Advances in Analysis tools/ecosystem

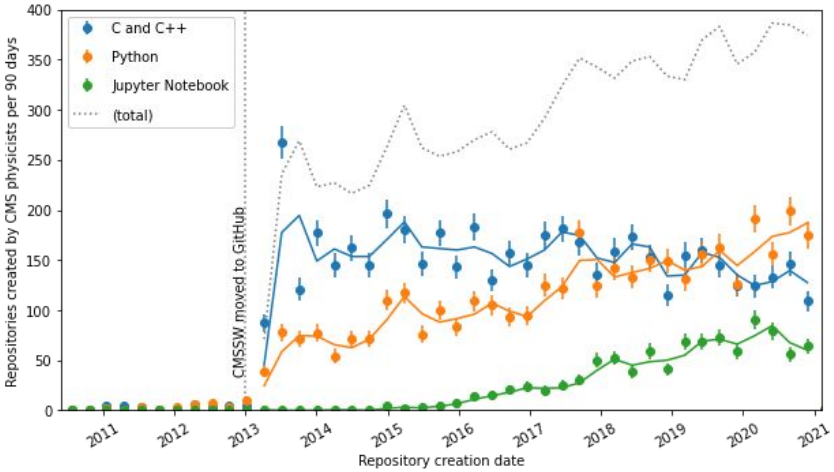
Oksana Shadura[1]
On behalf of the ALICE, ATLAS, CMS, and LHCb collaborations

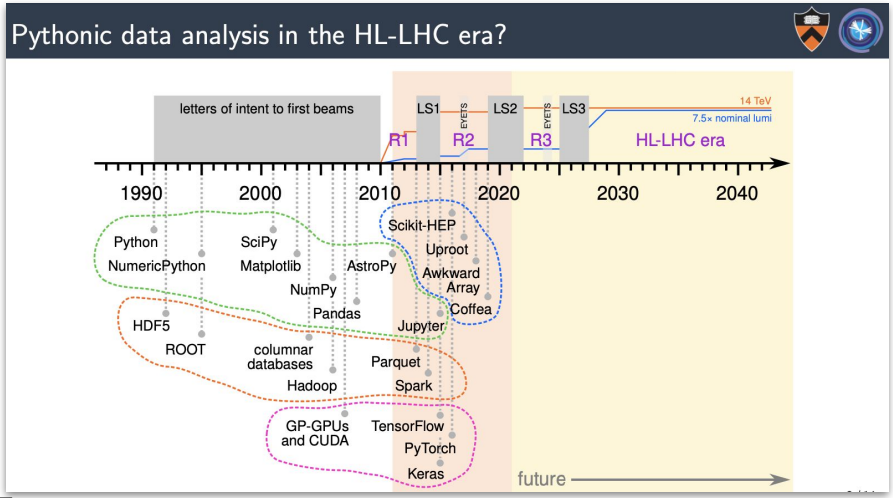
[1] oksana.shadura@cern.ch - University of Nebraska-Lincoln, USA

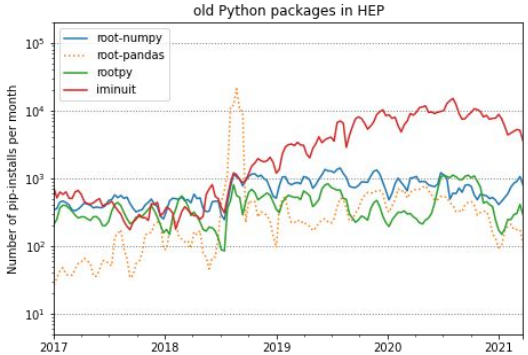
Evolution of analysis ecosystem for HL-LHC

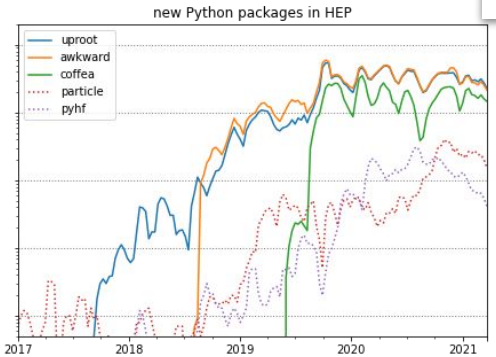
Analysis methods *<u>need to evolve to keep up with HL-LHC data ratio of data</u>* as well as **new opportunities from data science and evolution computational technology**!

Evolution of analysis ecosystem for HL-LHC





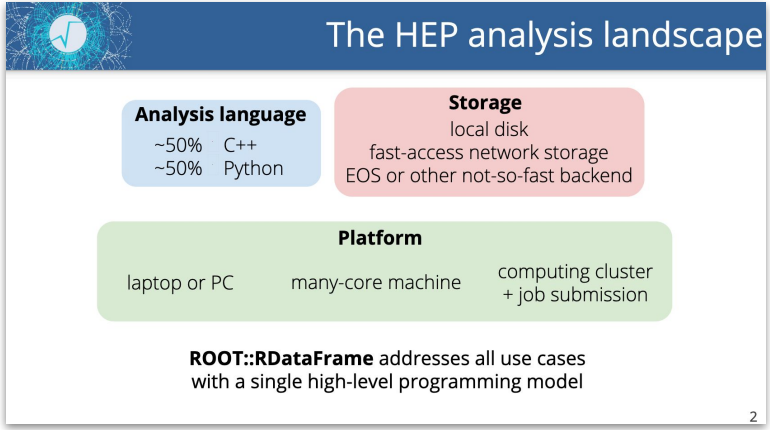




Figures from Jim Pivarski "Pythonic Data Analysis" @ ***HL-LHC Analysis Mini-Workshop*** *and* ***results from PyHEP2020 participants survey***

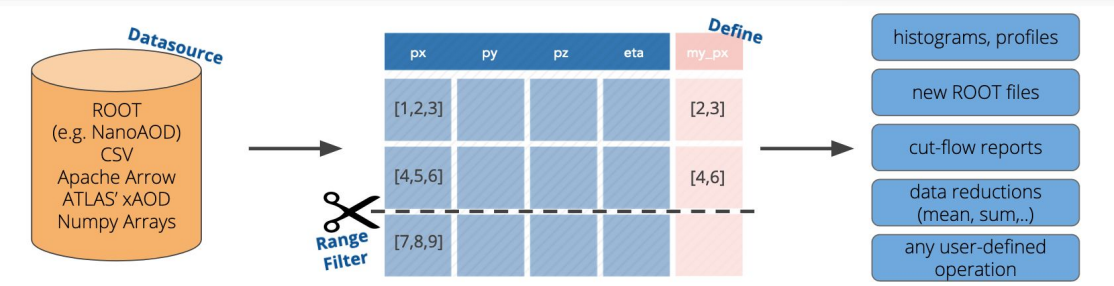
Making analysis in ROOT faster and more efficient

ROOT RDataframe



D Piparo, P Canal, E Guiraud et al. **"RDataFrame: Easy Parallel ROOT Analysis at 100 Threads"**

EPJ Web of Conferences **214**, 06029 (2019)
https://doi.org/10.1051/epjconf/201921406029




```
# enable multi-threading
ROOT.EnableImplicitMT()
df = ROOT.RDataFrame(dataset)
```

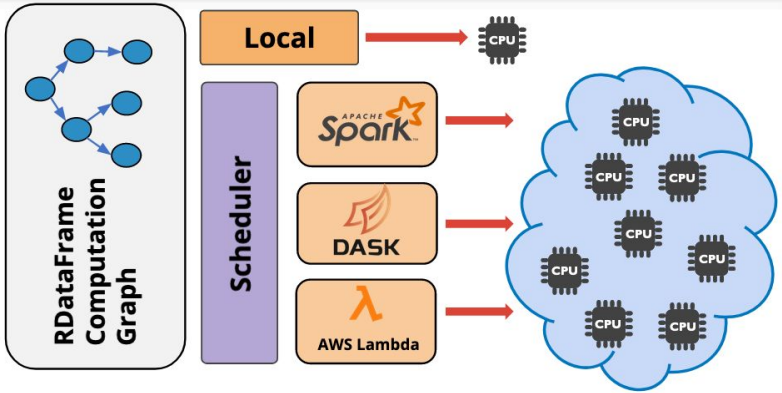
```
df = df.Range(2)
       .Define("my_px", "px[eta > 0]")
```

```
# filled in a single loop
h1 = df.Histo1D("my_px", "w")
h2 = df.Histo1D("px", "w")
```

The introduction of elements of **declarative programming** in the design (users say *what* they need to compute, RDataFrame chooses *how* to compute it) provides user-visible advantages such as less typing, increased readability and abstraction of complex operations.

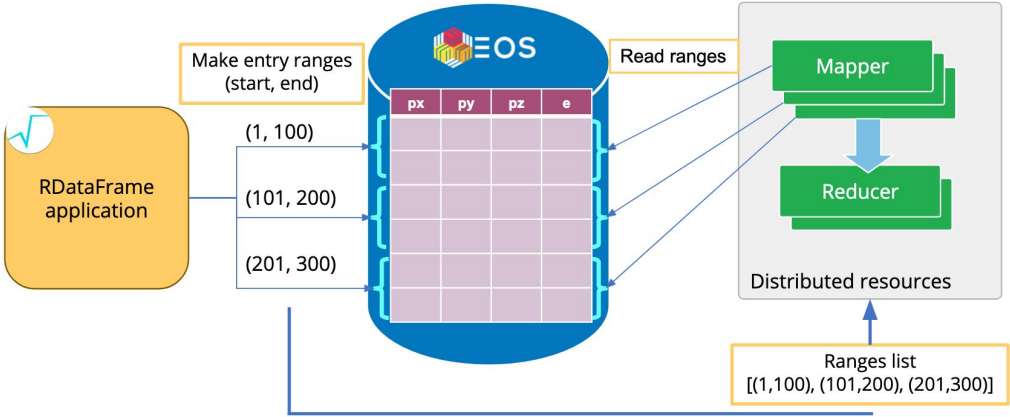
RDataframe tutorials

Distributed ROOT RDataframe




```
SparkRDF = ROOT.RDF.Experimental.Distributed.Spark.RDataFrame
df = SparkRDF(dataset)

df2 = df.Filter(...).Define(...) # No changes in user code
h1 = df2.Histo1D(...)
h1.Draw() # Transparently trigger distributed execution
```

- Enables interactive large-scale distributed data analysis with *RDataFrame*
- Can run with different schedulers
  - Spark
  - Dask
  - AWS
  - Other backends in the future
- Analysis from start to end in a single interface



ROOT RNTuple

Seamless transition from TTree to RNTuple

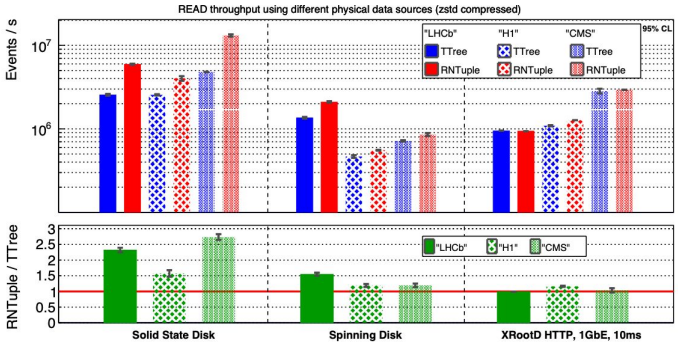
Event iteration
Reading and writing in event loops and through RDataFrame
`RNTupleDataSource, RNTupleView, RNTupleReader/Writer`

Logical layer / C++ objects
Mapping of C++ types onto columns
e.g. `std::vector<float>` ↦ index column and a value column
`RField, RNTupleModel, REntry`

Primitives layer / simple types
"Columns" containing elements of fundamental types (`float, int, ...`) grouped into (compressed) pages and clusters
`RColumn, RColumnElement, RPage`

Storage layer / byte ranges
`RPageStorage, RCluster, RNTupleDescriptor`

RNTuple provide modular storage layer that supports files as data containers but also file-less systems (object stores)!

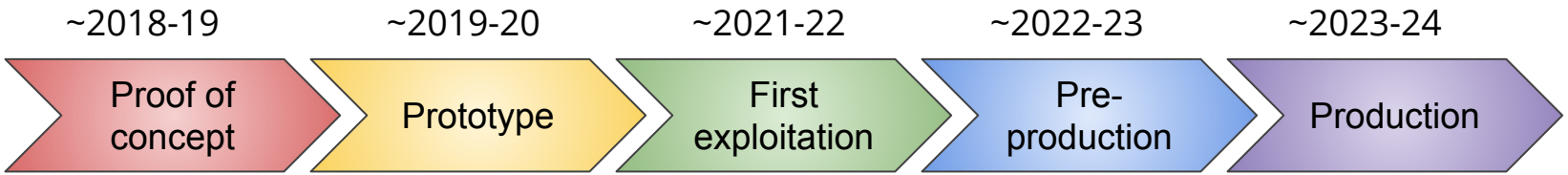


RNTuple R&D aiming at a **leap in data throughput**

- Updated (backwards incompatible) data format for next-generation event I/O
- *Expect ~10-15% smaller files,*
  x2-5 better single-core throughput on SSD
- Aims at using modern I/O devices to the full capacity
- Modern, robust API (e.g., thread-friendly, systematic use of exceptions)

Blomer, Jakob, et al. **"Evolution of the ROOT Tree I/O."**
EPJ Web of Conferences. Vol. 245. EDP Sciences, 2020.
https://doi.org/10.1051/epjconf/202024502030

ROOT RNTuple



- Class design
- File format R&D

- Interplay with other ROOT classes
- Performance validation

- Interplay with experiment frameworks
- Schema evolution

- PB scale test cases
- Production tests with last-stage ntuples

- Ready to use for new runs

ROOT team expect RNTuple to be adopted as a Run 4 technology

Available now in *ROOT::Experimental*
Note: *TTree* technology will remain available for the 1EB+ existing data sets

Extending data science tools to efficiently serve HEP needs

- *Community-driven* and *community-oriented effort* with the aim of providing HEP at large with a *toolset* ecosystem for data analysis in Python
- Widely adopted and used by physicists and *not only*
- Build a community of developers and users, having sustainability in mind

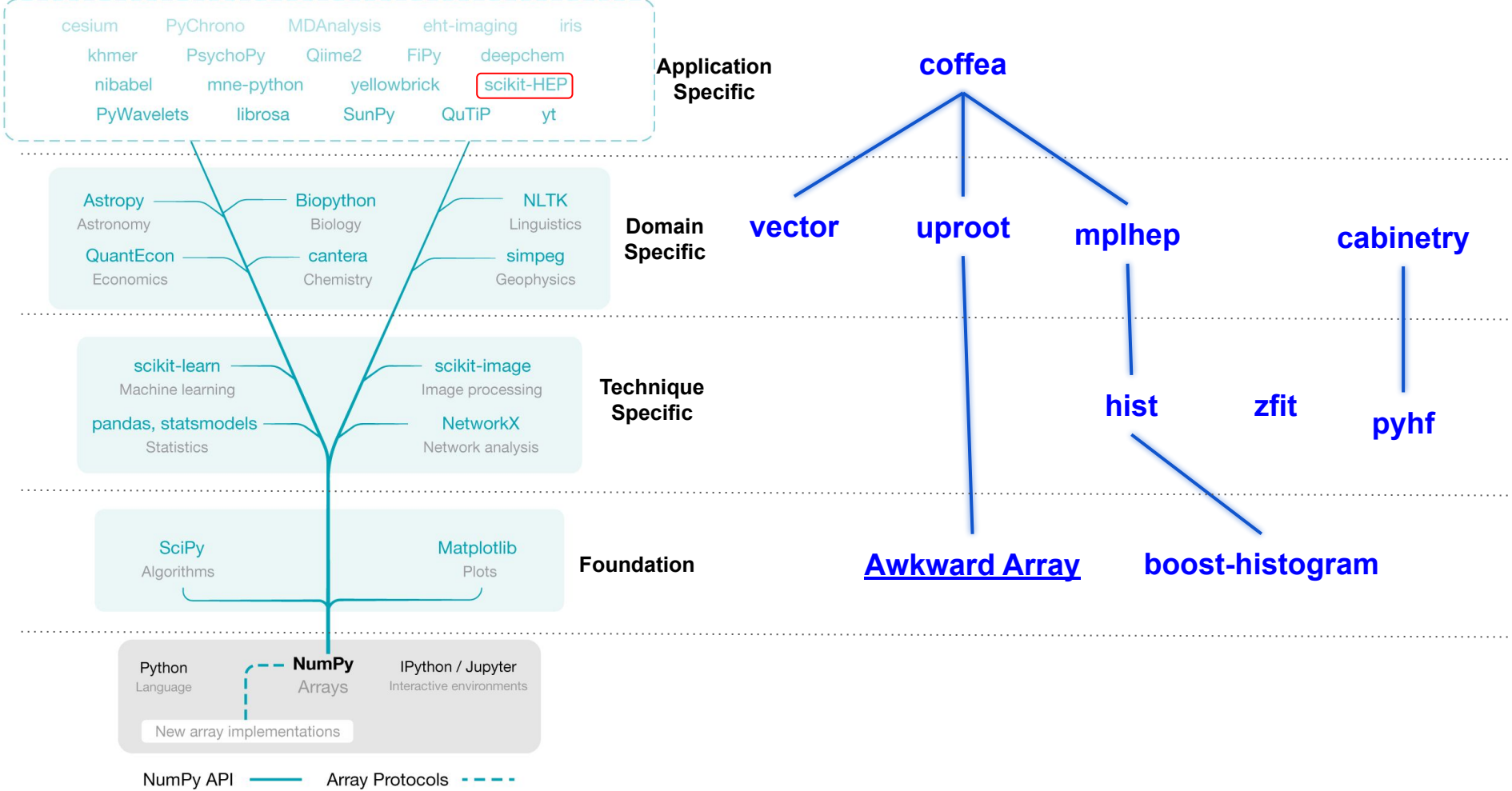
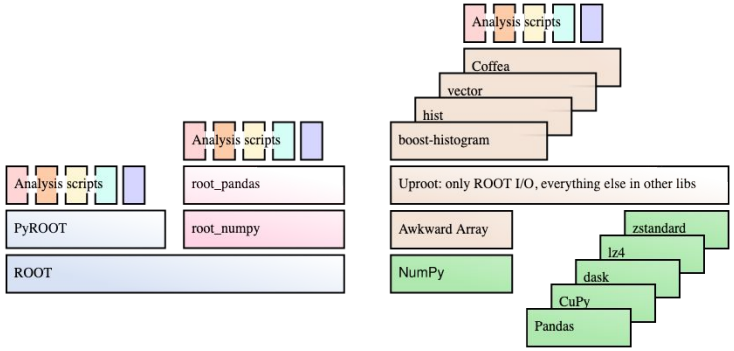


Figure from Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.* Array programming with NumPy.*Nature* 585, 357–362 (2020)

scikit-HEP packages in blue

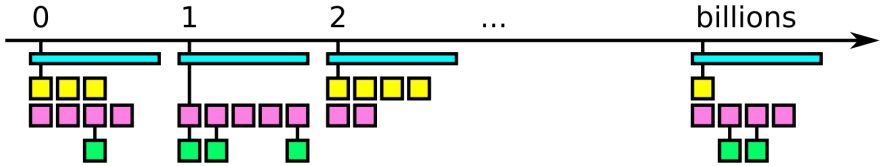
uproot



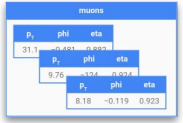
- Minimalist ROOT I/O in pure Python and Numpy
- Uproot easily bridges the ROOT and the NumPy-based ecosystems
- Unlike the standard C++ ROOT implementation, Uproot is only an I/O library, primarily intended to stream data into machine learning libraries in Python.

https://github.com/scikit-hep/uproot4

Awkward Array



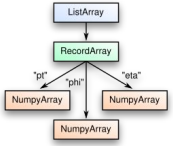
Logical view: particles as lists of nested objects




```
[[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)]
```

Physical layout: arrays grouped in a tree structure

```
offsets      0,                          3,      4,      5,          7
     pt   31.1,   9.76,   8.18,   5.27,   4.72,   8.59, 8.714
    phi -0.481, -0.123, -0.119,  1.246, -0.207, -1.754, 0.185
    eta  0.882,  0.924,  0.923, -0.991,  0.953, -0.264, 0.629
```



- Pure Python+Numpy library for manipulating complex data structures even if they:
  - Contain variable-length lists (jagged/ragged)
  - Are deeply nested (record structure)
  - Have different data types in the same list (heterogeneous)
  - Are not contiguous in memory

https://github.com/scikit-hep/awkward-1.0

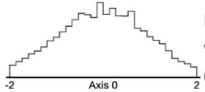
Hist

Hist is python bindings for Boost::Histogram, a C++14 library. (*Boost::Histogram is one of the fastest libraries for histogramming, while still providing the power of a full histogram object*)

In [1]:
```
import numpy as np
import hist
```

In [2]:
```
h = hist.Hist.new.Reg(30, -2, 2).Weight()
h.fill(np.random.normal(size=10_000))
h
```

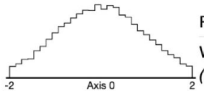
Out[2]:



Regular(30, -2, 2, label='Axis 0')
Weight() Σ=WeightedSum(value=9503, variance=9503)
(WeightedSum(value=10000, variance=10000) with flow)

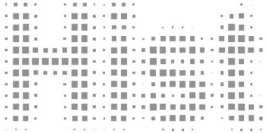
In [3]:
```
h.fill(np.random.normal(size=10_000), weight=3.)
h
```

Out[3]:



Regular(30, -2, 2, label='Axis 0')
Weight() Σ=WeightedSum(value=38120, variance=95354)
(WeightedSum(value=40000, variance=100000) with flow)

https://github.com/scikit-hep/hist



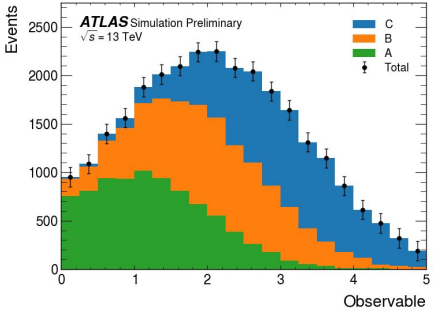
Boost-histogram: High-Performance Histograms as Objects |SciPy 2020| Schreiner, Pivarski & Dembinski

mplhep

- **Matplotlib** is a key tool for visualisation in the data science domain
  - *But it doesn't provide all that HEP wants*
- **Mplhep** distributes mpl stylesheets with required plotting styles for *ATLAS, CMS, ALICE and LHCb* and provides histogram plotting functions, which accept **hist** and **uproot(TH1)** histograms for convenience.

```
import mplhep as hep
hep.style.use(hep.style.ROOT) # For now ROOT defaults to CMS
# Or choose one of the experiment styles
hep.style.use(hep.style.ATLAS)
# or
hep.style.use("CMS") # string aliases work too
# {"ALICE" | "ATLAS" | "CMS" | "LHCb1" | "LHCb2"}
```

https://github.com/scikit-hep/mplhep



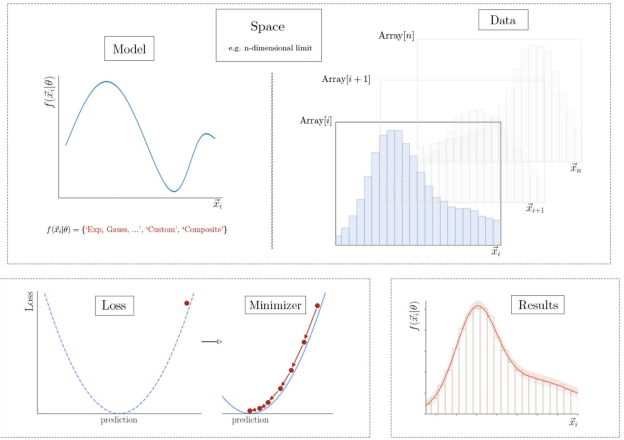
Vector

```
[1]: import vector

[2]: many_vectors = vector.array({
...      "px": [1.1, 1.2, 1.3, 1.4, 1.5],
...      "py": [2.1, 2.2, 2.3, 2.4, 2.5],
...      "pz": [3.1, 3.2, 3.3, 3.4, 3.5],
...      "E": [4.1, 4.2, 4.3, 4.4, 4.5],
... })

[3]: many_vectors.pt

[3]: array([2.37065392, 2.50599282, 2.64196896, 2.7784888 , 2.91547595])
```

- Vector implements *vectorized* **2D, 3D, and Lorentz vectors** in
  - Python objects, NumPy arrays, and Awkward Arrays
  - Transparent coordinate system conversion
  - Just-in-time compilation in Numba

https://github.com/scikit-hep/vector

zfit



zfit is designed for optimal parallelisation and scalability by making use of TensorFlow as its backend. The use of TensorFlow provides crucial features in the context of *model fitting* like taking care of the parallelisation and analytic derivatives

EPJ Web of Conferences **245**, 06025 (2020)
https://doi.org/10.1051/epjconf/202024506025

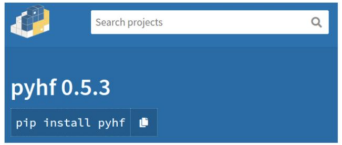
https://github.com/zfit/zfit

Pyhf and cabinetry

pyhf: HistFactory in pure Python

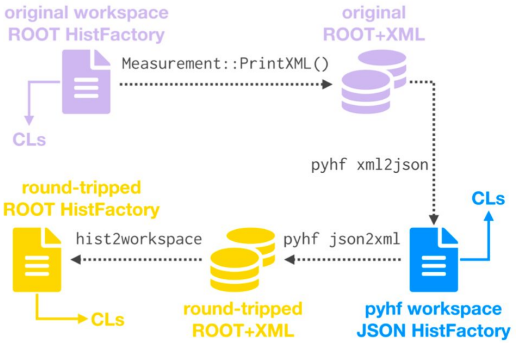
- First non-ROOT implementation of the HistFactory p.d.f. template
  - DOI 10.5281/zenodo.1169739
- pure-Python library with Python and CLI API
  - `$ pip install pyhf`
  - No dependence on ROOT!
- Open source tool for all of HEP
  - IRIS-HEP supported Scikit-HEP project
  - Used for reinterpretation in phenomenology paper (DOI: 10.1007/JHEP04(2019)144) and `SModelS`
  - Used in ATLAS SUSY groups and for internal pMSSM SUSY large scale reinterpretation
  - Maybe your experiment too!

pyhf
differentiable
Likelihoods



https://github.com/scikit-hep/pyhf

ROOT + XML to JSON and back



https://github.com/alexander-held/cabinetry/

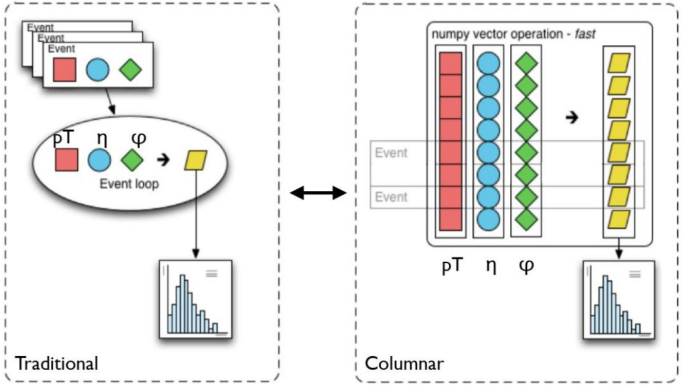
Building and steering template fits with cabinetry" Alex Held vCHEP 2021

- **cabinetry** is a **Python library** for constructing and operating **HistFactory** models
  - `>pip install cabinetry`
  - uses **pyhf** (**HistFactory** model in Python)
  - integrates seamlessly with the flourishing Python HEP ecosystem
  - modular design: drop in and out of **cabinetry** whenever needed

cabinetry

pyhf
differentiable
Likelihoods

Coffea Analysis Framework



New columnar data analysis concepts!

User just needs to define a high-level wrapper around user analysis code: **the coffea processor** and coffea framework will take care of everything incl. ***scaling-out***

Distributed executors!

Smith, Nicholas, Lindsey Gray, Matteo Cremonesi, Bo Jayatilaka, Oliver Gutsche, Allison Hall, Kevin Pedro et al. "COFFEA Columnar Object Framework For Effective Analysis." https://doi.org/10.1051/epjconf/202024506012
EPJ Web of Conferences, vol. 245, p. 06012. EDP Sciences, 2020.

https://github.com/CoffeaTeam/coffea

Coffea Analysis Framework

Persisting non-event data

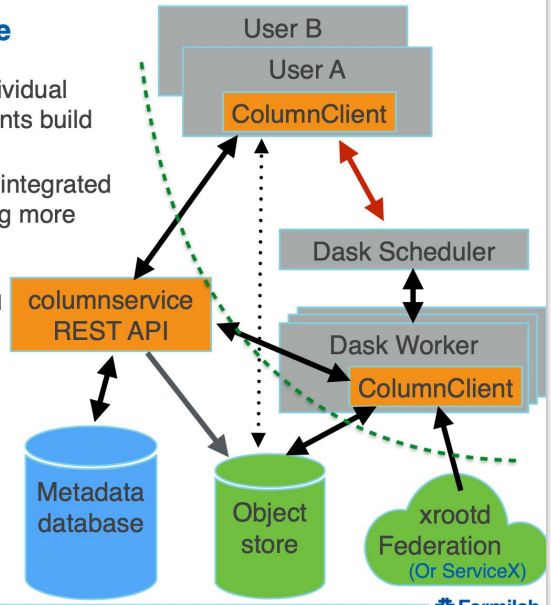
- We want a service that can decide when to cache function output
- Necessary ingredient: persist-able function definitions
  - Bonus: analysis preservation?
- Coffea distributed executors all use cloudpickle
  - No forward or backward compatibility guarantees for pickled python functions
  - Good for getting user code to scale-out mechanisms, bad for persistence
- Correctionlib may be a possible solution
  - Store corrections in JSON format with a flexible schema
  - Implement evaluator(s)
    - High-performance scalar function evaluator provided by library
    - High-level types handled by extension libraries
  - Join the fun: https://github.com/cms-nanoAOD/correctionlib

```
def f(*args: Union[str,int,float]) -> float:
    return ...

double Correction::evaluate(const std::vector<std::variant<int, double, std::string>>& values) const;
```

14 Feb. 17, 2021 Nick Smith I Metadata discussions

Columnservice prototype

- Manage the metadata of individual column objects and help clients build array chunks for processing
- Originally a k8s service with integrated dask cluster, now considering more lightweight solutions
  - Ideally ship columnservice with coffea, with e.g. SQLite for local and Postgres for site installs
  - User provides dask cluster, site provides object store (off the shelf)



11 Feb. 17, 2021 Nick Smith I Metadata discussions

HSF DAWG: Metadata discussions - Nick Smith
https://indico.cern.ch/event/993424/

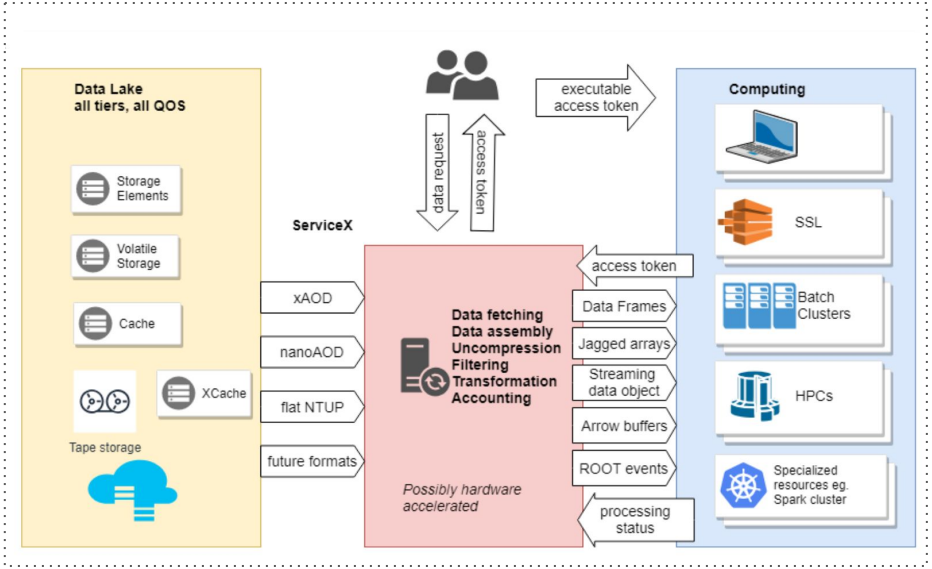
..and many more interesting developments!

R&D on various *services and facilities*
as a
demonstration of several technologies under development for use in HL-LHC analysis

ServiceX service

ServiceX is a scalable HEP event data extraction, transformation and delivery system (it provides user level ntuple production)

- Converts experiment-specific datasets to columns: **ATLAS xAOD/DAOD, CMS NanoAOD, ROOT Flat Ntuple**
- Enable simple cuts or simple derived columns and fields
- *Delivery:* deliver to a user or stream into Analysis System
- *Scalable:* runs on any Kubernetes cluster, scales up workers when necessary



[ServiceX doc]

[https://github.com/ssl-hep]

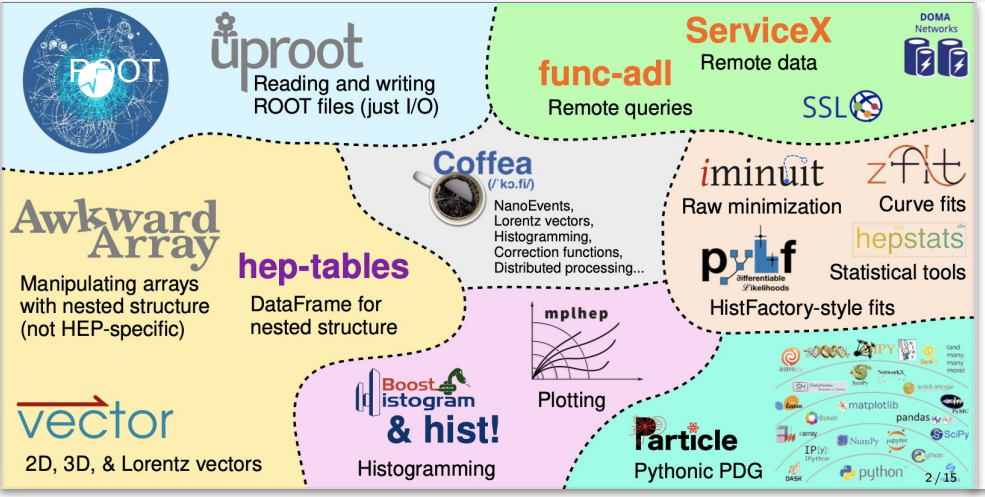
["Towards Real-World Applications of ServiceX, an Analysis Data Transformation System" Kyungeon Choi vCHEP]

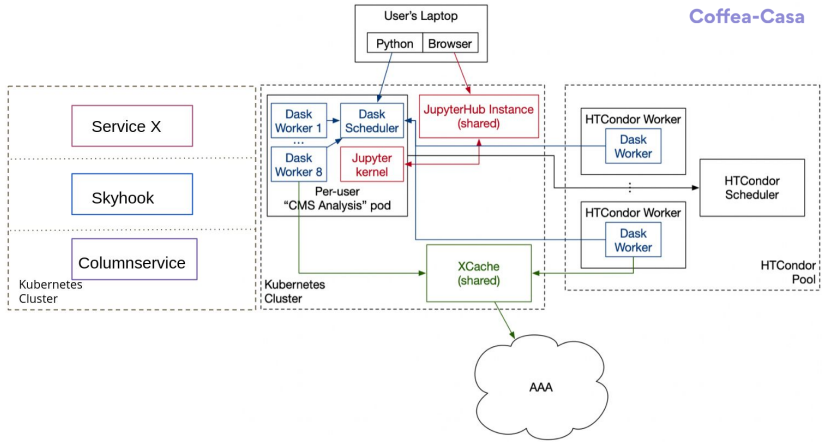
ServiceX

***Servicex** tested with* DASK *and*

Can be easily deployed on the local computer as well in analysis facility!

Broader ecosystem: analysis facility services development





….and many more

(IRIS-HEP Analysis Systems)

Coffea-casa: an analysis facility prototype', vCHEP 2021 plenary

Coffea-casa: an analysis facility prototype, M. Adamec, G. Attebury, K. Bloom, B. Bockelman, C. Lundstedt, O. Shadura and J. Thiltges, *arXiv* 2103.01871

SWAN - CERN's Notebook service

https://swan.cern.ch

- A SWAN session gives access to limited (shared) resources
  - CPU, memory
  - Not intended for heavy, long-running computation
  - Heavy computations should be **offloaded**
- SWAN can be used as the entry point to access computational resources
  - Examples: the Grid, Spark clusters, GPUs, HPC clusters (soon)

"Introduction to SWAN" - Diogo C., Prasanth K., Enric T., on behalf of the SWAN team
https://indico.cern.ch/event/847492/

GPUs for SWAN

- GPUs are available in the pilot instance of SWAN on Kubernetes at https://swan-k8s.cern.ch
  - It will become the default instance at https://swan.cern.ch
- In the pilot instance in CERN cloud, we are offering 5 GPUs (4x Tesla T4 + 1x V100)
  - If there is demand, we will ask for more from CERN cloud
- Software packages from CVMFS
  - The latest release (99 Cuda 10.1 Python3) has Tensorflow 2.3.0 and PyTorch 1.4.0
  - Other releases have older versions Tensorflow 2.1.0 etc
- How are the resources shared?
  - The user gets 1 GPU, 2 cores and 16 GB RAM from the available pool
  - Users are removed after 6 hours of inactivity

41

Conclusions

The life of analysts will be significantly *better* in 2027 (HL-LHC):

- Heavily data science technique/concept driven
- Less code and more expressive
- Easier scaling to cluster and other computing resources
- *The ultimate goal is to develop high-throughput, low-latency analysis systems to be able to handle data collected during HL-LHC era*