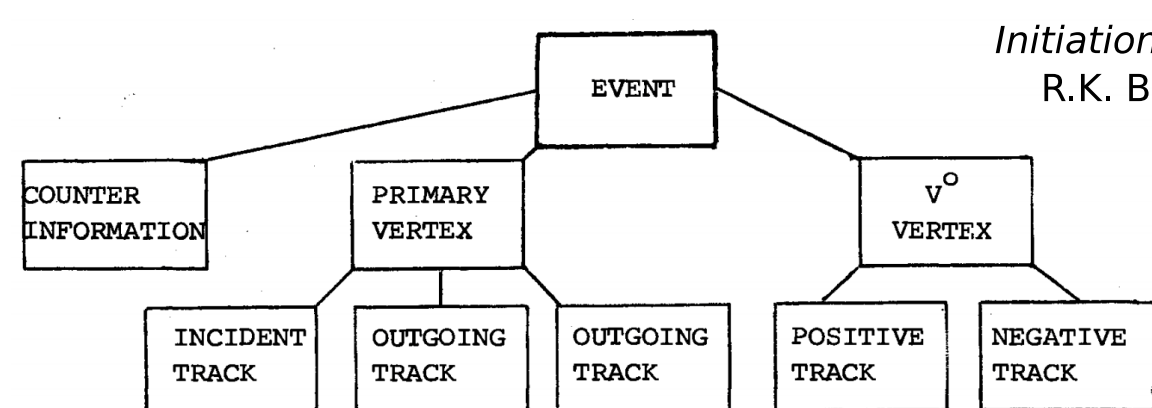


Why it's needed

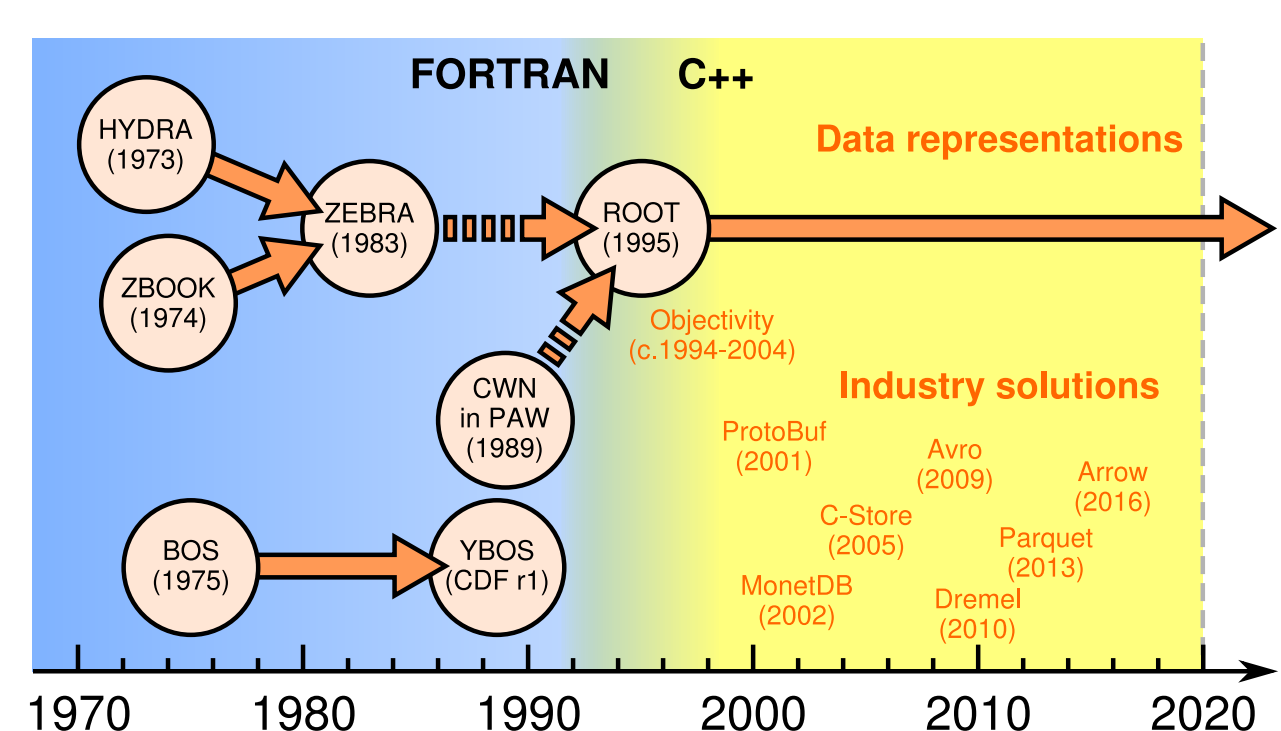
Particle physicists have always needed **big datasets** of **nested, variable-sized data**.

Figure from a 45-year old physics-software manual:



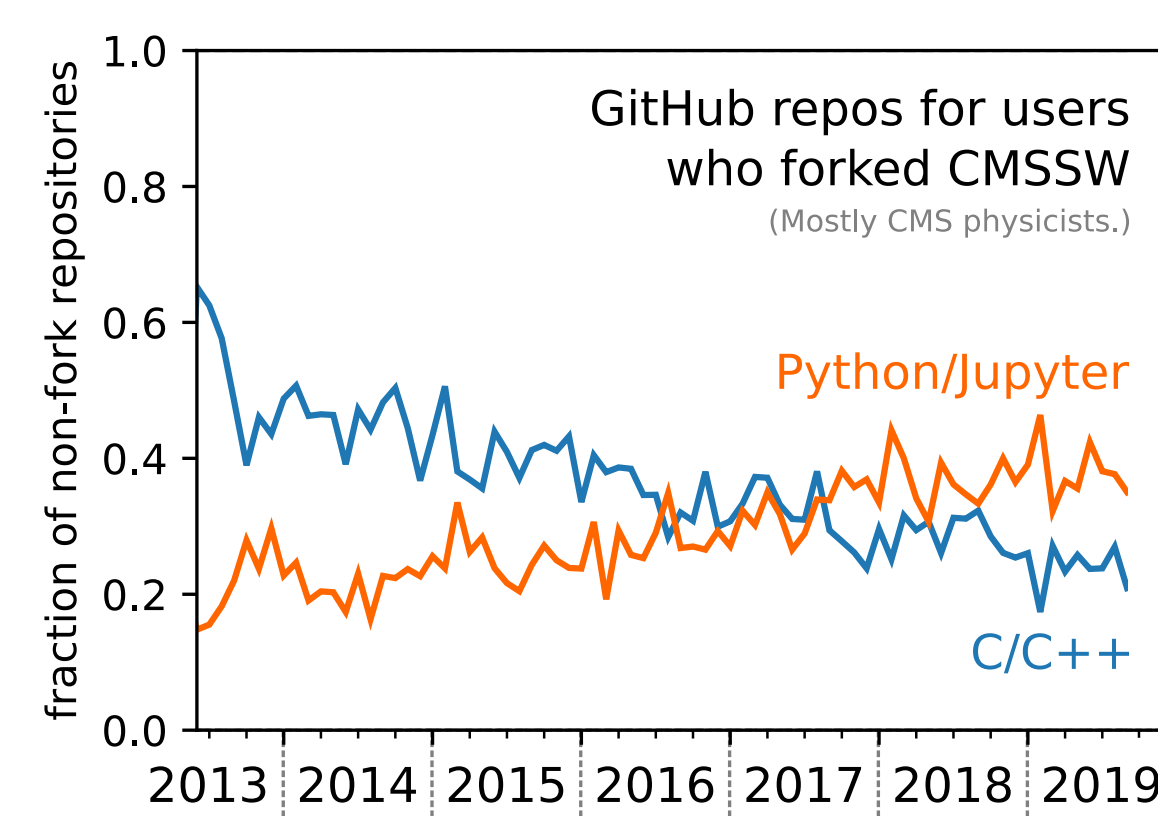
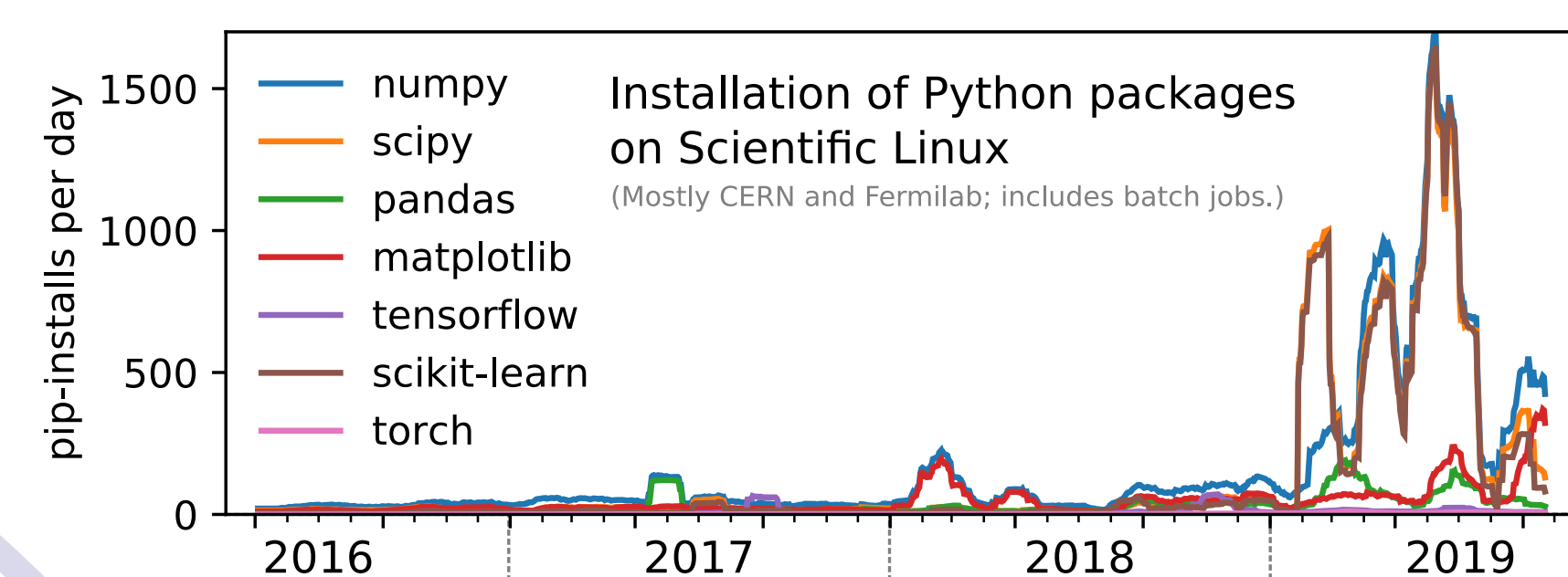
We'd draw similar figures today!

Traditionally, this problem was solved by making data analysts use ~~Fortran~~ C++.



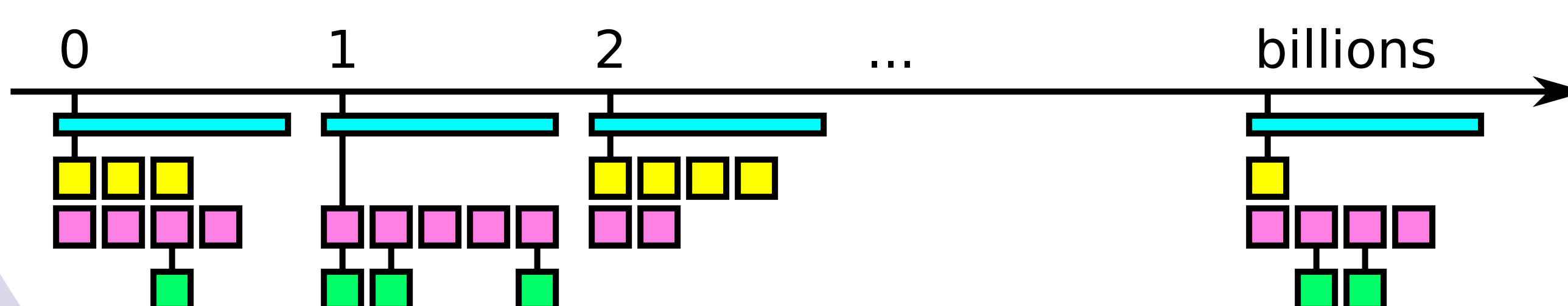
Why now?

Python/NumPy is rapidly becoming a standard language for data analysis in particle physics.



Awkward Array

An array library for **nested, variable-sized data**, including arbitrary-length lists, records, mixed types, and missing data, using **NumPy-like idioms**.



Arrays are **dynamically typed**, but operations on them are **compiled and fast**.

Coincides with NumPy when arrays are regular; generalizes when they're not.

But...
NumPy does not work on nested, variable-sized data!

What it does

Jagged broadcasting of NumPy "ufuncs"

```
[[[1,2,3],[4],[5,6]]]
+
[[[1],[2],[3]]]
[[[1,2,3],[4],[5,6]]]
```

Advanced indexing

```
# select muons 0,1 from events
>>> events[:, "muons", [0,1]]

# select muons with pt > 50
>>> events[events["muons", "pt"] > 50]
```

Combinatorics

```
[[[1,2,3],[4],[5,6]]]
⊗
[[[1],[2],[3]]]
[[[1,2,3],[4],[5,6]]]
```

Reshaping for plotting and machine learning

```
[[[1,2,3],[4],[5,6]]] → flatten → [1,2,3,4,5,6]
[[[1,2],[3],[4]]] → rpad → [[1,2],[NA,NA],[3,NA]]
```

Jagged reducers

```
[[[1,2,4],[8,16]]]
sum axis=1 → [7,0,24]
sum axis=0 → [9,18,4]
```

ROOT & Arrow/Parquet I/O

Originally intended as an array type for ROOT files, Awkward Arrays are convertible to/from Apache Arrow and Parquet (sometimes zero-copy).

Interface with Numba

Awkward Arrays can be arguments and return values in Numba's JIT-compiled functions, enabling for-loop logic at the speed of compiled code.

...with Pandas

Awkward Arrays can be columns of a Pandas DataFrame.

...NumExpr, Autograd, ...

Jagged broadcasting is applied to all elementwise array calculations.

How it works

Arrays and their operations are **columnar**.

Consider these lists of particle objects:

```
[Muon(31.1, -0.481, 0.882), Muon(9.76, -0.124, 0.924), Muon(8.18, -0.119, 0.923)],
[Muon(5.27, 1.246, -0.991)],
[Muon(4.72, -0.207, 0.953)],
[Muon(8.59, -1.754, -0.264), Muon(8.71, 0.185, 0.629)]
```

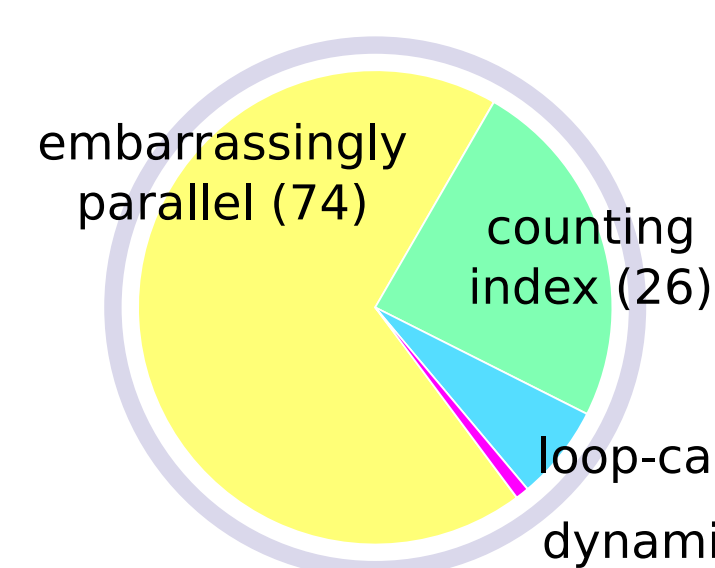
We represent them in columnar arrays, contiguous by field:

ListArray								
starts		0,		3,	4,	5		
stops		3,		4,	5,	7		
RecordArray								
"pT"	NumpyArray	31.1,	9.76,	8.18,	5.27,	4.72,	8.59,	8.71
"phi"	NumpyArray	-0.481,	-0.124,	-0.119,	1.246,	-0.207,	-1.754,	0.185
"eta"	NumpyArray	0.882,	0.924,	0.923,	-0.991,	0.953,	-0.264,	0.629

To transform the data, for example to remove the first element from each list, we only need to replace the ListArray:

```
[..., Muon(9.76, -0.124, 0.924), Muon(8.18, -0.119, 0.923)], [...], [...],
..., Muon(8.71, 0.185, 0.629)]
```

New ListArray								
starts		1,		4,	5,	6		
stops		3,		4,	5,	7		
Same RecordArray (by reference)								



The library consists of a suite of "kernels" that transform arrays into arrays.

Most are embarrassingly parallel and are good candidates for GPU acceleration.

Who uses it?

Mostly physicists, but a few geneticists and data scientists have expressed interest.

