots are only in our tools.
- Good log-scale handling is hard to find, too.

Advanced histogramming

The histograms themselves, however, are more sophisticated in particle physics software than elsewhere.

- As far as I have found, *only* particle physics packages (ROOT, YODA, go-hep/hbook, AIDA, HippoDraw, Jas3, mn_fit, PAW, HBOOK) conceive of histograms as containers to be filled.

  Exception: Boost.Histogram, currently under review (Hans Dembinski, LHCb)

- In non-physics packages, "histogram" is more of a display option than an analysis tool, with no way to access contents or control binnning.

- Profile plots are only in our tools.

- Good log-scale handling is hard to find, too.

These features are our responsibility.

Machine learning versus ansatz fitting

Machine learning: **it's just fitting.**

Machine learning: **it's just 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.

Machine learning: **it's just 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

Machine learning: **it's just 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.**

Machine learning versus ansatz fitting

Machine learning: **it's just 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 our field: RooFit, GooFit, HistFitter, HistFactory, Combiner, 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

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

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 particle physics 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 physics 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 particle physics 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 physics 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!