Image credit: Marguerite Tonjes

Inferring nature at scale

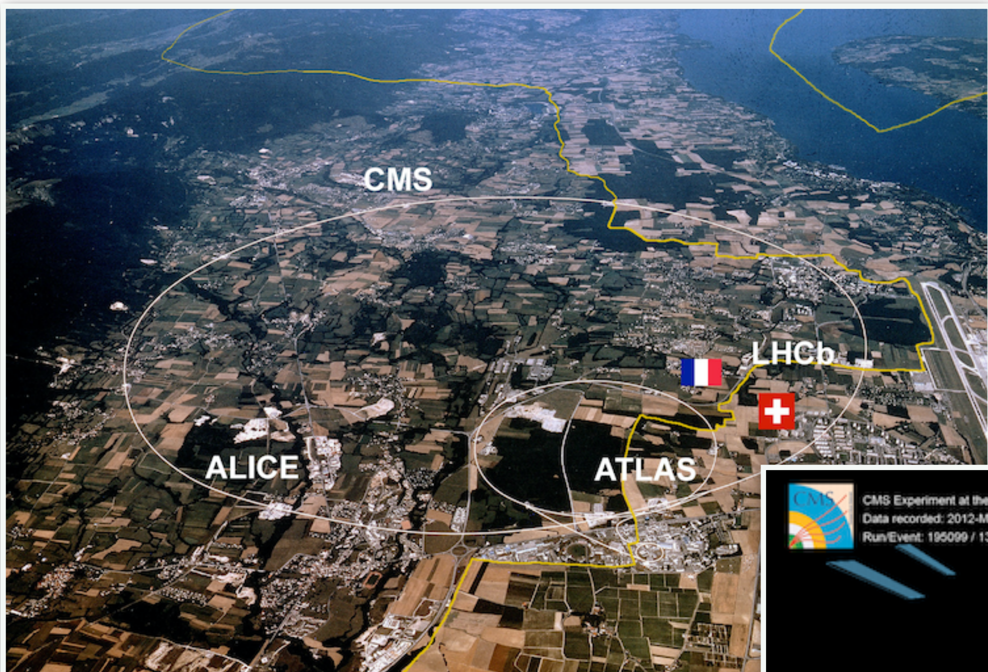
Innovative software tools for big data analysis in HEP

Nick Smith (FNAL)

Computational HEP Traineeship Summer School

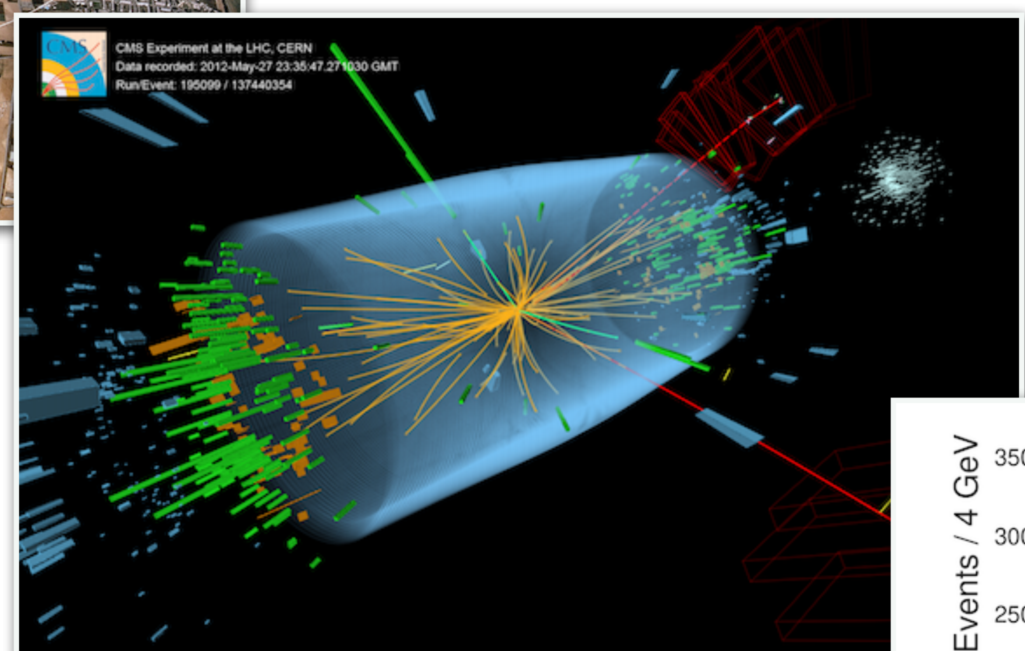
25 July 2023

HEP Experiment: three easy steps



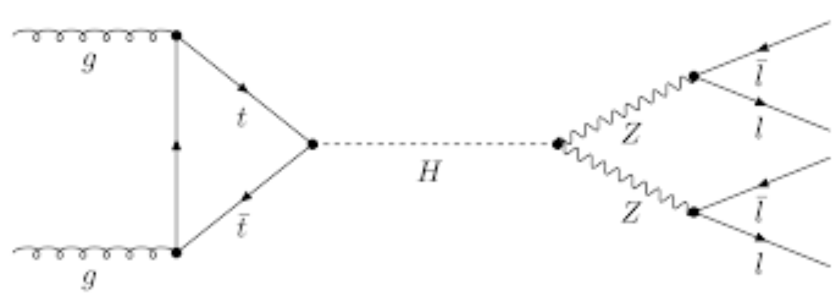
1. Collide particles

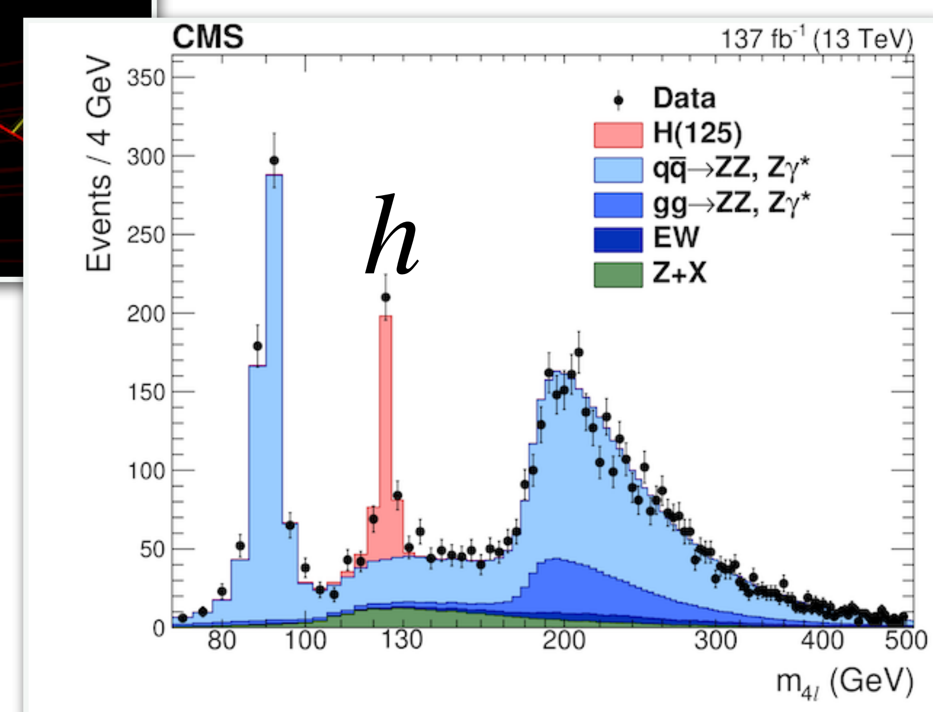
2. Take pictures



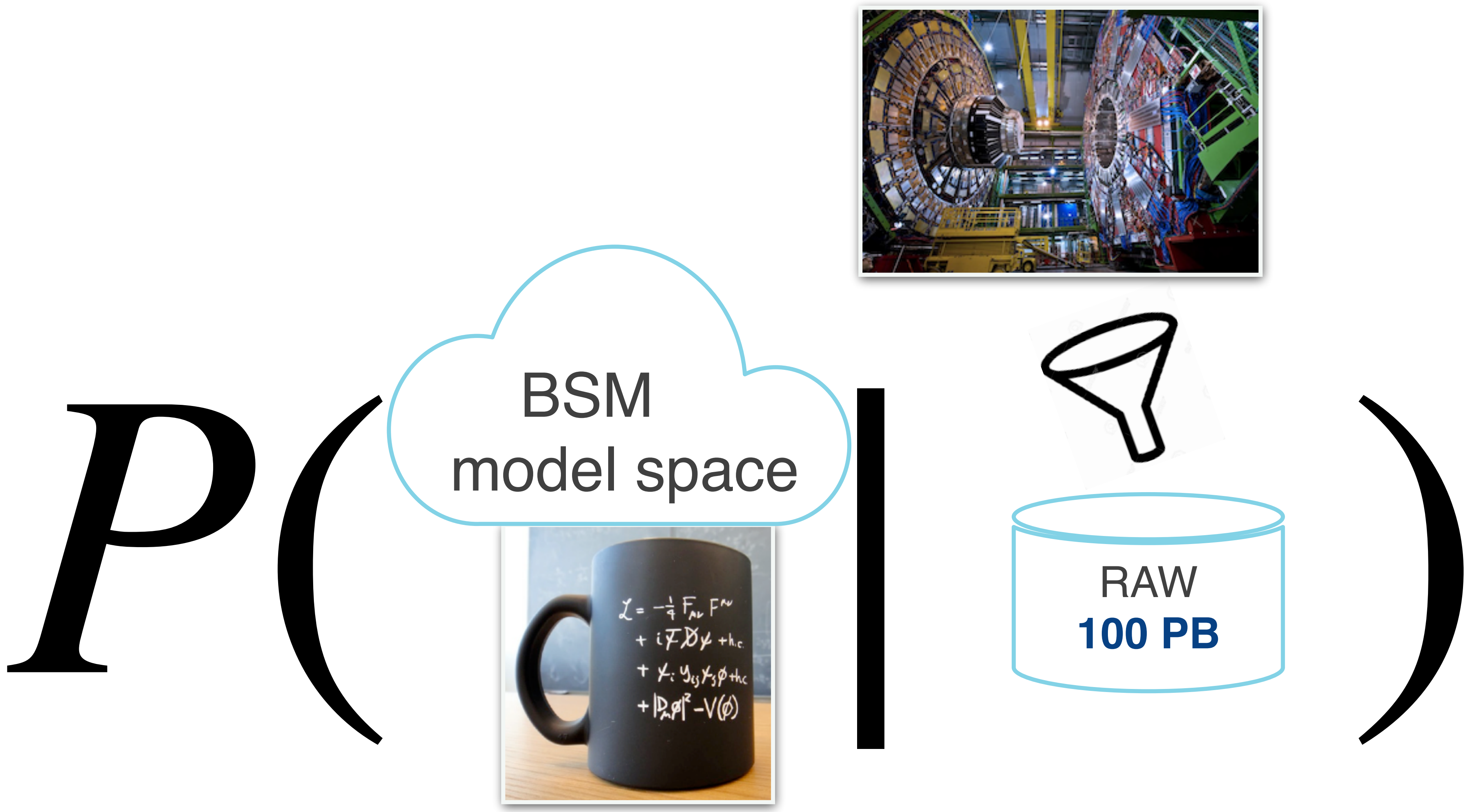
3. Infer parameters

~1MB / event
~100B events



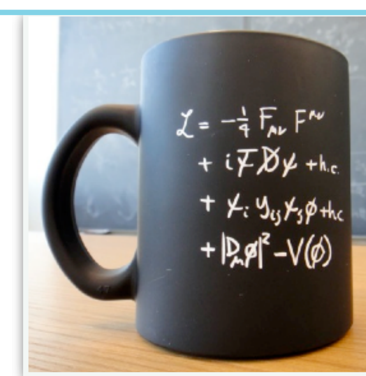


Inference: the dream



Inference: the dream
^ frequentist

$P($ RAW **100 PB** $|$ BSM model space



$)$

Inference: the dream

^ frequentist

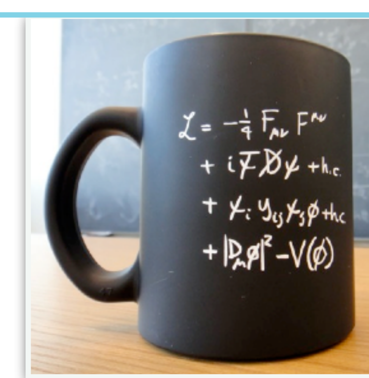
Need to reduce:
- Model space
- Data space

$P($

RAW
100 PB

$|$

BSM
model space

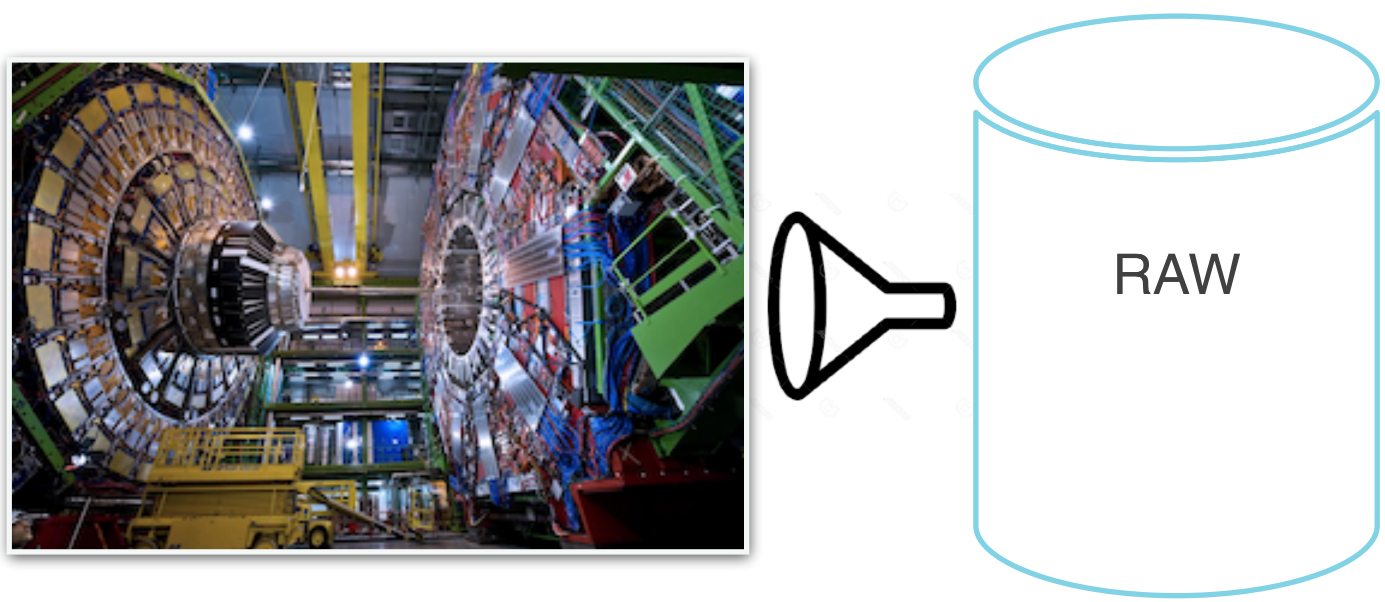


$)$

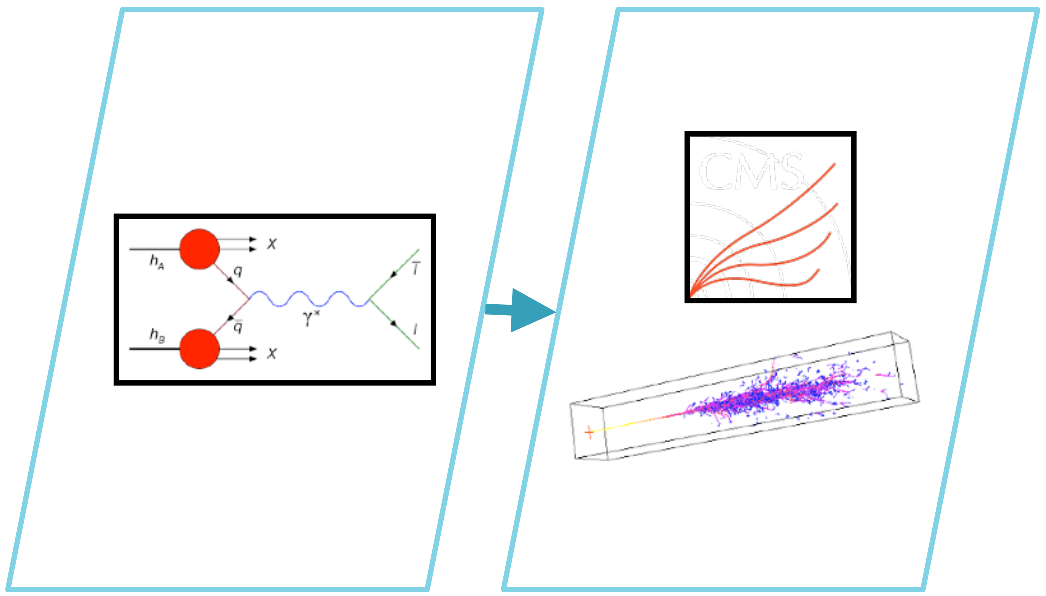
Fermilab

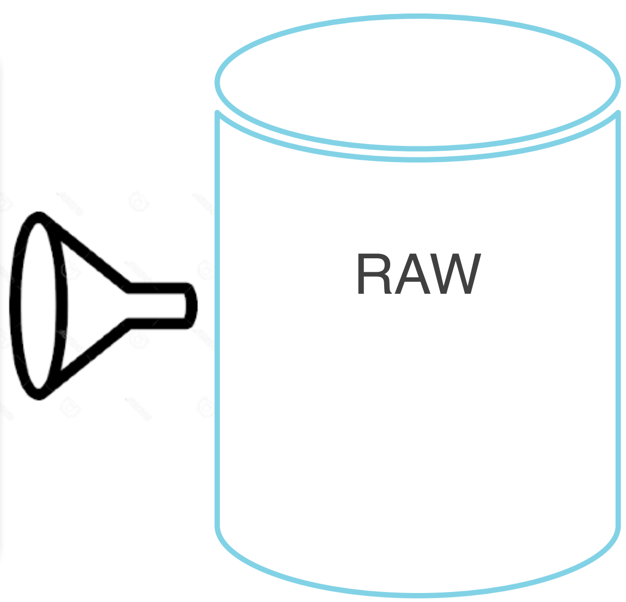
Inference: the reality

Inference: the reality

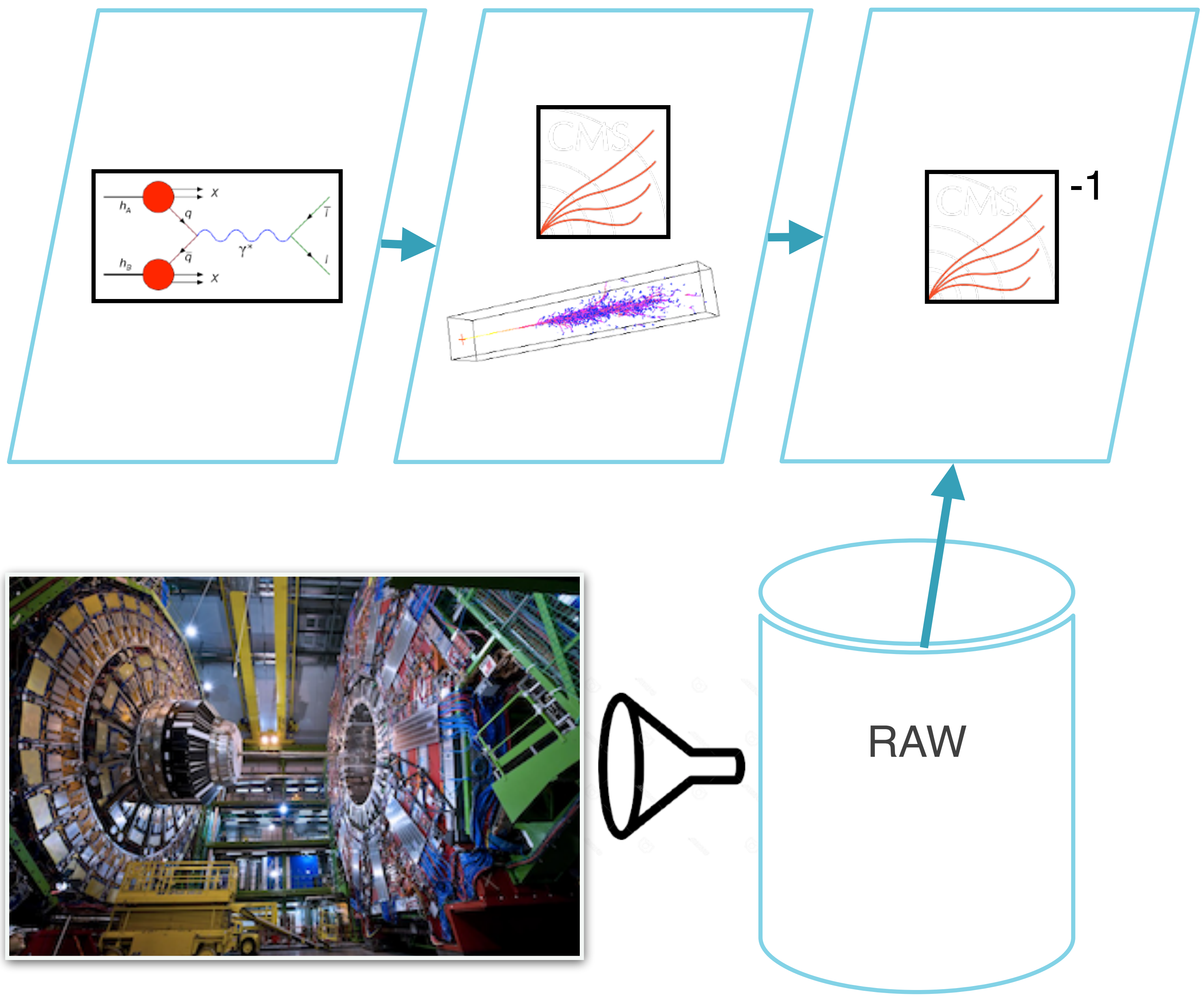


Inference: the reality

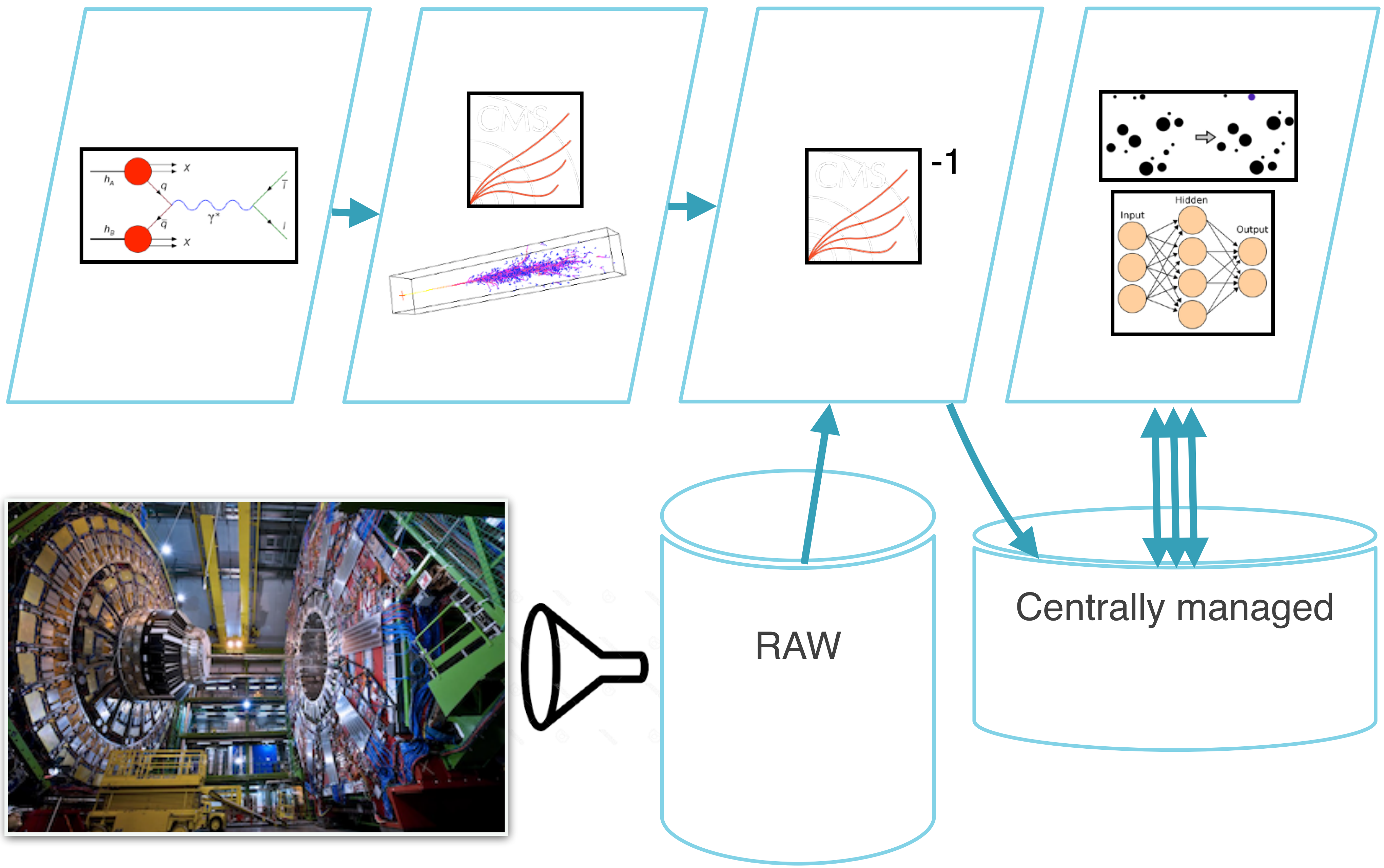




Inference: the reality

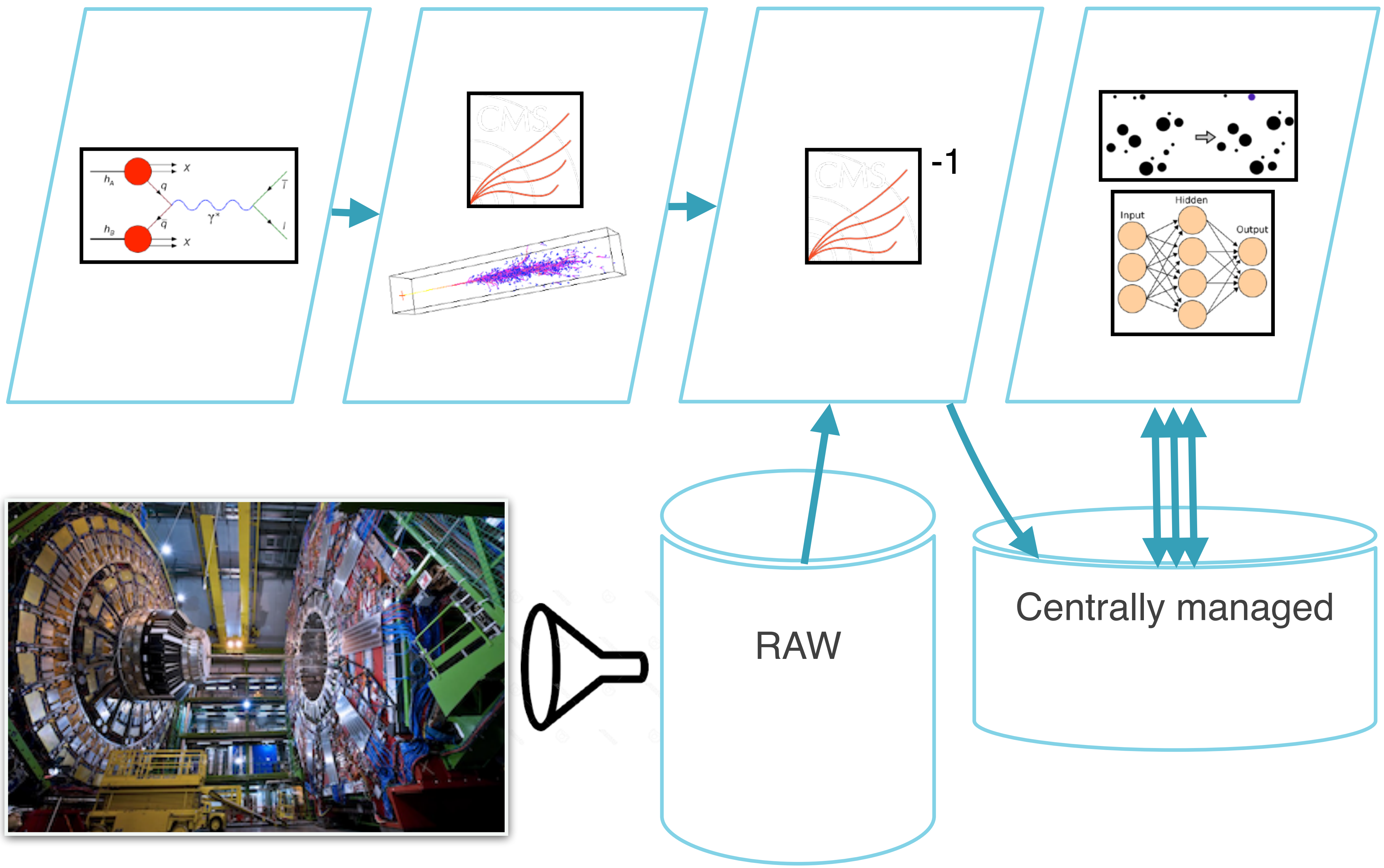


Inference: the reality



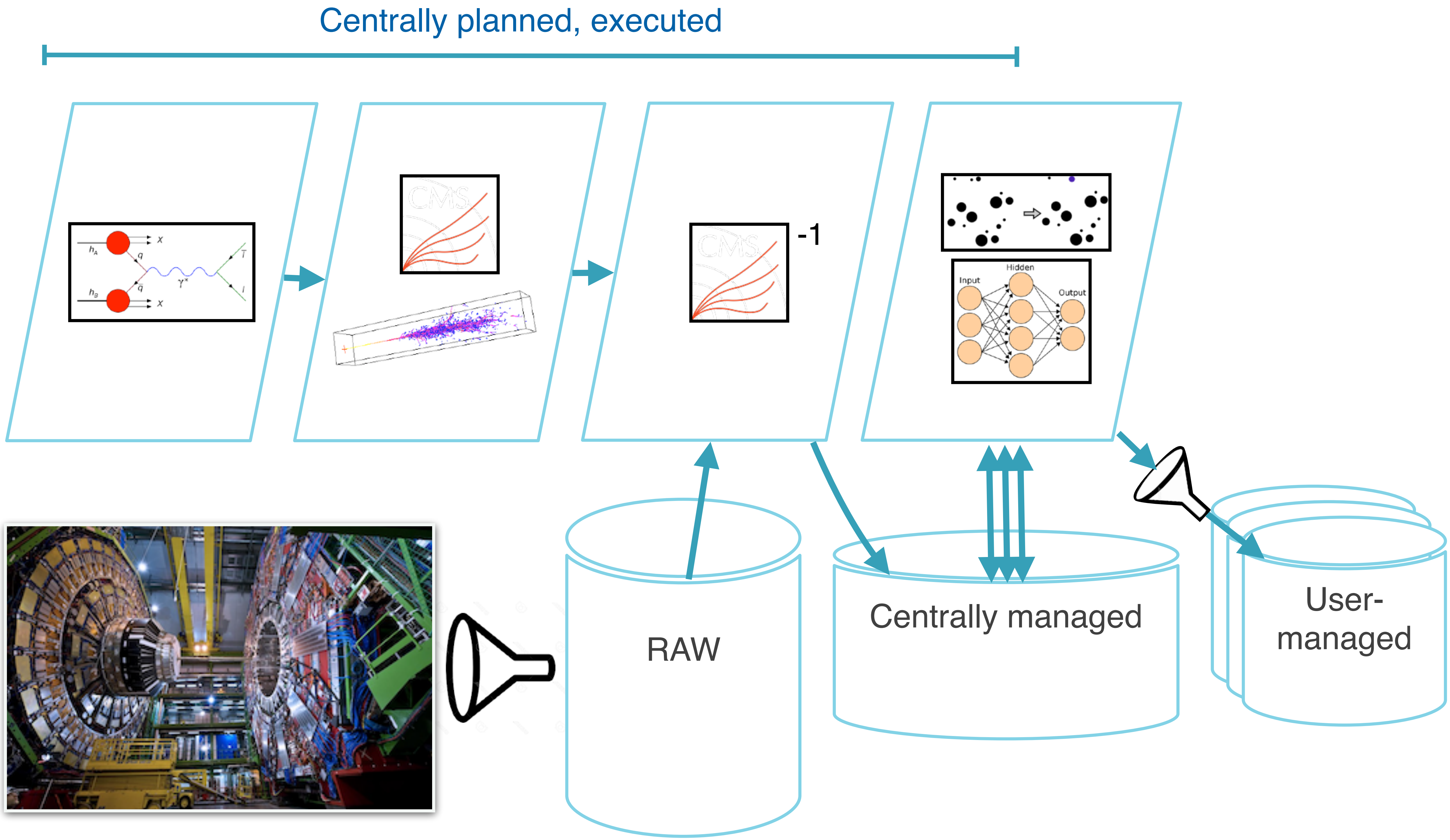
Inference: the reality

Centrally planned, executed

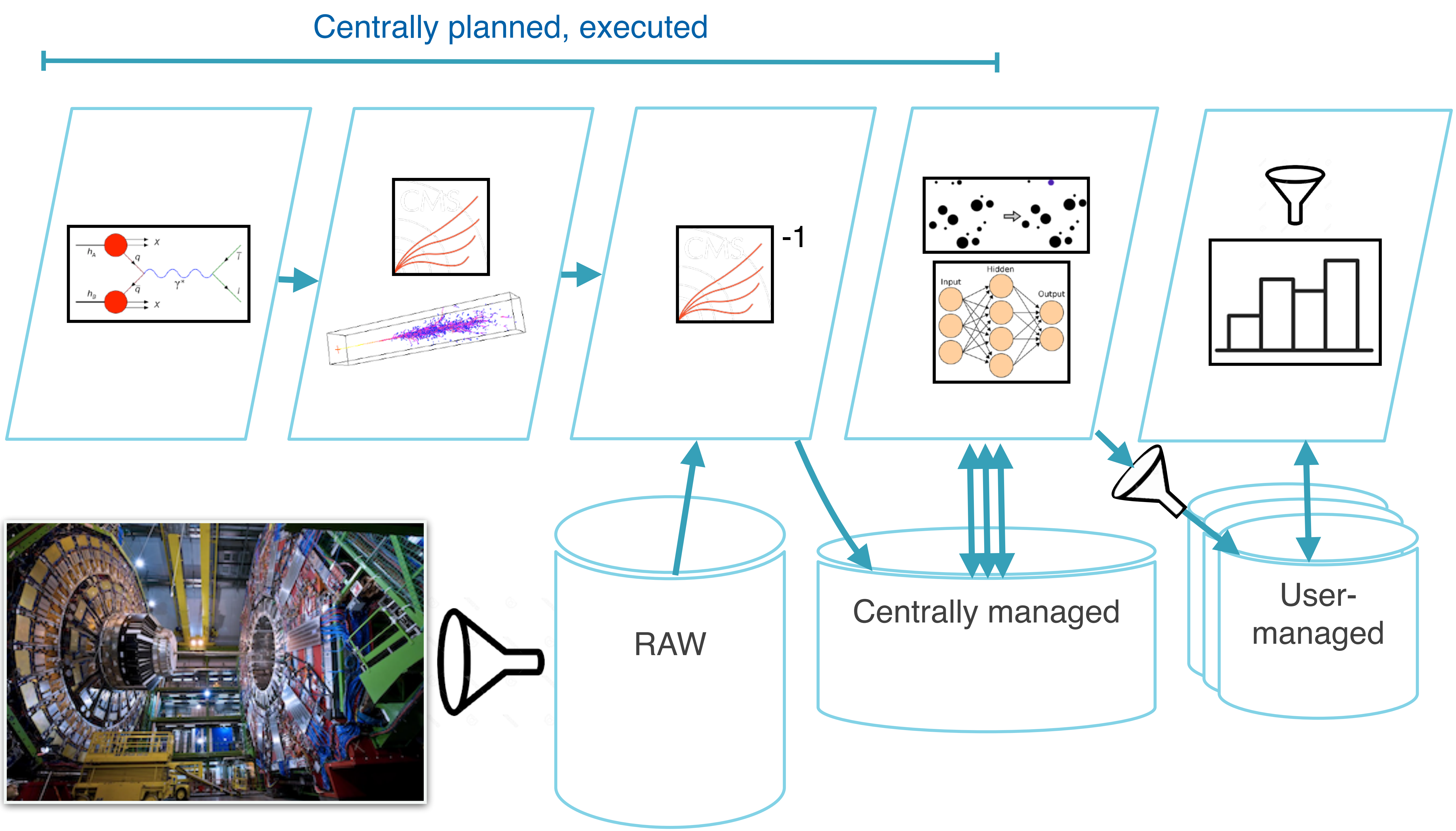


Fermilab

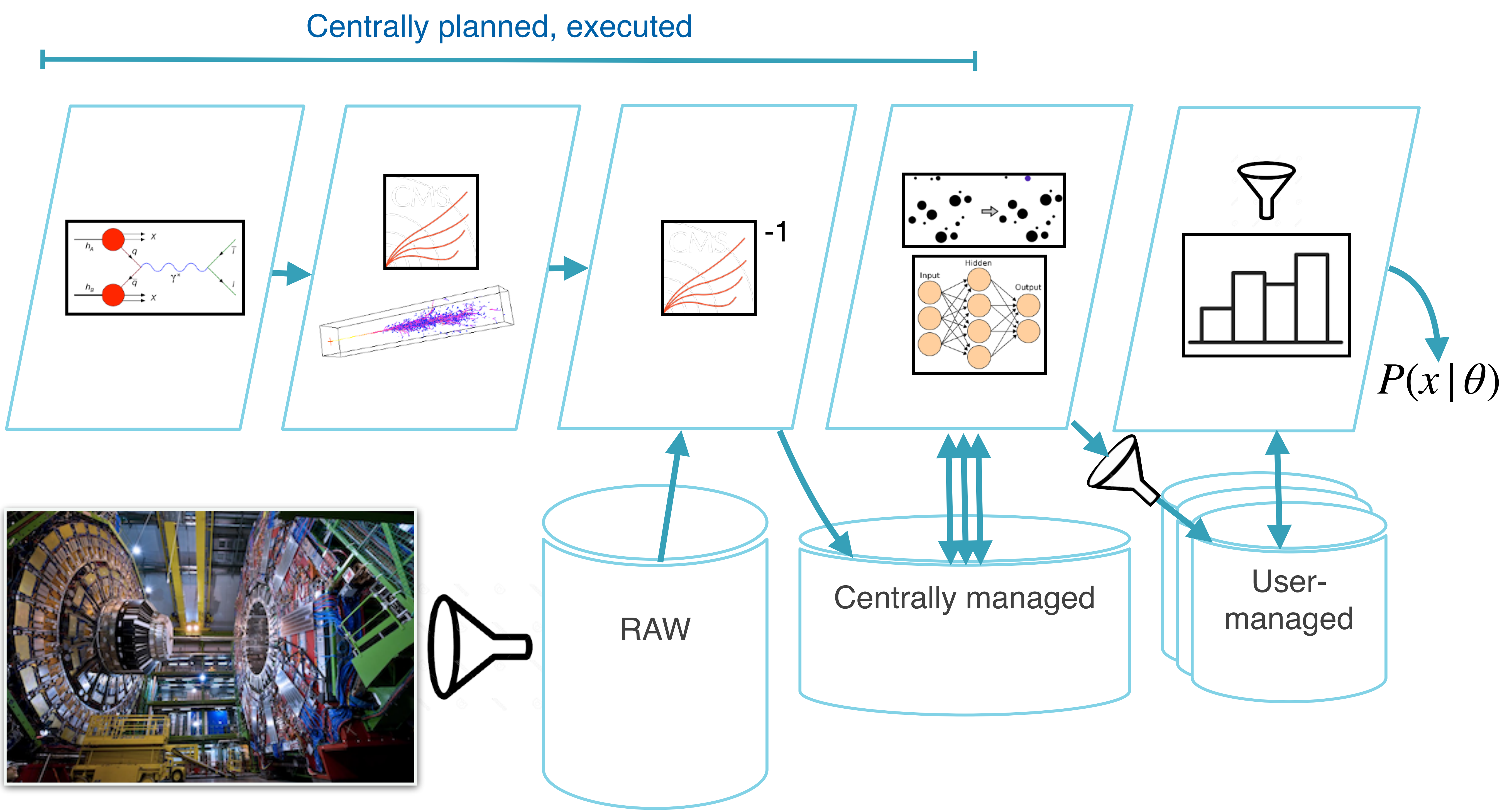
Inference: the reality



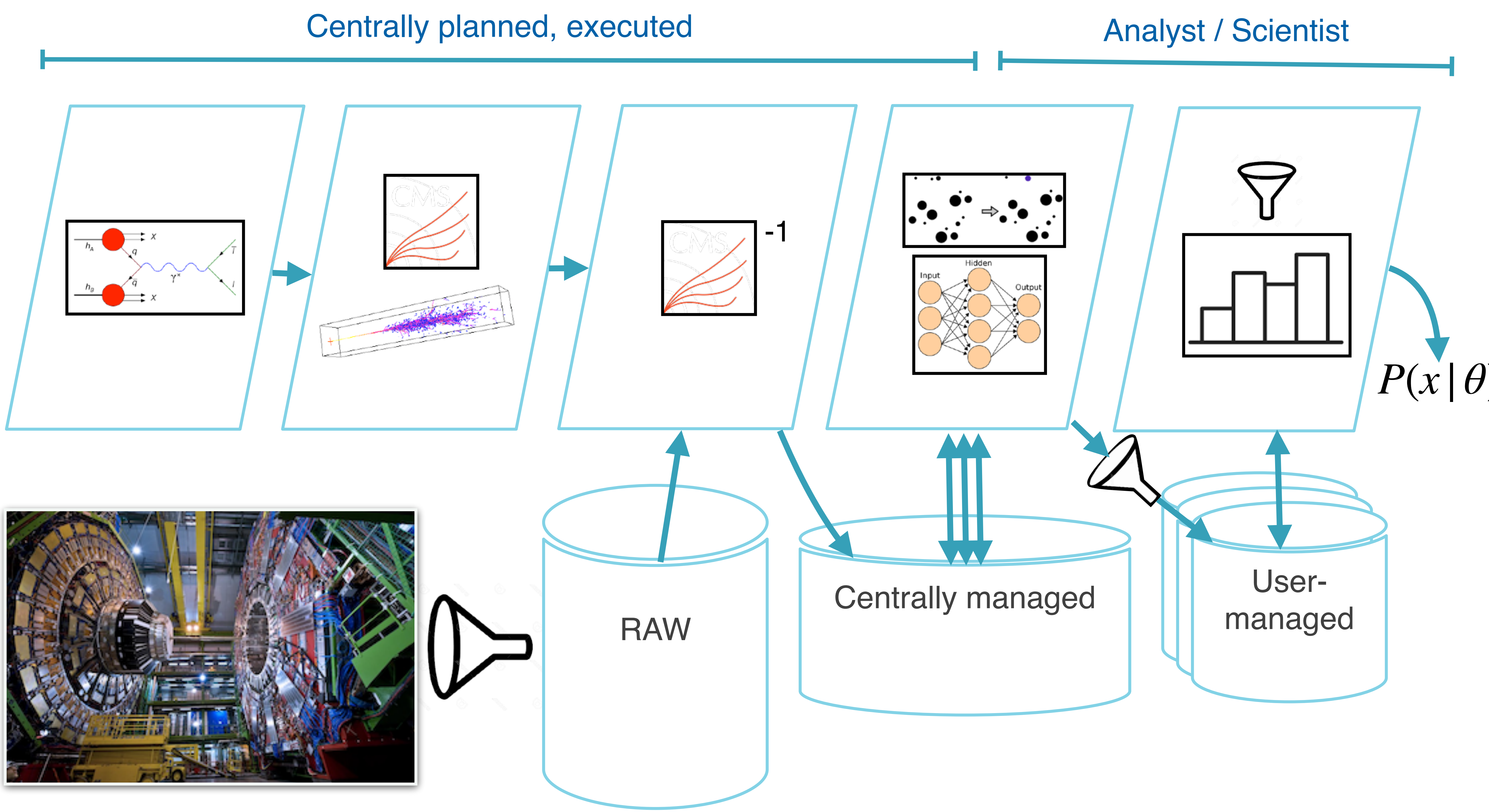
Inference: the reality



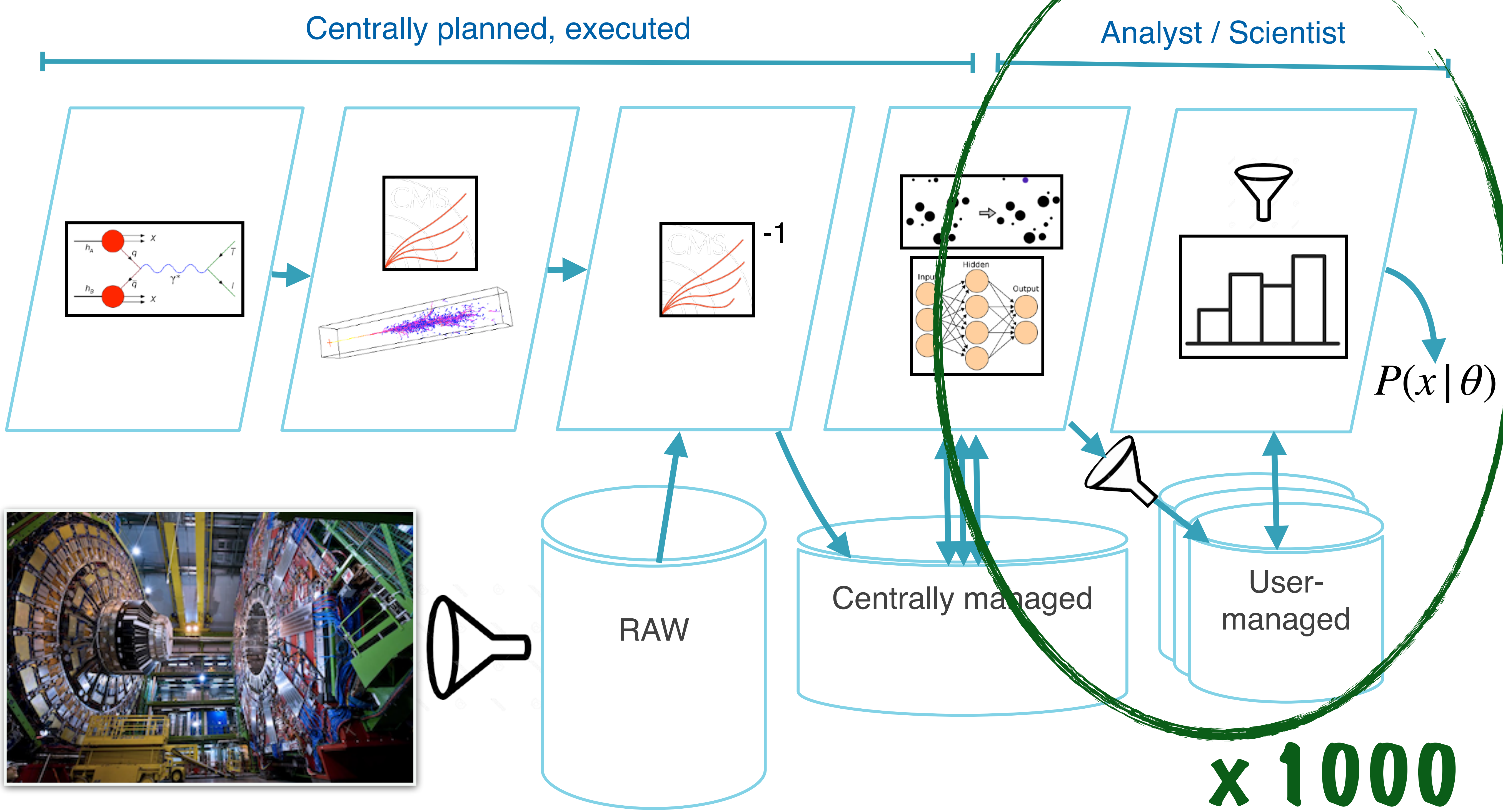
Inference: the reality



Inference: the reality

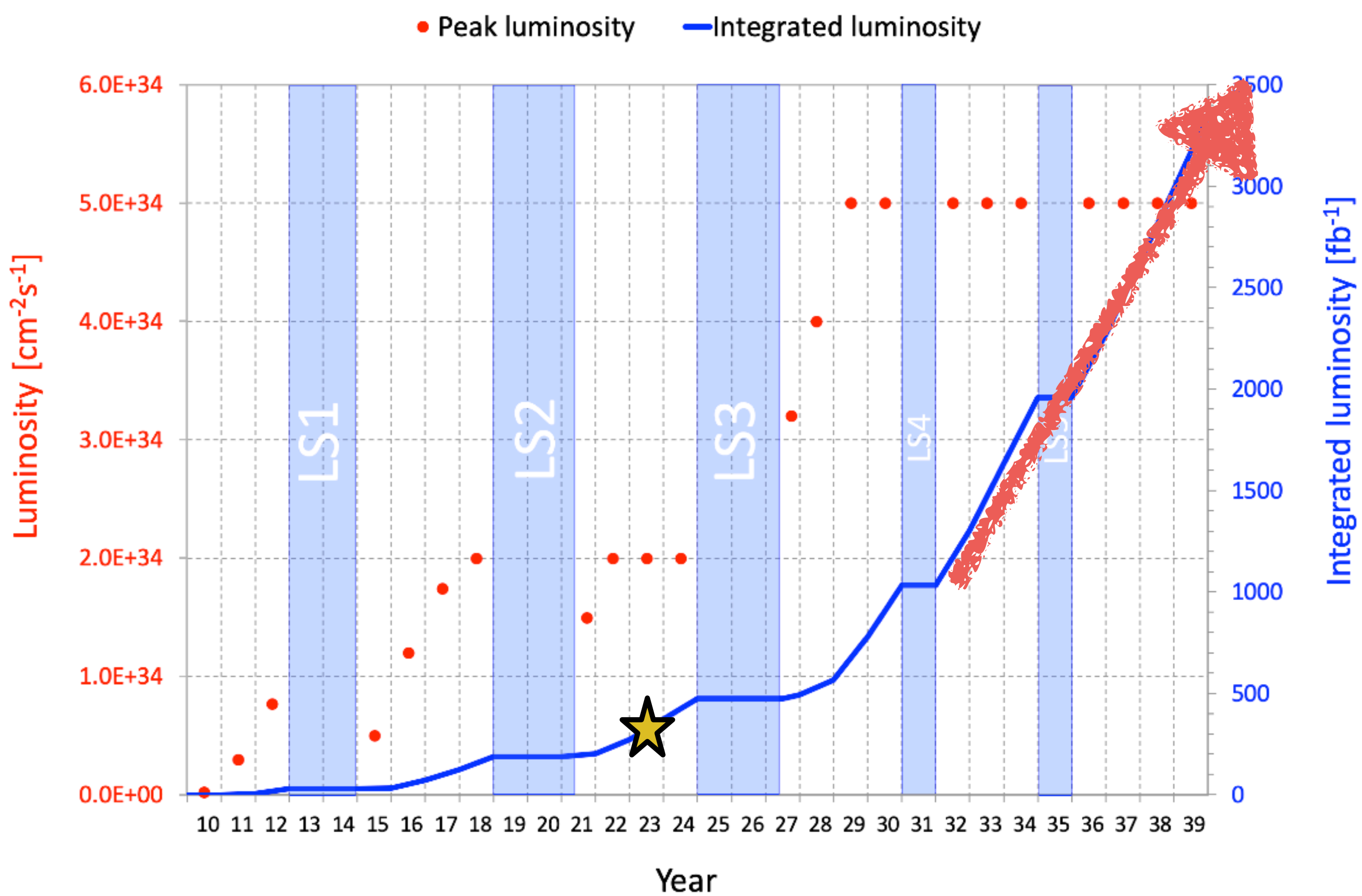


Inference: the reality

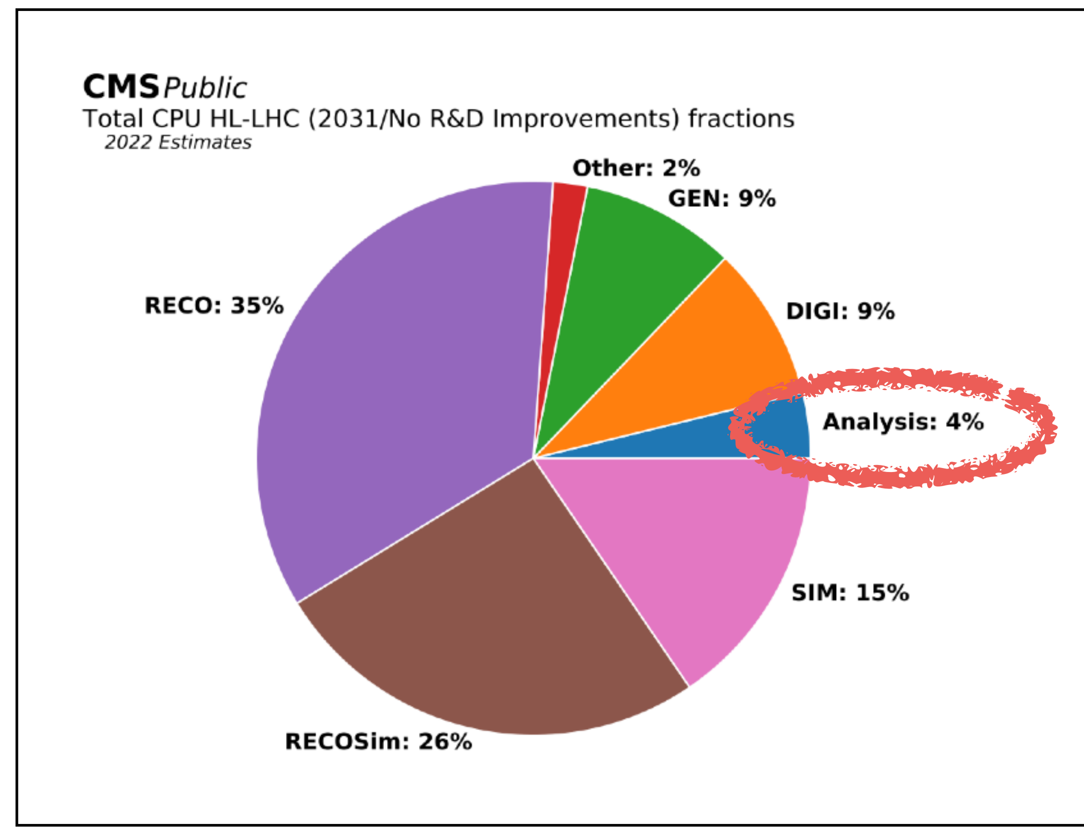


Physics - The Motivation

- The present challenge: analyze all LHC Run 2 data
  - Ensure data quality & optimize algorithms with fast time-to-insight
  - Design *increasingly complex analyses* to probe new physics signatures
- Multiply by O(1000) data analysts
- These challenges magnified 20x in HL-LHC
  - But not driving compute capacity projections



HL-LHC (pre-covid) nominal schedule



CMS Computing Projections

Fermilab

Requirements

Solutions must be:

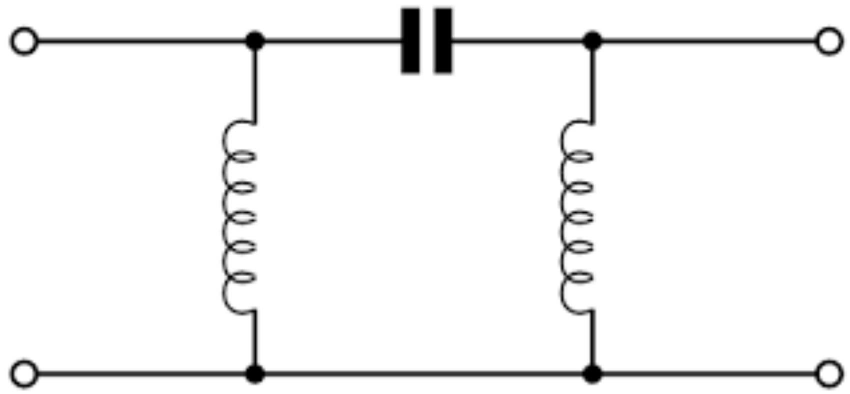
Easy to use

Scalable

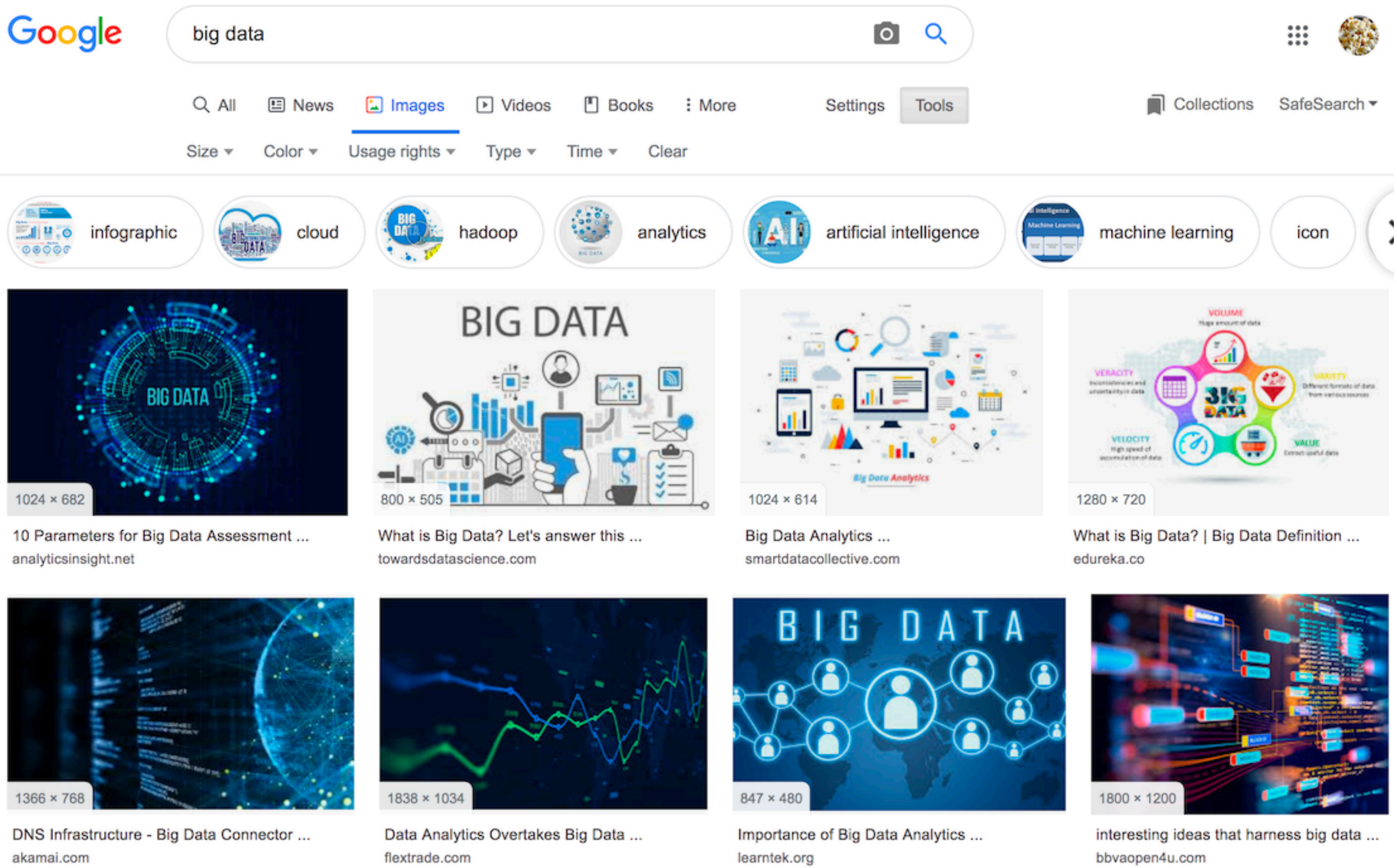
Fast

Impedance Mismatches

- ROOT File <-> Machine Learning
- Big data <-> Python
- HEP Physicist <-> Industry



Big Data


Google
big data
All
News
Images
Videos
Books
More
Settings
Tools
Collections
SafeSearch
Size
Color
Usage rights
Type
Time
Clear
infographic
cloud
hadoop
analytics
artificial intelligence
machine learning
icon
BIG DATA
1024 × 682
10 Parameters for Big Data Assessment ...
analyticsinsight.net
BIG DATA
800 × 505
What is Big Data? Let's answer this ...
towardsdatascience.com
1024 × 614
Big Data Analytics ...
smartdatacollective.com
1280 × 720
What is Big Data? | Big Data Definition ...
edureka.co
1366 × 768
DNS Infrastructure - Big Data Connector ...
akamai.com
1838 × 1034
Data Analytics Overtakes Big Data ...
flextrade.com
BIG DATA
847 × 480
Importance of Big Data Analytics ...
learntek.org
1800 × 1200
interesting ideas that harness big data ...
bbvaopen4u.com

Big Data

ML / Quant / Science
Array programming

Business Analytics
SQL-like

Big Data

ML / Quant / Science
Array programming

Business Analytics
SQL-like

Big Data

ML / Quant / Science
Array programming

Business Analytics
SQL-like

Big Data

Gorebyss

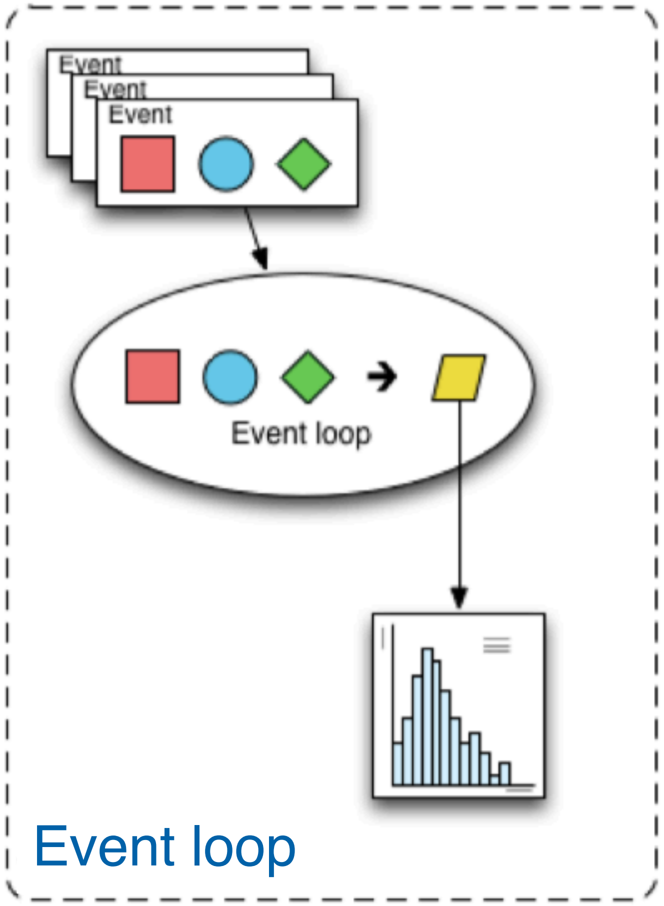
Big Data

Pokemon

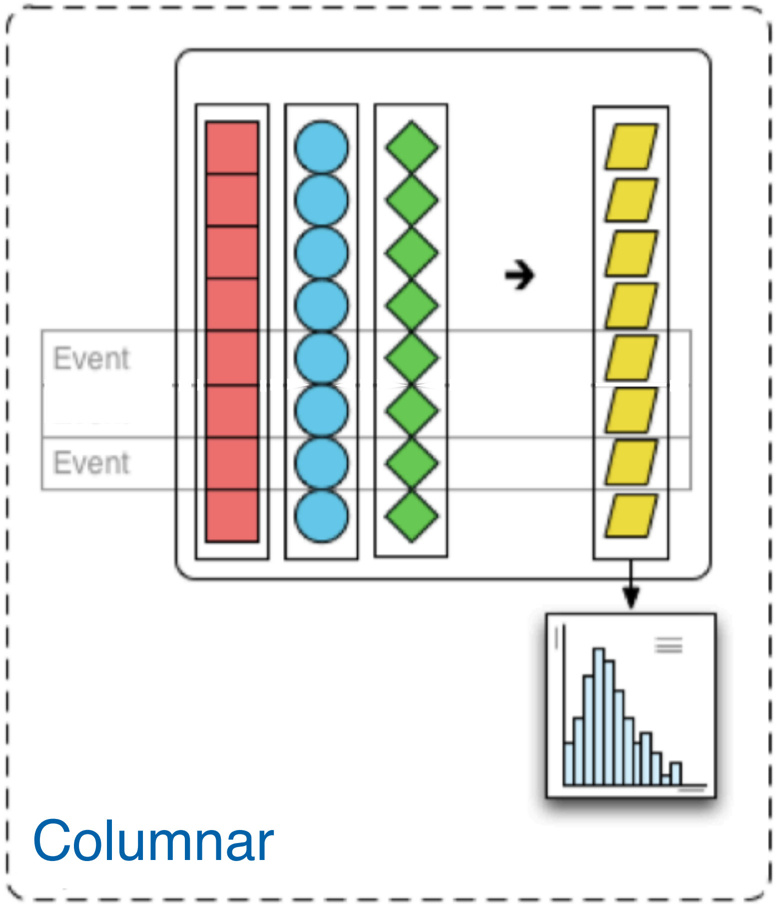
https://pixelastic.github.io/pokemonorbigdata/

The paradigm shift

- Event loop analysis:
  - Load relevant values for a specific event into local variables
  - Evaluate several expressions
  - Store derived values
  - Repeat (explicit outer loop)

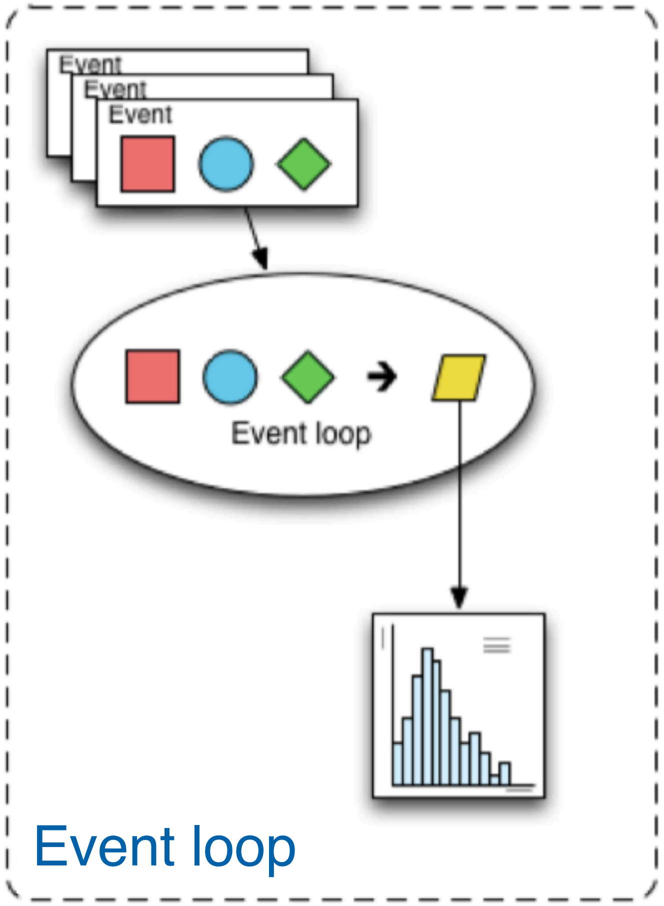


- Columnar analysis:
  - Load relevant values for many events into contiguous arrays
  - Evaluate several **array programming** expressions
    - Implicit *inner* loop**s**
  - Store derived values

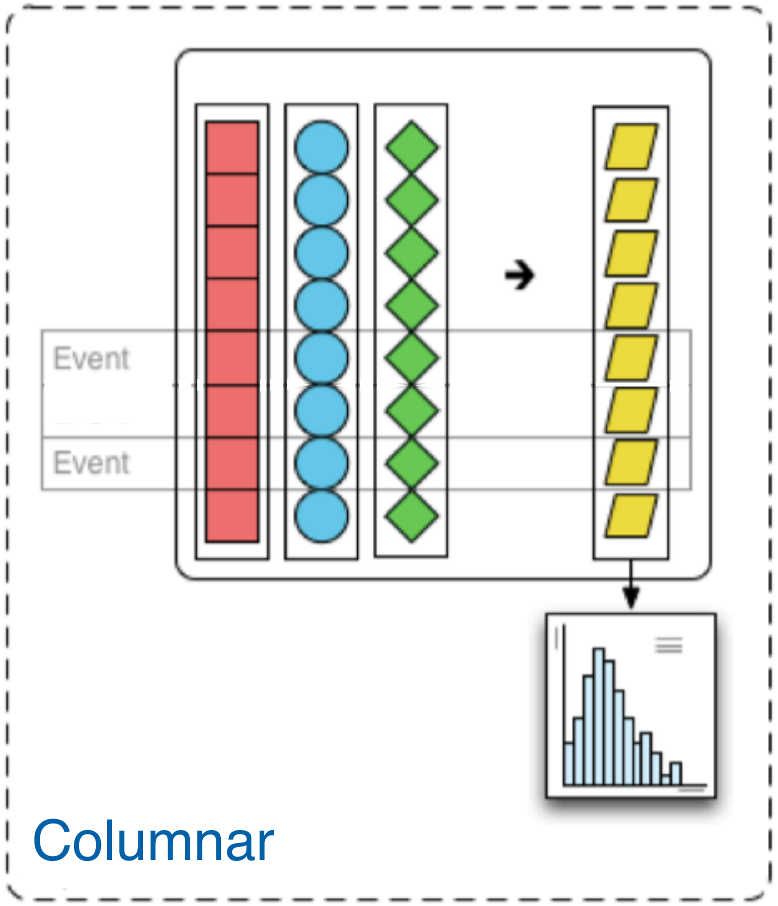


The paradigm shift

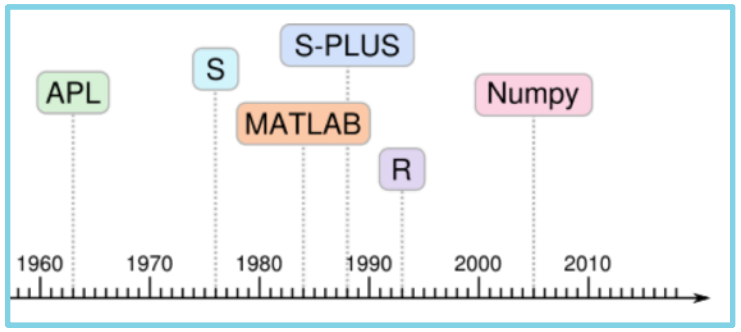
- Event loop analysis:
  - Load relevant values for a specific event into local variables
  - Evaluate several expressions
  - Store derived values
  - Repeat (explicit outer loop)



- Columnar analysis:
  - Load relevant values for many events into contiguous arrays
  - Evaluate several **array programming** expressions
    - Implicit *inner* loop**s**
  - Store derived values



Array programming is not new!
APL demo on YouTube



Awkward Array: JSON-like data, NumPy-like idioms

AWKWARD
PRINCETON UNIVERSITY iris hep
Jim Pivarski, Ianna Osborne, Peter Elmer

```python
array = ak.Array([
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [1, 2, 3, 4]}, {"x": 5.5, "y": [1, 2, 3, 4, 5]}]
])
```

```python
output = []
for sublist in python_objects:
    tmp1 = []
    for record in sublist:
        tmp2 = []
        for number in record["y"][1:]:
            tmp2.append(np.square(number))
        tmp1.append(tmp2)
    output.append(tmp1)
```

```python
output = np.square(array["y", ..., 1:])
```

```
[
    [[], [4], [4, 9]],
    [],
    [[4, 9, 16], [4, 9, 16, 25]]
]
```

2.3 minutes to run

4.6 seconds to run

(single-threaded on a 2.2 GHz processor with a dataset 10 million times larger than the one shown)

SciPy2020 awkward presentation

Fermilab

Columnar analysis - not a panacea

Event loop → **Columnar**

Event Reconstruction
1 MB/evt

Complex algorithms operating on large per-event inputs

Intra-event vectorization

→

Analysis Objects
40-400 kB/evt

Fewer complex algorithms, smaller per-event inputs

→

Filtering & Projection
(skimming & slimming)
1 kB/evt

Few complex algorithms, O(10) column inputs

→

Empirical PDFs
(histograms)
No event scaling

Trivial operations

Concrete example

```
void MyClass::Loop() {
 size_t nEvents;
 // load...

 for (Long64_t iEvent=0; iEvent<nEvents; iEvent++) {
  double MET_pt;
  int nElectron;
  double * Electron_pt;
  double * Electron_eta;
  // load...

  if ( MET_pt > 100. ) continue;

  for(size_t iEl=0; iEl<nElectron; ++iEl) {
   if ( Electron_pt[iEl] > 30. ) {
    hist->Fill(Electron_eta[iEl]);
   }
  }
 }
}
```

Event loop

Concrete example

```
void MyClass::Loop() {
  size_t nEvents;
  // load...

  for (Long64_t iEvent=0; iEvent<nEvents; iEvent++) {
    double MET_pt;
    int nElectron;
    double * Electron_pt;
    double * Electron_eta;
    // load...

    if ( MET_pt > 100. ) continue;

    for(size_t iEl=0; iEl<nElectron; ++iEl) {
      if ( Electron_pt[iEl] > 30. ) {
        hist->Fill(Electron_eta[iEl]);
      }
    }
  }
}
```

Event loop

```
void MyClass::Loop() {
  size_t nEvents;
  double * MET_pt;
  int * nElectron;
  size_t nElectron_flat;
  double * Electron_pt;
  double * Electron_eta;
  // load...

  bool * eventmask = allocate(nEvents);
  for (size_t i=0; i<nEvents; i++)
    eventmask[i] = MET_pt[i] > 100.;

  bool * entrymask = allocate(nElectron_flat);
  for (size_t i=0; i<nElectron_flat; ++i)
    entrymask[i] = Electron_pt[i] > 30.;

  bool * entrymask2 = allocate(nElectron_flat);
  size_t * parents = get_parents(nEvents, nElectron);
  for (size_t i=0; i<nElectron_flat; ++i)
    entrymask2[i] = eventmask[parents[i]] & entrymask[i];

  double * take_result = allocate(nElectron_flat);
  size_t idx = 0;
  for (size_t i=0; i<nElectron_flat; ++i)
    if ( entrymask2[i] )
      take_result[idx++] = Electron_eta[i];

  for (size_t i=0; i<idx; i++)
    hist->Fill(take_result[i]);
}
```

Columnar

Concrete example

```
void MyClass::Loop() {
 size_t nEvents;
 // load...

 for (Long64_t iEvent=0; iEvent<nEvents; iEvent++) {
  double MET_pt;
  int nElectron;
  double * Electron_pt;
  double * Electron_eta;
  // load...

  if ( MET_pt > 100. ) continue;

  for(size_t iEl=0; iEl<nElectron; ++iEl) {
   if ( Electron_pt[iEl] > 30. ) {
    hist->Fill(Electron_eta[iEl]);
   }
  }
 }
}
```

Event loop

```
cut = (events.MET.pt < 100.) & (events.Electron.pt > 30.)
hist.fill(eta=ak.flatten(events.Electron.eta[cut]))
```

Columnar

Concrete example

```
void MyClass::Loop() {
 size_t nEvents;
 // load...

 for (Long64_t iEvent=0; iEvent<nEvents; iEvent++) {
  double MET_pt;
  int nElectron;
  double * Electron_pt;
  double * Electron_eta;
  // load...

  if ( MET_pt > 100. ) continue;

  for(size_t iEl=0; iEl<nElectron; ++iEl) {
   if ( Electron_pt[iEl] > 30. ) {
    hist->Fill(Electron_eta[iEl]);
   }
  }
 }
}
```

Event loop

- Human time is most expensive
- Improve design *and* performance

```
cut = (events.MET.pt < 100.) & (events.Electron.pt > 30.)
hist.fill(eta=ak.flatten(events.Electron.eta[cut]))
```

Columnar

Concrete example

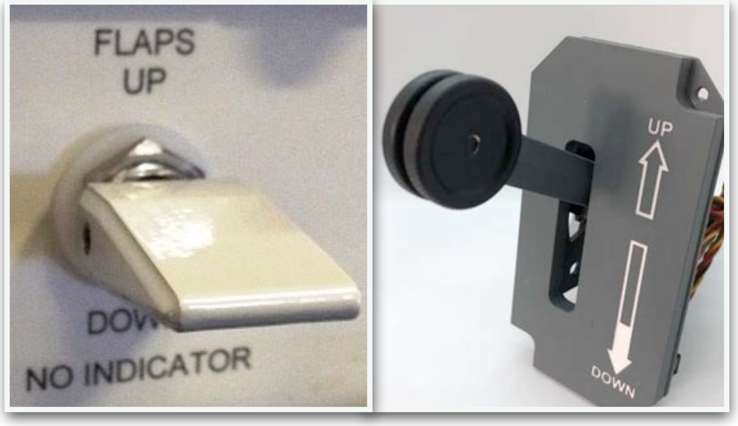
```
void MyClass::Loop() {
  size_t nEvents;
  // load...

  for (Long64_t iEvent=0; iEvent<nEvents; iEvent++) {
    double MET_pt;
    int nElectron;
    double * Electron_pt;
    double * Electron_eta;
    // load...

    if ( MET_pt > 100. ) continue;

    for(size_t iEl=0; iEl<nElectron; ++iEl) {
      if ( Electron_pt[iEl] > 30. ) {
        hist->Fill(Electron_eta[iEl]);
      }
    }
  }
}
```

Event loop

- Human time is most expensive
- Improve design *and* performance

```
cut = (events.MET.pt < 100.) & (events.Electron.pt > 30.)
hist.fill(eta=ak.flatten(events.Electron.eta[cut]))
```

Columnar



Concrete example

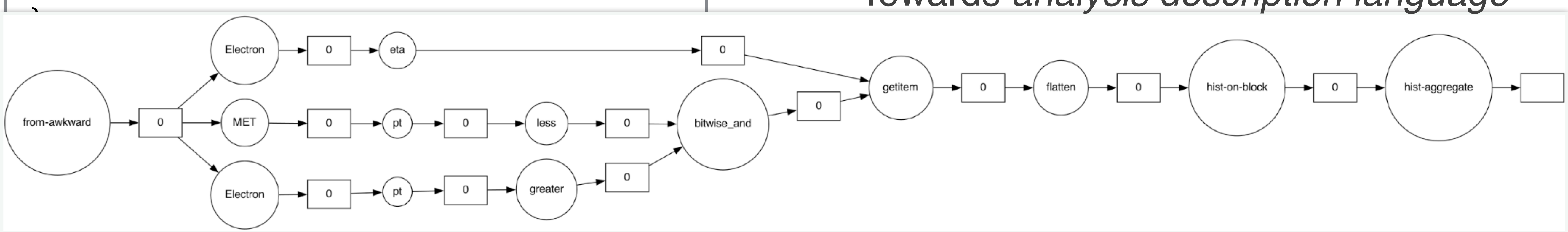
```
void MyClass::Loop() {
 size_t nEvents;
 // load...

 for (Long64_t iEvent=0; iEvent<nEvents; iEvent++) {
  double MET_pt;
  int nElectron;
  double * Electron_pt;
  double * Electron_eta;
  // load...

  if ( MET_pt > 100. ) continue;

  for(size_t iEl=0; iEl<nElectron; ++iEl) {
   if ( Electron_pt[iEl] > 30. ) {
    hist->Fill(Electron_eta[iEl]);
   }
  }
```

- Human time is most expensive
- Improve design *and* performance

```
cut = (events.MET.pt < 100.) & (events.Electron.pt > 30.)
hist.fill(eta=ak.flatten(events.Electron.eta[cut]))
```

Columnar

- Analysis captured in *task graph*
  - Automated transformations possible
  - Towards *analysis description language*



Convergent evolution

- ROOT also migrating to implicit loops & task graph construction
  - RDataFrame accepts pre-defined & user-defined C++ primitives
- Interoperability is on everyone's mind
  - cppyy automatic Python bindings
  - RDataFrame views into awkward-array & vice-versa
  - UHI Plottable protocol

Select and fill: fully compiled C++ code

```
RVecD selectPt(RVecD &pt, RVecD &eta) { return pt[eta > 0]; }

auto h = RDataFrame("tree", "f.root")
            .Define("pt", selectPt, {"muon_pt", "muon_eta"})
            .Histo1D<RVecD>("pt");
h->Draw();
```

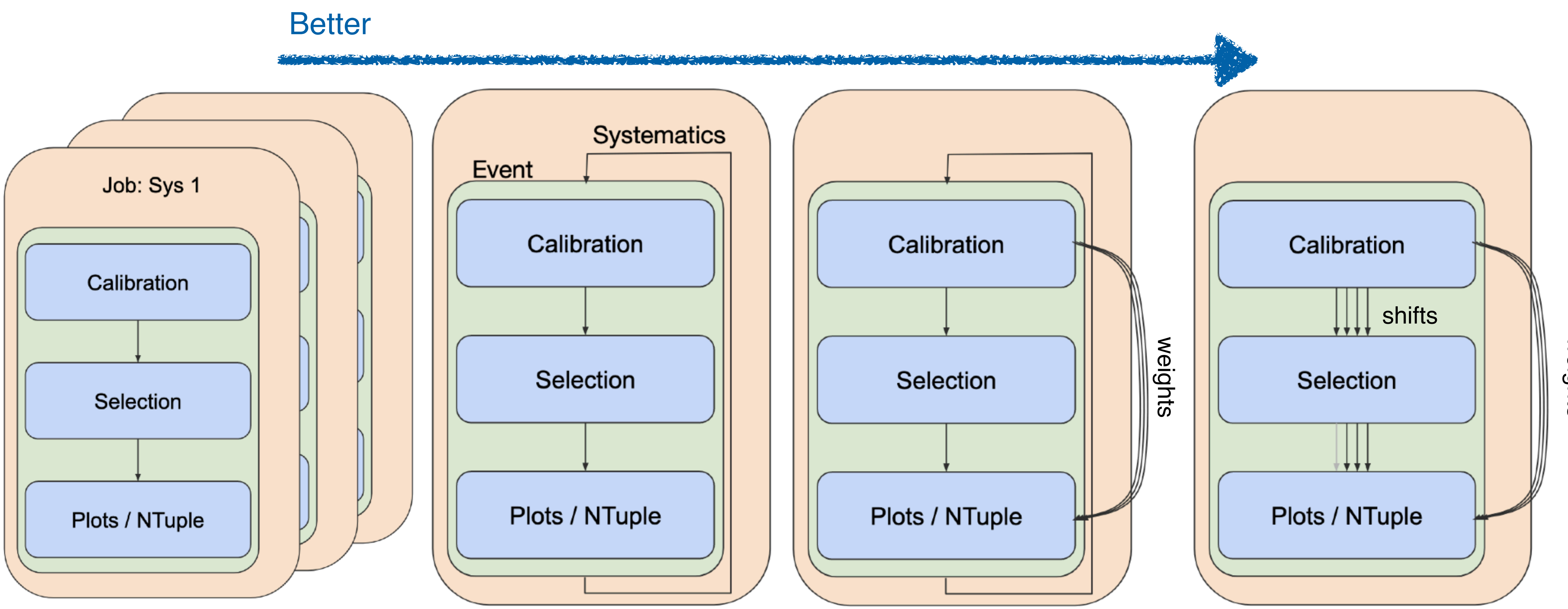
E. Guiraud

```
array = ak._v2.Array([
    [{"x": 1, "y": [1.1]}, {"x": 2, "y": [2.2, 0.2]}],
    [],
    [{"x": 3, "y": [3.0, 0.3, 3.3]}]])

ak_array_1 = array["x"]
ak_array_2 = array["y"]

data_frame = ak._v2.to_rdataframe(
    {"x": ak_array_1, "y": ak_array_2}
)
h = data_frame.Histo1D("x")
h.Draw()
```

I. Osborne

Task graphs enable automated systematics

- Simplest solution: re-run everything with alternate calibration/selection/weight
- Better: loop over event while in-memory (likely CPU cache)
  - Why? Because IO is very expensive
- Best: compute all weights, compute calibration shifts only as necessary



S. Hageboeck

Analysis task graph transformation in ROOT

- RDataFrame::Vary simplifies implementation!
  - Coffea as well now that dask-awkward is out (shown at CHEP2023!)
    - Current: embed lazy-evaluated systematics in NanoEvents

Each __getattr__ initiates evaluation

```
events.jets.JES_jes.up.pt
```

In ROOT 6.26 (experimental)

Python

attach an up/down variation to “pt”

proceed as usual, as if working with nominal values only

obtain all variations

```python
nominal_hx =
  df.Vary("pt", "RVecD{pt*0.9, pt*1.1}", ["down", "up"])
  .Filter("pt > k")
  .Define("x", someFunc, ["pt"])
  .Histo1D("x")

hx = ROOT.RDF.VariationsFor(nominal_hx)
hx["nominal"].Draw()
hx["pt:down"].Draw("SAME")
```

E. Guiraud

Fermilab

Analysis description language?

How can I extract Parton distribution function weights using the coffea data analysis framework?

Coffea is a data analysis framework for high energy physics that provides a Python interface to perform event-level data analysis. To extract Parton Distribution Function (PDF) weights using Coffea, you can use the built-in functionality to read in the events from a Monte Carlo simulation that contains PDF weight information.

Here is an example of how you might extract PDF weights using Coffea:

1. First, you'll need to import the necessary modules and define the input file path:

Copy code

```
from coffea import hist
from coffea.analysis_objects import JaggedCandidateArray

file_path = '/path/to/input/file.root'
```

2. Next, you'll need to define the dataset and read in the events:

Copy code

```
dataset = {'file': [file_path]}
events = JaggedCandidateArray.candidatesfrom_tree(dataset, tree='Events')
```

3. To extract the PDF weights, you can access the **`pdf_weights`** attribute of the events. This attribute is a nested array that contains the weights for each event and each PDF set:

Copy code

```
pdf_weights = events.pdf_weights.content
```

4. You can then use this array to create histograms of the PDF weights, or use them for further analysis:

Copy code

```
pdf_weights_hist = hist.Hist("Counts", hist.Cat("dataset", "Dataset"),
hist.Bin("pdf_weight", "PDF weight", 50, 0, 1))
pdf_weights_hist.fill(dataset=dataset, pdf_weight=pdf_weights)
```

This is a basic example of how to extract PDF weights using Coffea, and you may need to adjust it to suit your specific analysis needs.

Please note that this is a simplified example and in a real analysis you should use the appropriate pre-processing, cuts, selections and other functionalities provided by the Coffea framework to extract meaningful physics results.

More seriously, see e.g. ADL workshop 2019

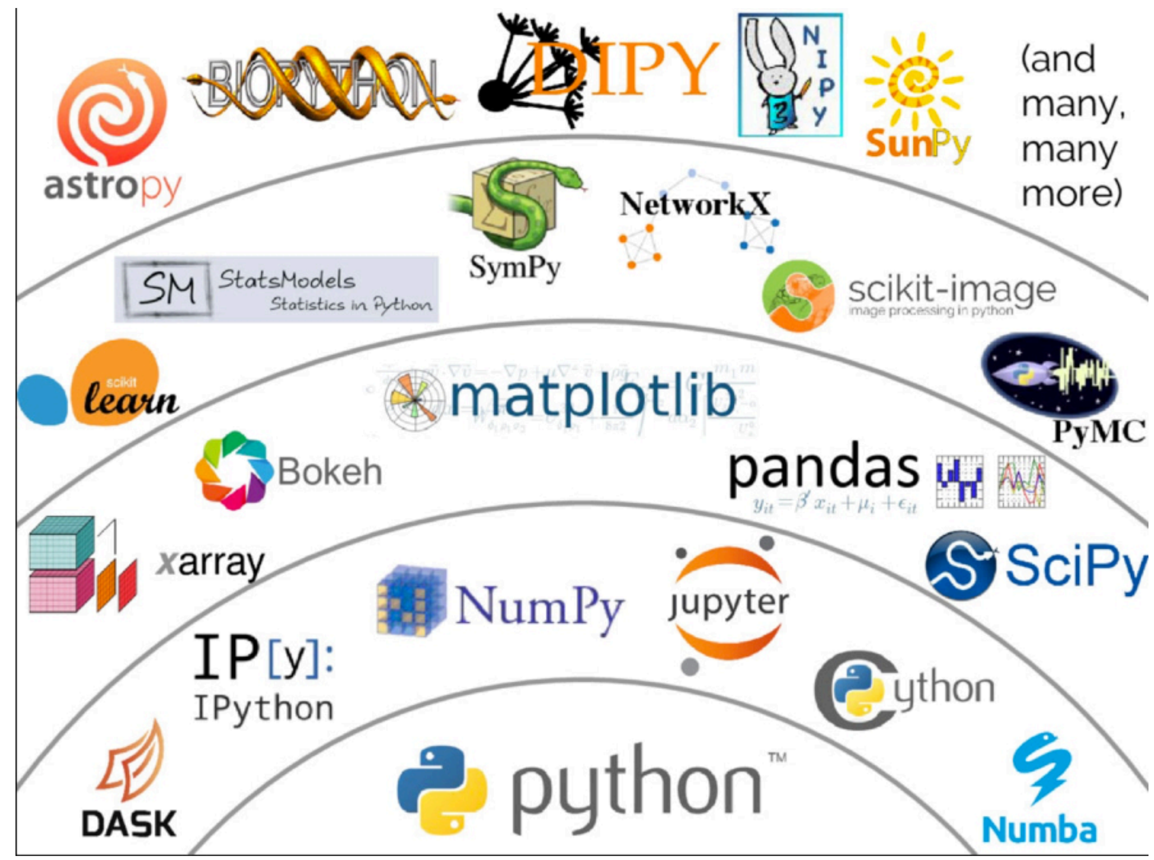
The other paradigm shift

- From (more) vertically-integrated C++ ecosystem to Scientific Python

Experiment-specific and analysis frameworks

ROOT Data Analysis Framework

https://root.cern.ch/

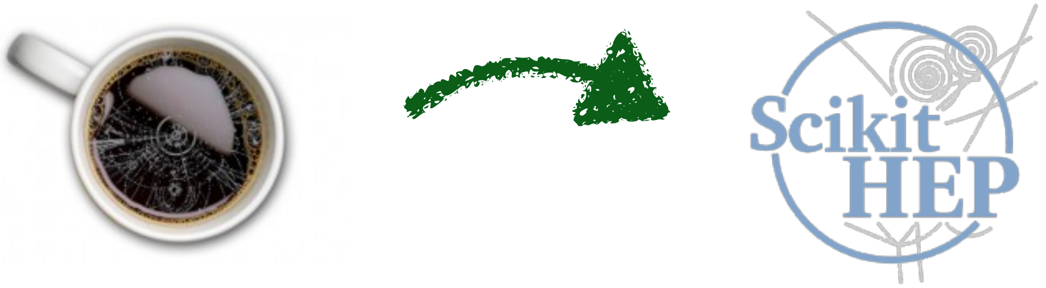


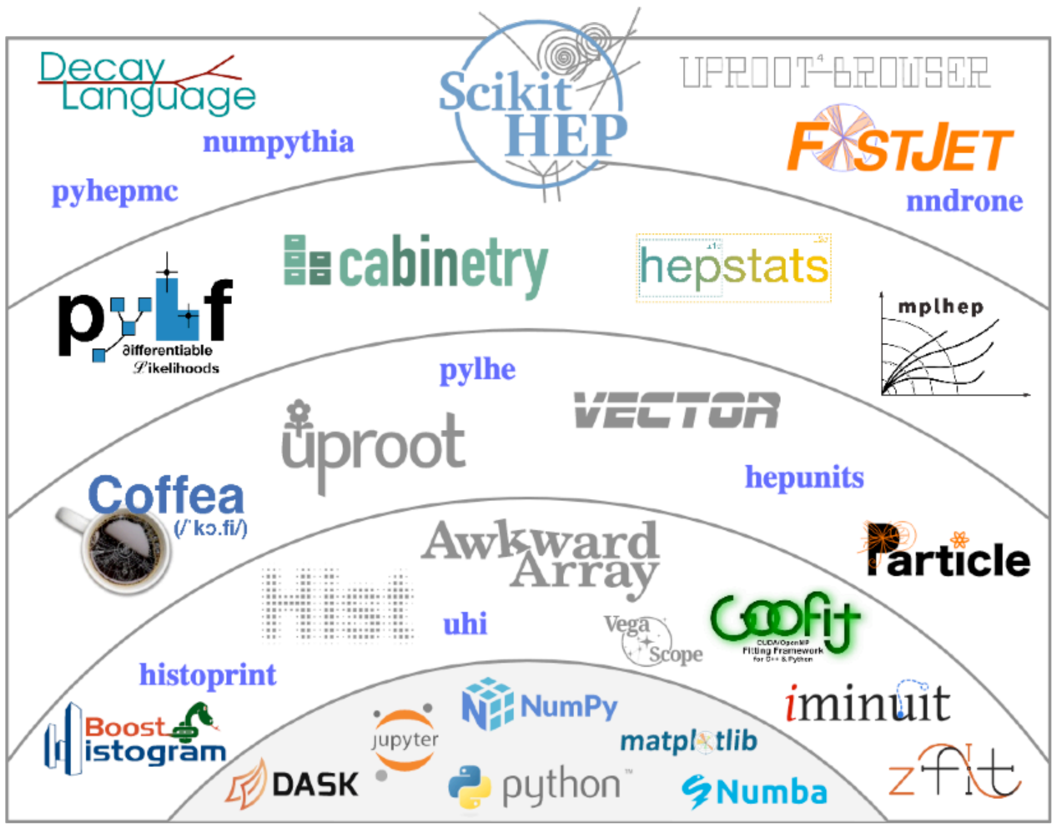
Scientific Python Ecosystem

Fermilab

Coffea project

- A user interface to *columnar analysis*
  - Optimized array programming kernels build an **expressive *and* performant** language
  - Seamless integration with ML tools due to shared interface

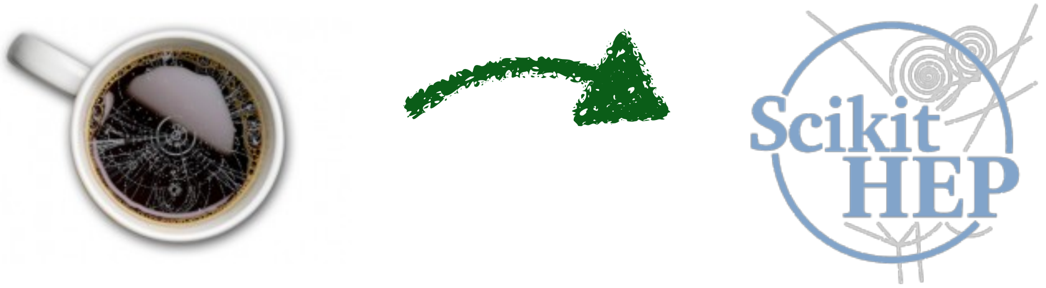


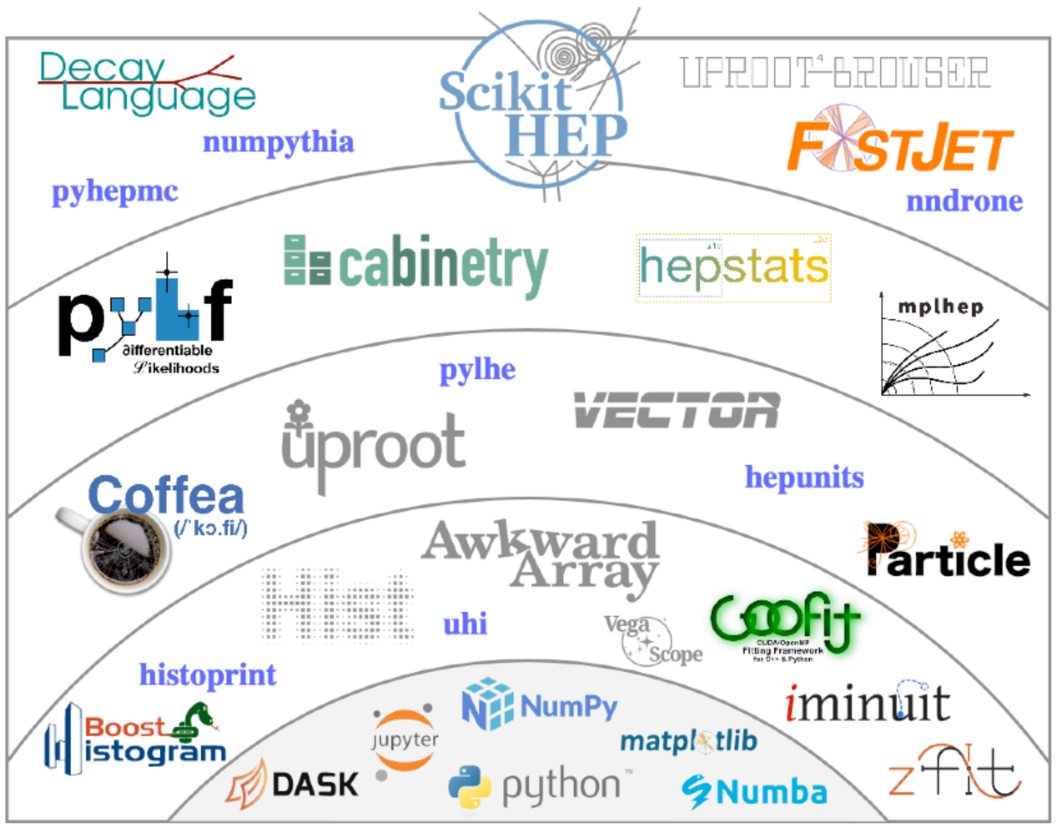


Fermilab

Coffea project

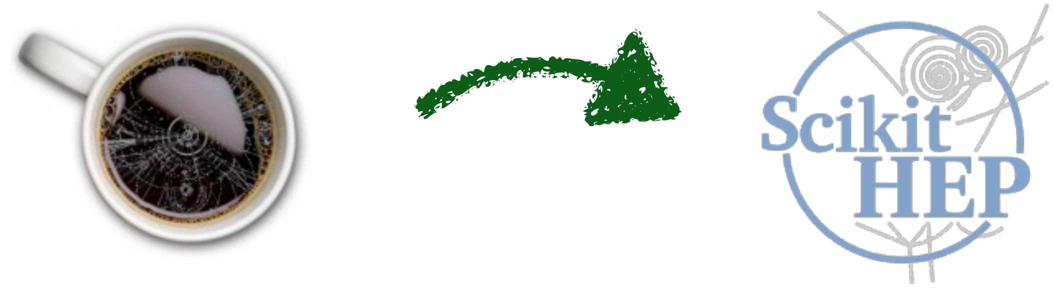
- A user interface to *columnar analysis*
  - Optimized array programming kernels build an **expressive *and* performant** language
  - Seamless integration with ML tools due to shared interface
- An incubator for rapid prototyping
  - Fill in missing pieces of ecosystem
  - Good abstractions are factored out

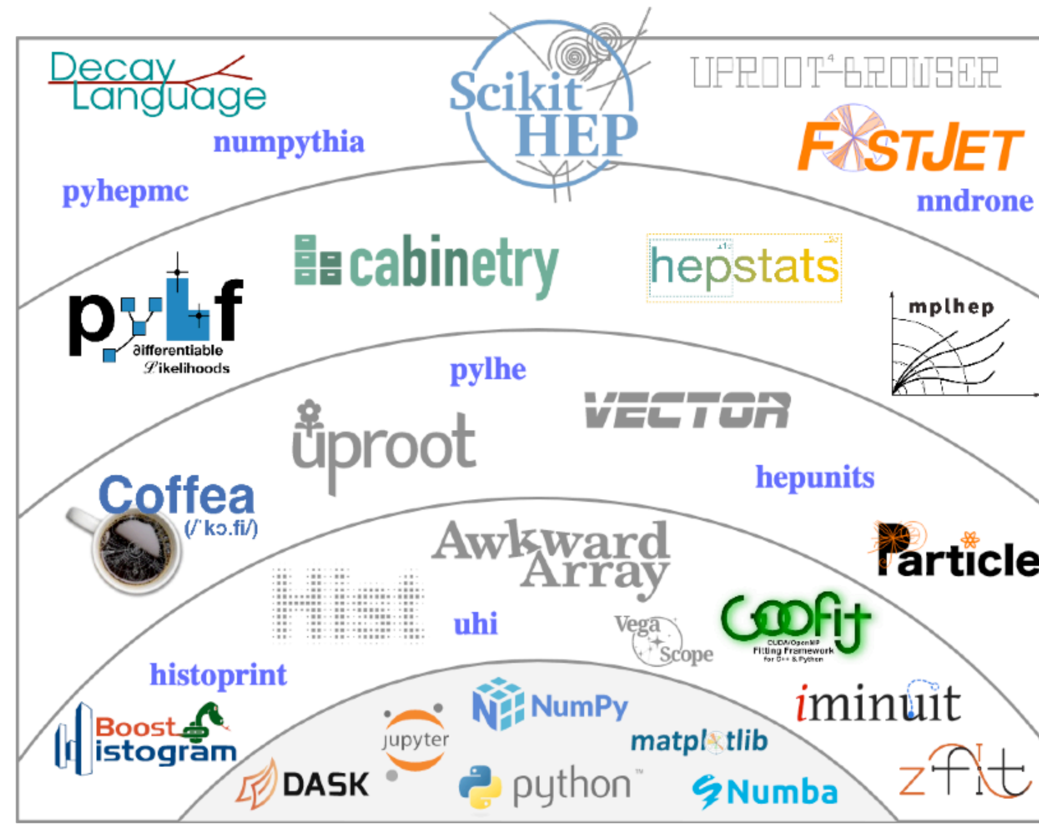


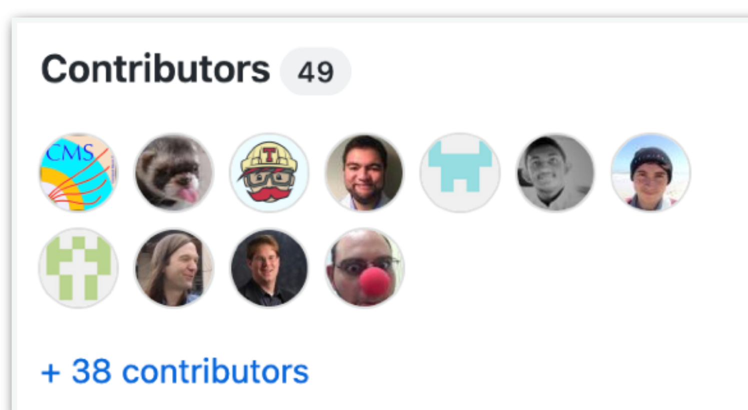


Coffea project

- A user interface to *columnar analysis*
  - Optimized array programming kernels build an **expressive *and* performant** language
  - Seamless integration with ML tools due to shared interface
- An incubator for rapid prototyping
  - Fill in missing pieces of ecosystem
  - Good abstractions are factored out
- A minimum viable product
  - Already used in several CMS publications
  - In use by ATLAS, ProtoDUNE collaborators
  - Early feedback builds ecosystem roadmap
    - Vibrant contributor community

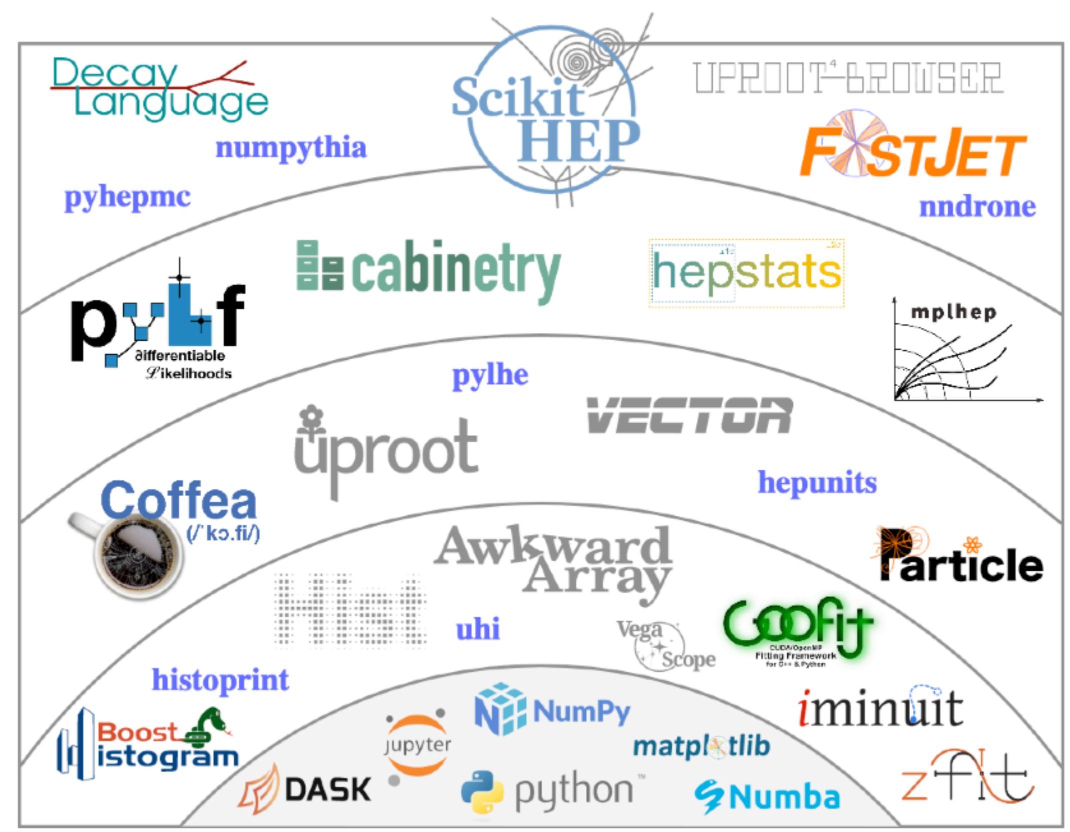






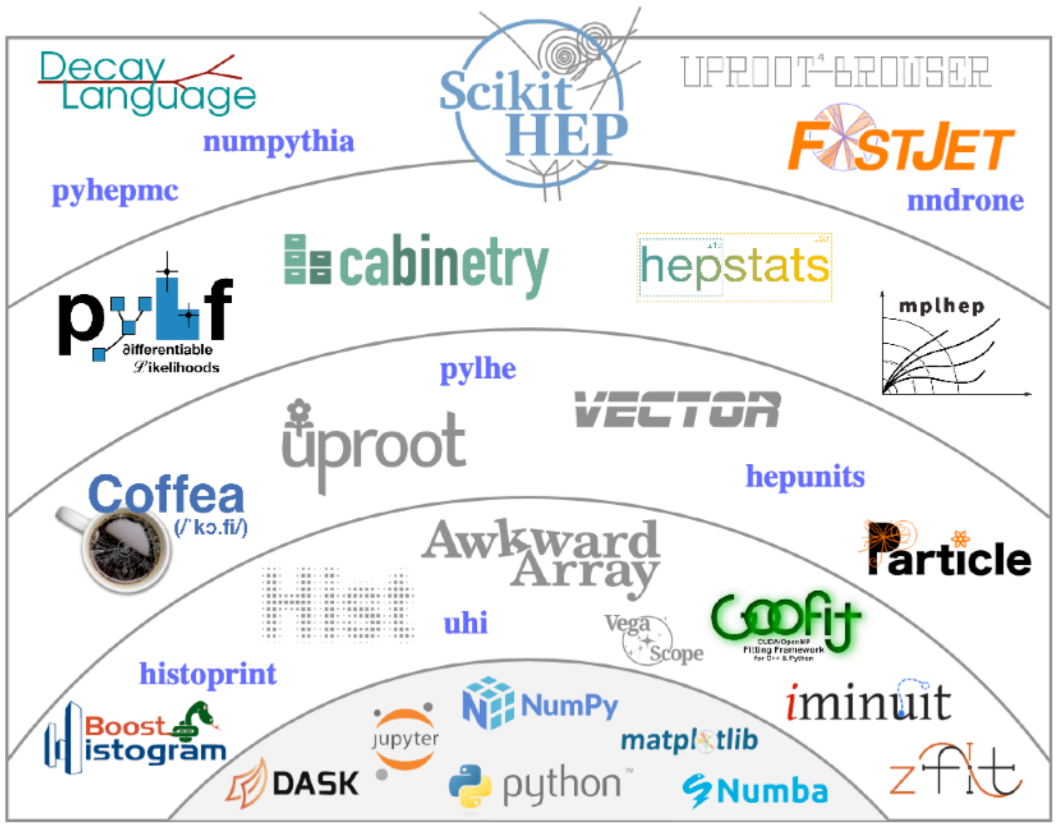
Scikit-HEP: a young ecosystem

- As the community grows, new array-oriented interfaces are built
  - boost-histogram + hist (after dozens of competitors)
  - vector for Lorentz math
  - mplhep: HEP experiment plot styles in matplotlib
  - iMinuit: classic minimization library (scipy.optimize has many modern options)
  - fastjet: re-cluster awkward arrays of Lorentz vectors
  - And more…



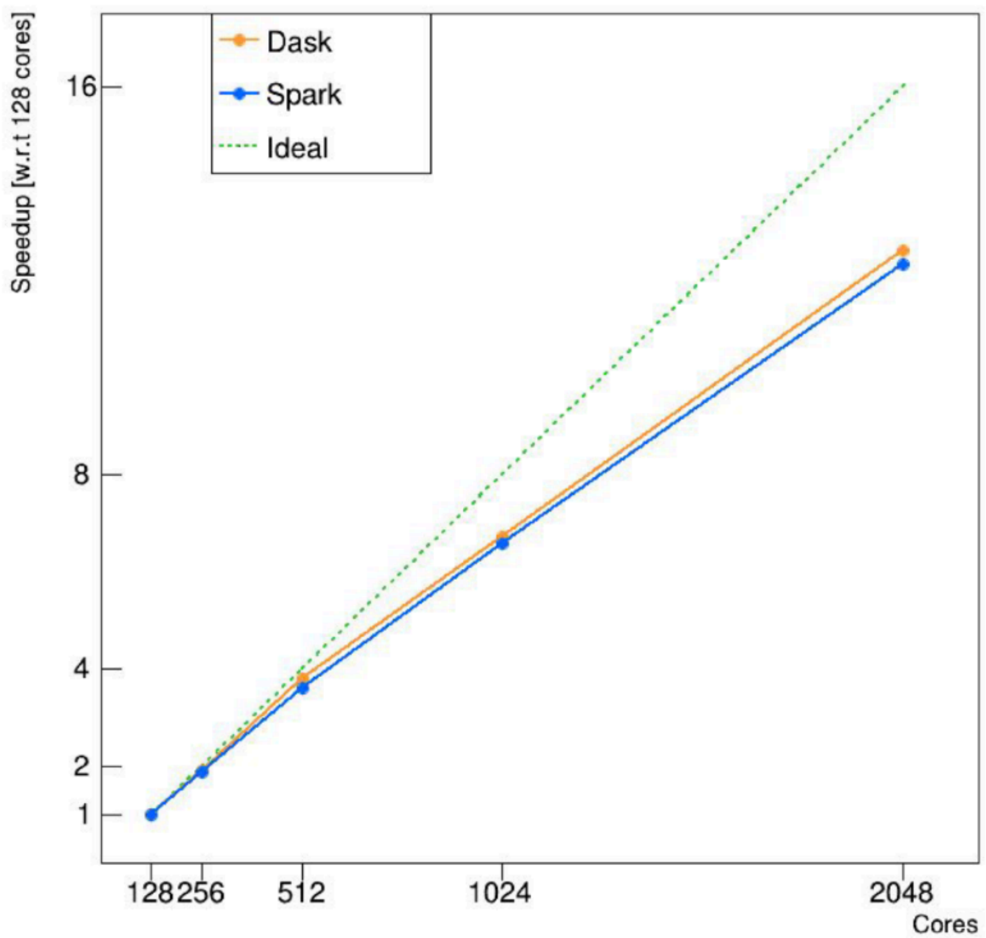
Scikit-HEP: a young ecosystem

- As the community grows, new array-oriented interfaces are built
  - boost-histogram + hist (after dozens of competitors)
  - vector for Lorentz math
  - mplhep: HEP experiment plot styles in matplotlib
  - iMinuit: classic minimization library (scipy.optimize has many modern options)
  - fastjet: re-cluster awkward arrays of Lorentz vectors
  - And more…
- Common tools reduce development burden
  - e.g. parton: PDF weight evaluation
    - Uses an off-the-shelf interpolator (same as LHAPDF)
    - Just needed vectorized evaluation
  - Transparent *contribution* path important

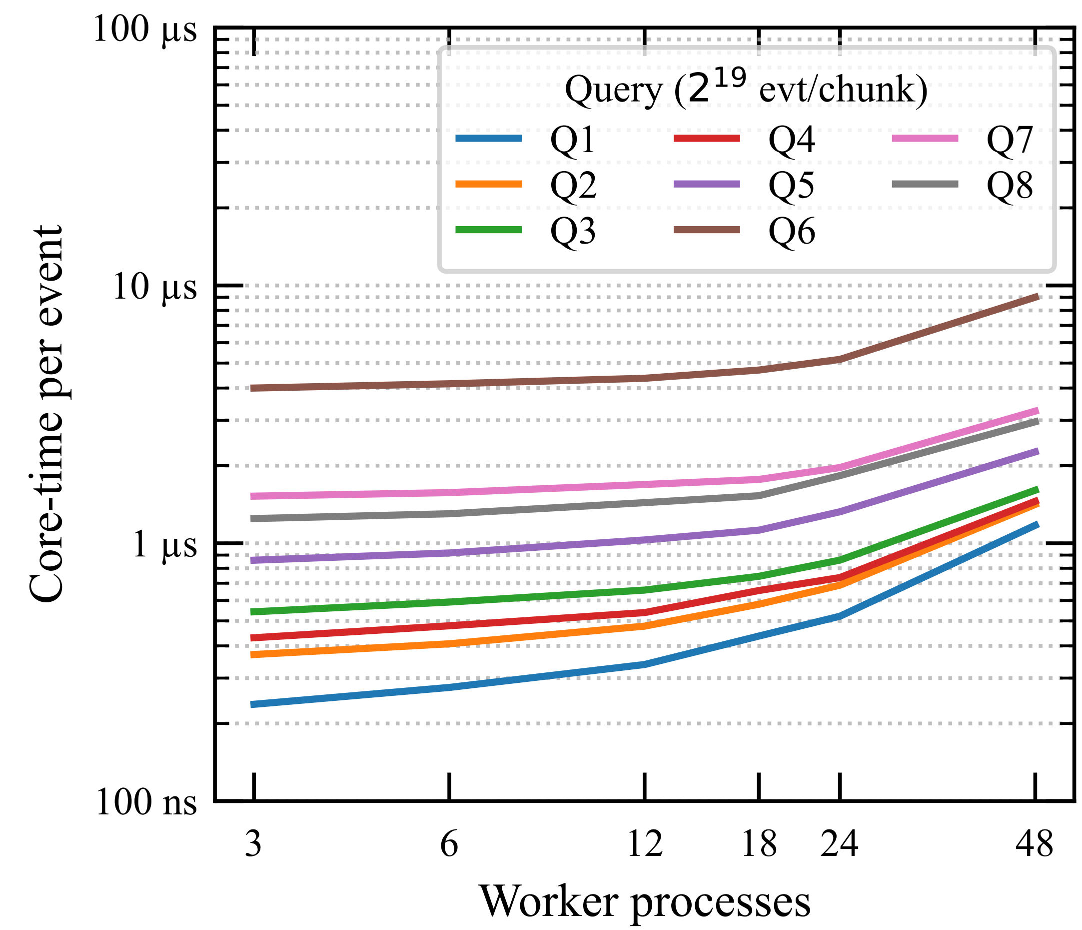


Are our solutions scalable?

- More than just thread/core scaling
  - But good to check



[E. Guiraud](#)



NS, [ACAT2021](#)

Scalability requirements

- Easy **transition** from local to distributed execution
  - Ideally no user code change
- Challenge: user library code not always easy to distribute
  - Shared filesystems are a luxury for some facilities
  - Further discussion in [CoffeaTeam/coffea#511]

```python
from coffea import nanoevents, processor


if __name__ == "__main__":
    runner = processor.Runner(
        executor=processor.FuturesExecutor(workers=4),
        schema=nanoevents.NanoAODSchema,
    )

    output = runner(
        fileset={"SingleMu": ["Run2012B_SingleMu.root"]},
        treename="Events",
        processor_instance=MyProcessor(),
    )
```

```python
from coffea import nanoevents, processor
from distributed import Client


if __name__ == "__main__":
    runner = processor.Runner(
        executor=processor.DaskExecutor(client=Client()),
        schema=nanoevents.NanoAODSchema,
    )

    output = runner(
        fileset={"SingleMu": ["Run2012B_SingleMu.root"]},
        treename="Events",
        processor_instance=MyProcessor(),
    )
```

Scalability requirements

- Easy **transition** from local to distributed execution
  - Ideally no user code change
- Challenge: user library code not always easy to distribute
  - Shared filesystems are a luxury for some facilities
  - Further discussion in [CoffeaTeam/coffea#511]()

Local	Distributed

Importing RDataFrame

Local:

```python
from ROOT import RDataFrame
```

Distributed:

```python
import ROOT
RDataFrame = \
ROOT.RDF.Experimental.Distributed.Dask.RDataFrame
```

Constructing RDataFrame

Local:

```python
df = RDataFrame('treename', 'filename.root')
```

Distributed:

```python
from dask.distributed import Client
df = RDataFrame('treename', 'filename.root',
     daskclient = Client('tcp://hostname:port'))
```

Rest of application

```python
df2 = df.Filter(...).Define(...)
h1 = df2.Histo1D(...)
h1.Draw()
```

V. Padulano

Scalability solutions

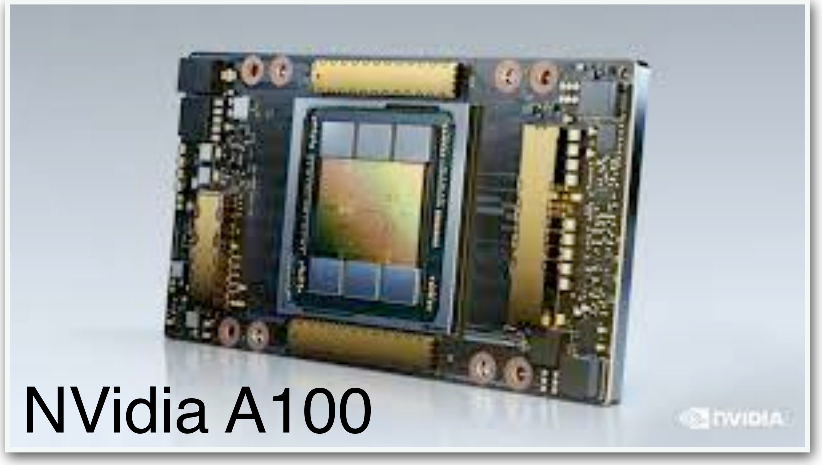
- We have so many options!
  - Dask
    - Large community, excellent numpy integration, dask-awkward in development
  - Apache Spark
    - Scala with python bindings, *very large* community
  - JobLib
    - Common in scikit-learn community, targeting CPU-bound tasks
  - Celery
    - Generic task queue leveraging modern distributed foundations: rabbitMQ, redis, etc.
  - Ray
    - Multi-scheduler design, more focused on distributed ML tasks
  - Parsl, WorkQueue
    - Popular in academic/HPC communities
  - Higher-level task graph libraries:
    - Apache Airflow, Luigi, Snakemake
- Batch queues (HTCondor, slurm, etc.) are now resource provisioning

Facility integration

- **Data delivery** is a main bottleneck for modern analysis at scale
  - True also for AI/ML workloads
  - Requires hardware-software integration to overcome

Facility integration

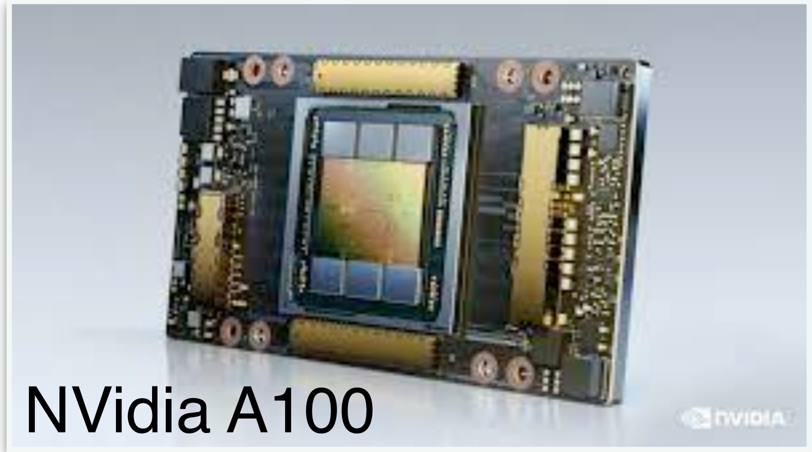
- **Data delivery** is a main bottleneck for modern analysis at scale
  - True also for AI/ML workloads
  - Requires hardware-software integration to overcome
- The ideal analysis facility integration would:
  - Reduce manual user data curation
  - Offload expensive algorithms to accelerators
  - Save compute and storage resources



NVidia A100

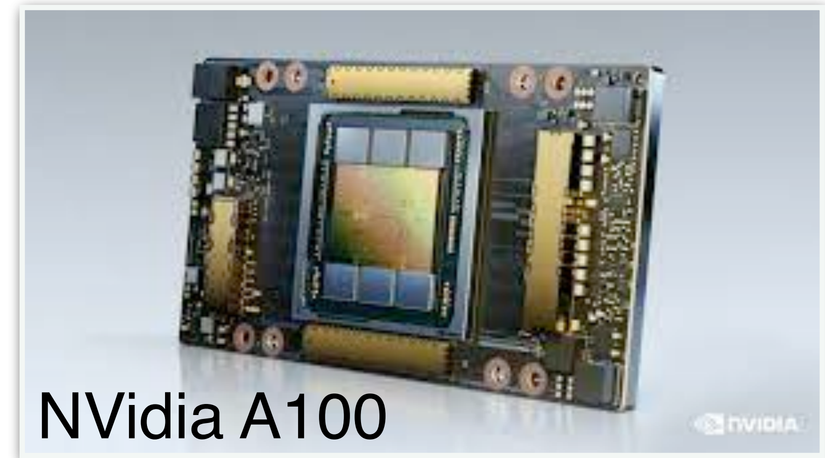
Facility integration

- **Data delivery** is a main bottleneck for modern analysis at scale
  - True also for AI/ML workloads
  - Requires hardware-software integration to overcome
- The ideal analysis facility integration would:
  - Reduce manual user data curation
  - Offload expensive algorithms to accelerators
  - Save compute and storage resources
- …to maintain fast *time to ~~insight~~ physics*

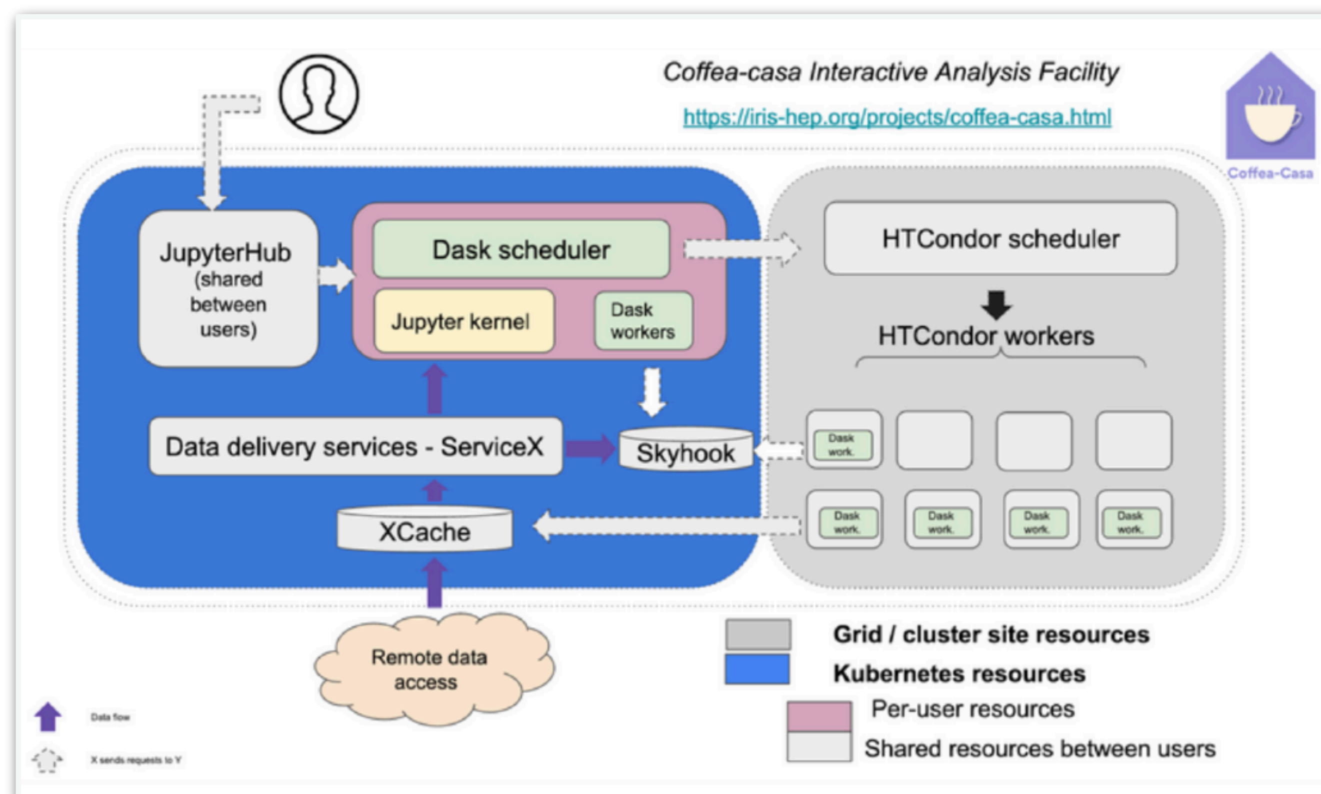


Facility integration

- **Data delivery** is a main bottleneck for modern analysis at scale
  - True also for AI/ML workloads
  - Requires hardware-software integration to overcome
- The ideal analysis facility integration would:
  - Reduce manual user data curation
  - Offload expensive algorithms to accelerators
  - Save compute and storage resources
- …to maintain fast *time to ~~insight~~ physics*



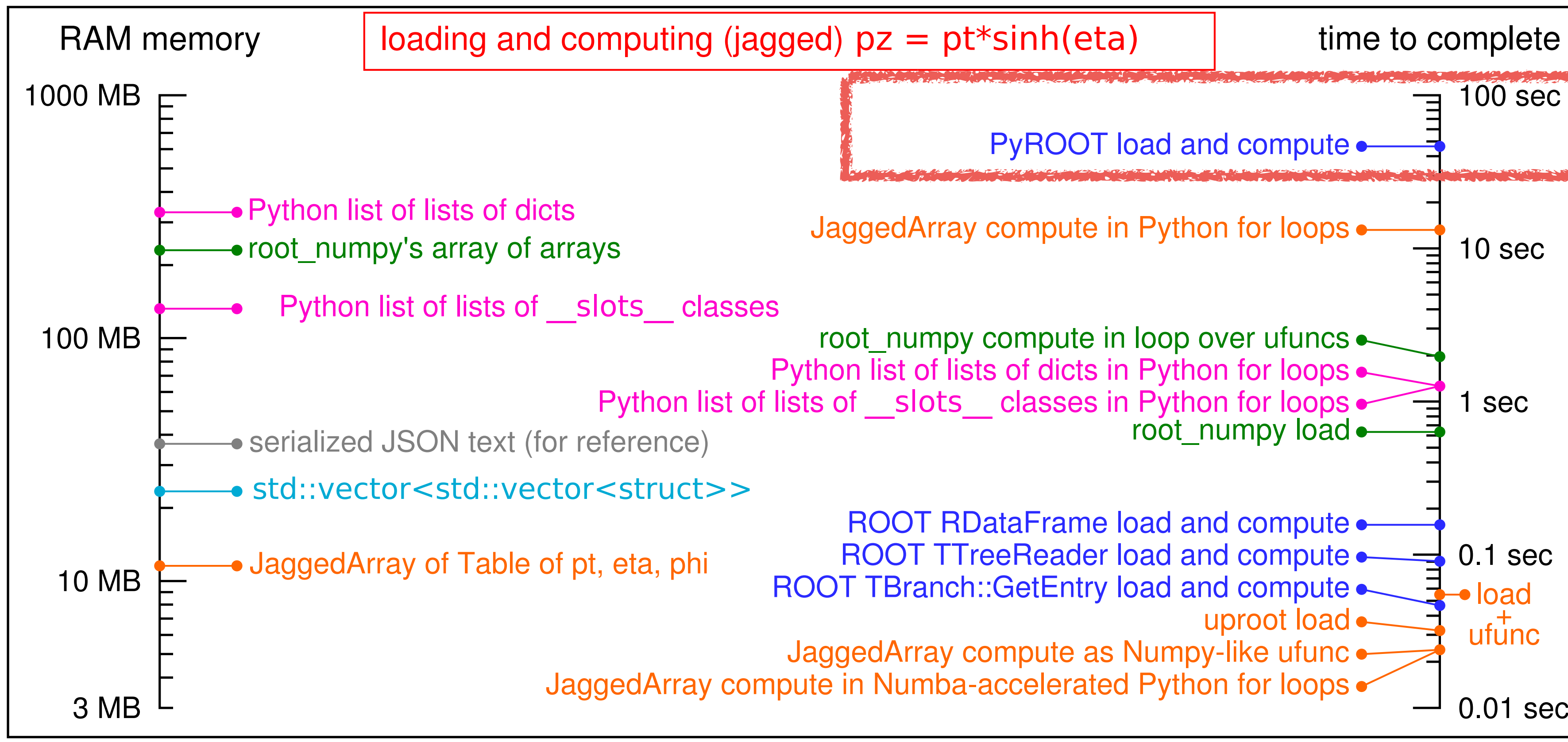
- We now have the playgrounds to realize this vision
  - FNAL Elastic Analysis Facility
  - UNL Coffea-casa
  - & many more…
  - Snowmass contrib. [arxiv:2203.10161]



Fermilab

Performance

- For library designers, important to know when we are fast *enough*
  - **μs to ms per event**
- Users may have other plans



N. Manganelli

J. Pivarski

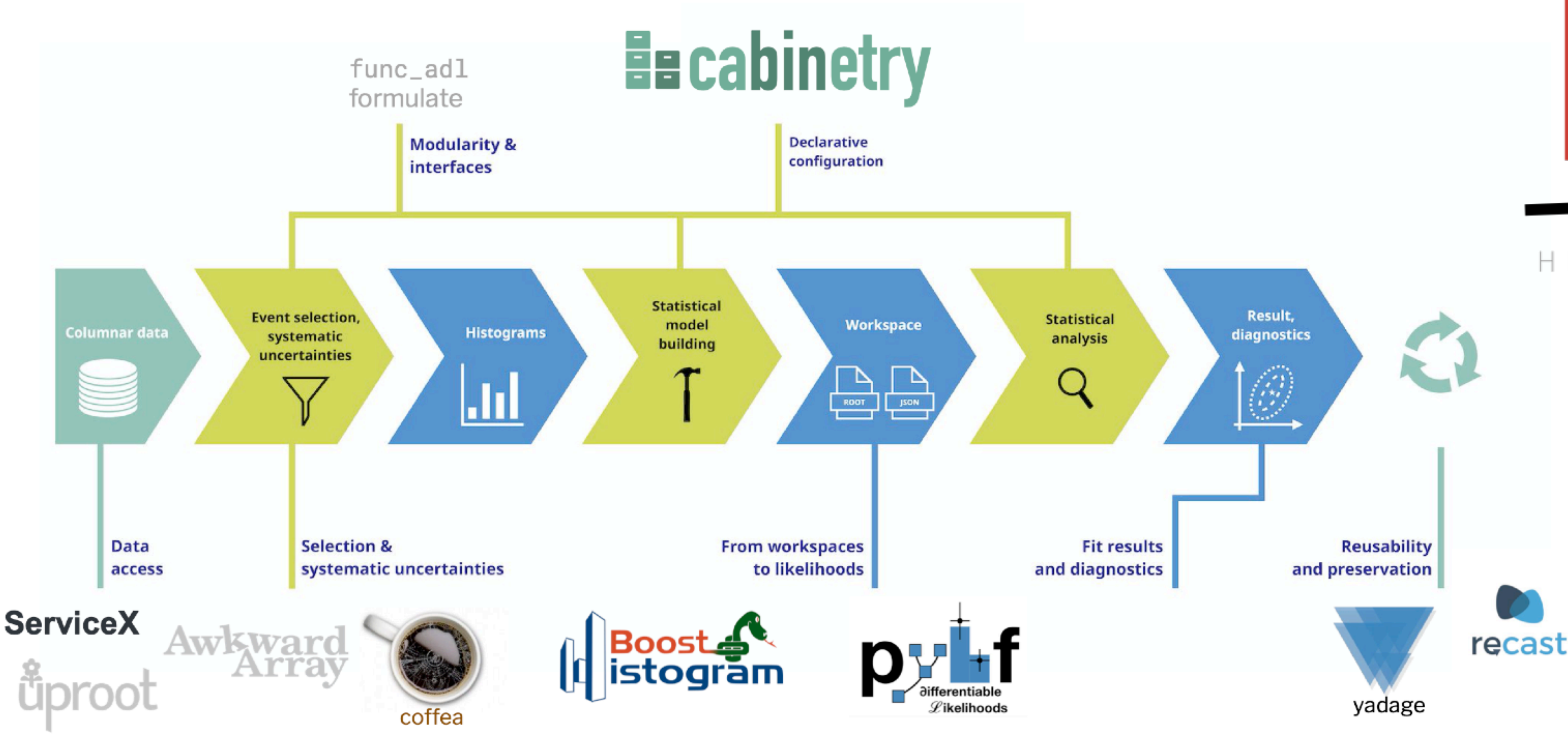
Benchmarking the code and coming out fastest is fantastic

- Factor 3x* is small compared to the O(1000)-O(10000) improvement RDF/coffea have against TTree::Draw-based frameworks (I know of several)

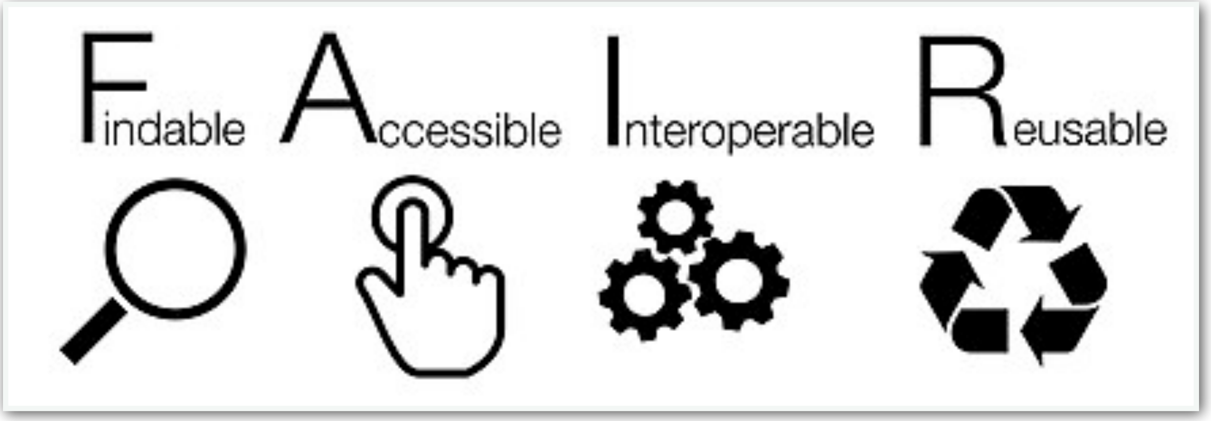
Where can I join this effort?

- Open a PR to any Scikit-HEP repository!
  - Good first issues are often labeled (or fix your own issue)
- Follow HEP Software Foundation meetings
  - [Data Analysis Working Group]
- Talk to IRIS-HEP Analysis systems group
  - https://iris-hep.org/as.html (fellowship programs available)

HEP Software Foundation

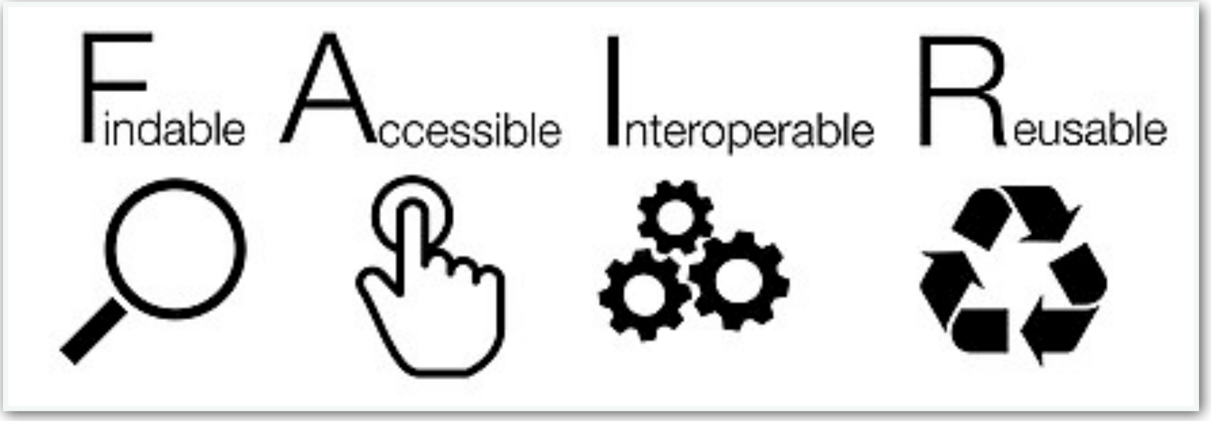


Particle Physics Software Culture



- Analysis software is critical for HEP
  - But new collaborators can struggle
    - Chase down requirements / “recipes”
    - Join group with mature framework / toolset
  - Is our data and metadata FAIR?
    - Software sometimes viewed as competitive advantage

Particle Physics Software Culture

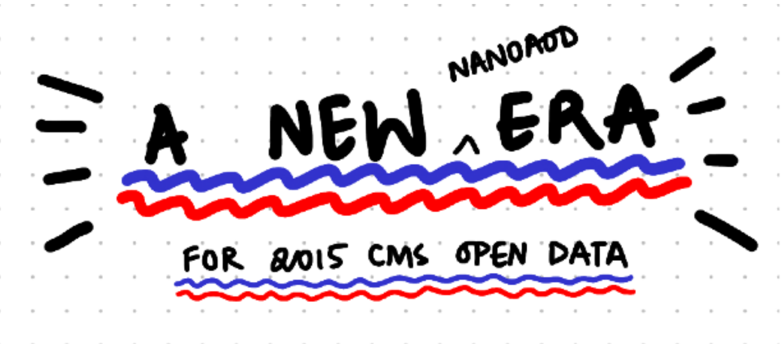


- Analysis software is critical for HEP
  - But new collaborators can struggle
    - Chase down requirements / “recipes”
    - Join group with mature framework / toolset
  - Is our data and metadata FAIR?
    - Software sometimes viewed as competitive advantage

- Need training & mentorship pipeline
  - In same sense as pixels, calorimeters, …software detector?
  - Career paths: Traditional track, Research Software Engineer (US-RSE), ?

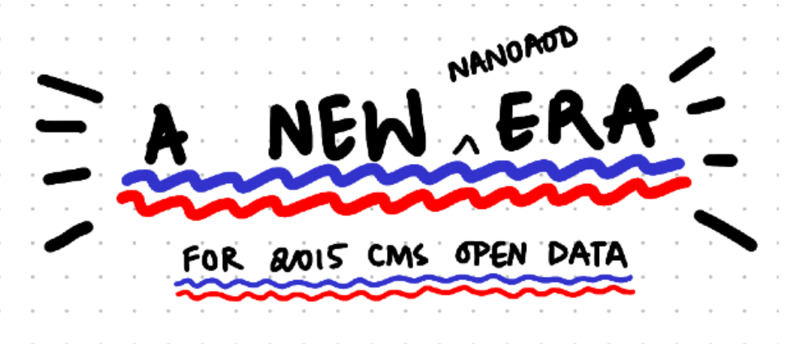
Particle Physics Software Culture

- Open data may enable new discovery
  - Why impose our priors? (Cross-checks expected)
    - Data, metadata, and tools need to be FAIR

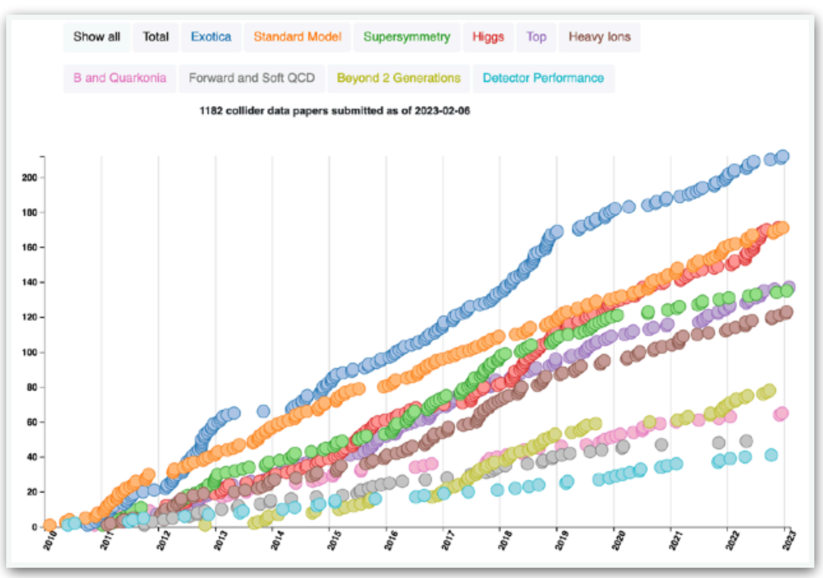


Particle Physics Software Culture

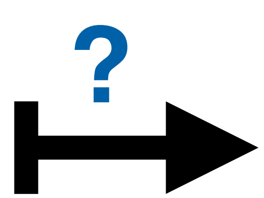
- Open data may enable new discovery
  - Why impose our priors? (Cross-checks expected)
    - Data, metadata, and tools need to be FAIR

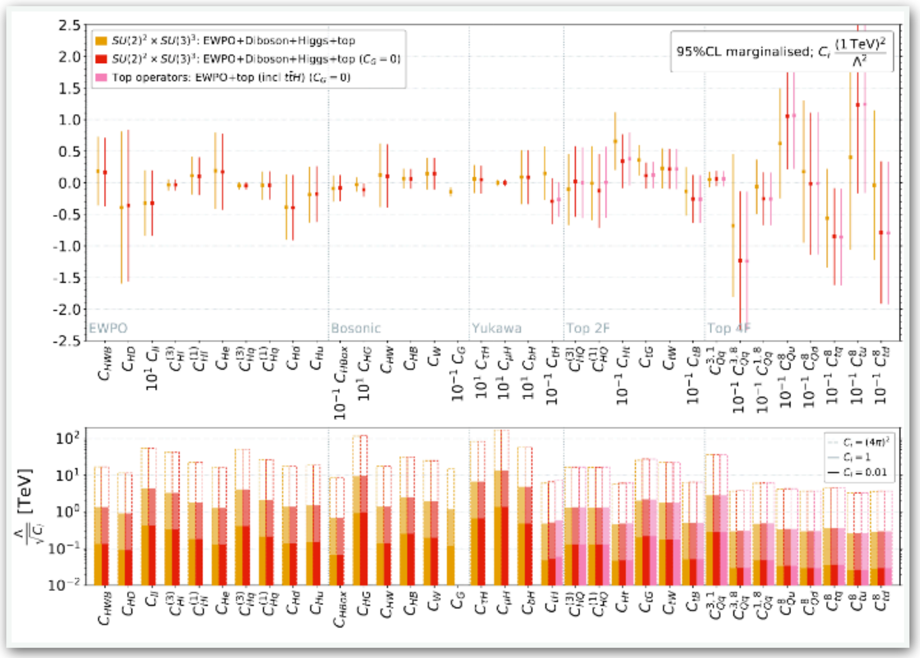


- When will papers fail to capture physics output?
  - e.g. high-dimensional model space: interpretability challenge
    - Combinations of many observables may be key to find new physics
  - Data formats, software tools can meet this challenge



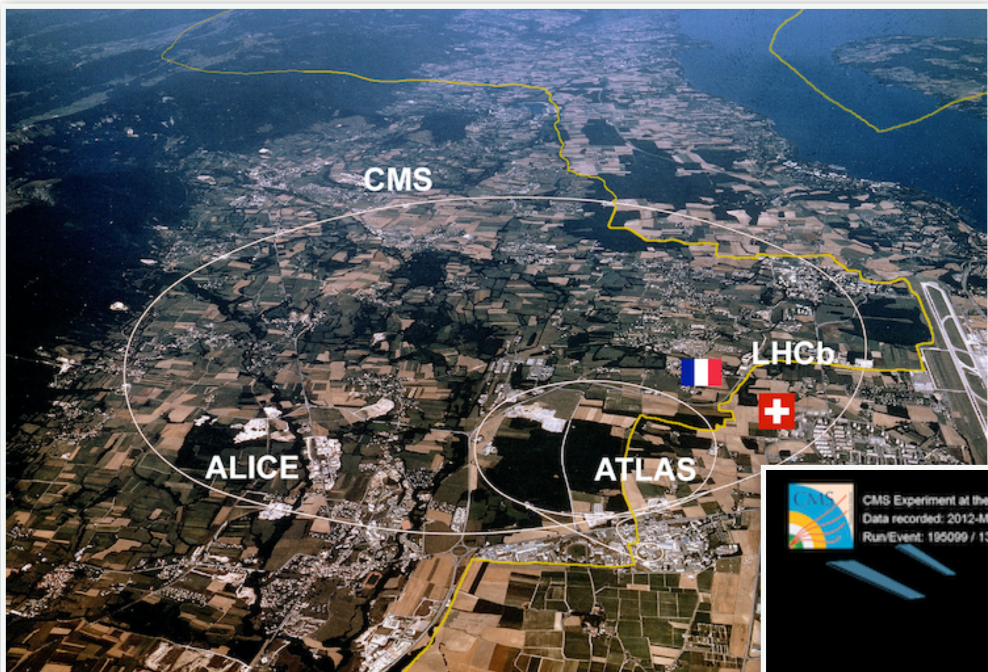
CMS publication count





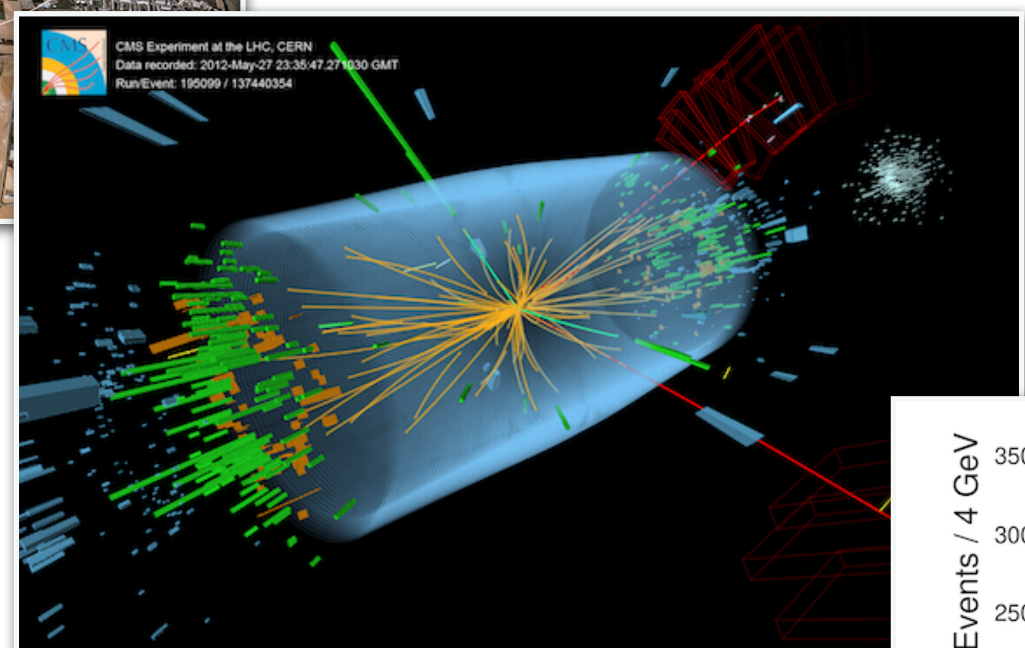
Indirect searches for new physics

HEP Experiment: three easy steps



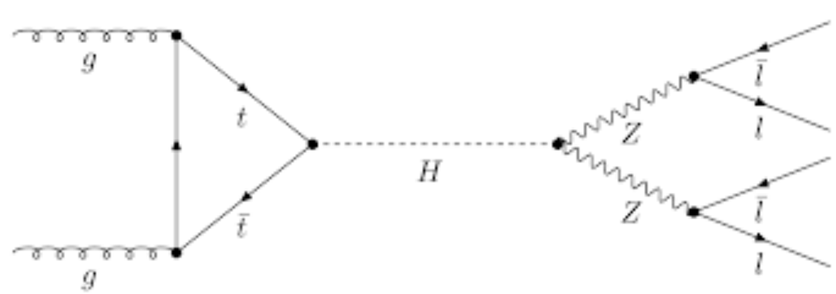
1. Collide particles

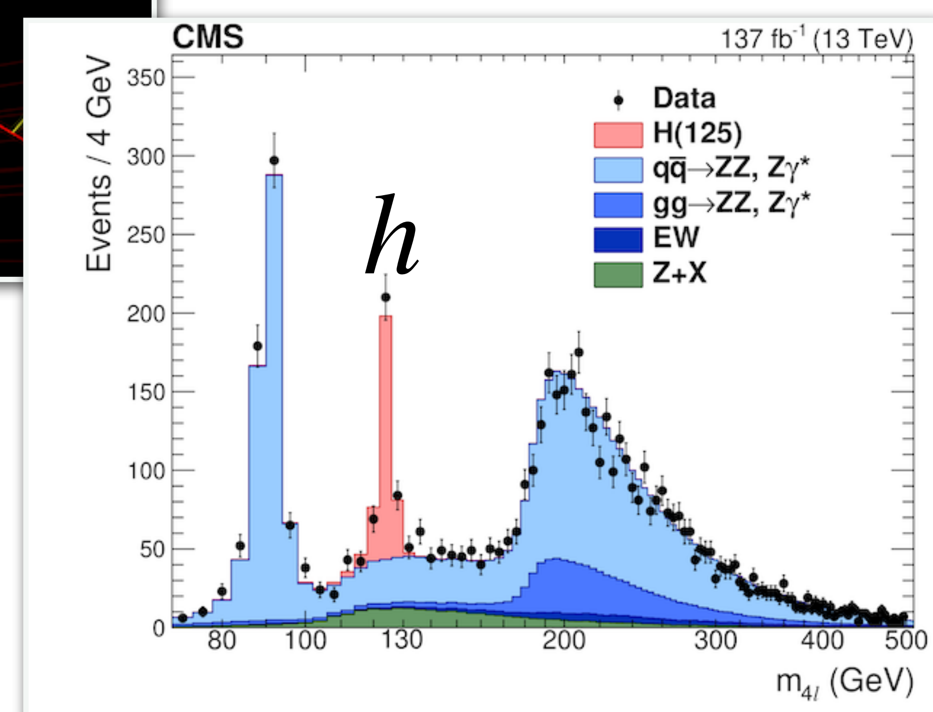
2. Take pictures



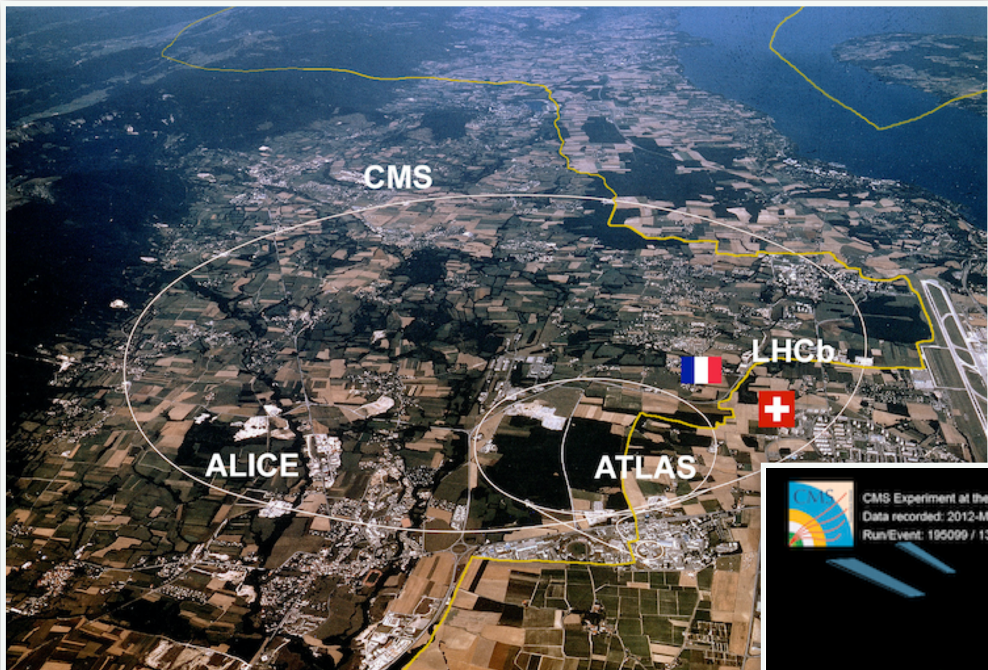
3. Infer parameters

~1MB / event
~100B events



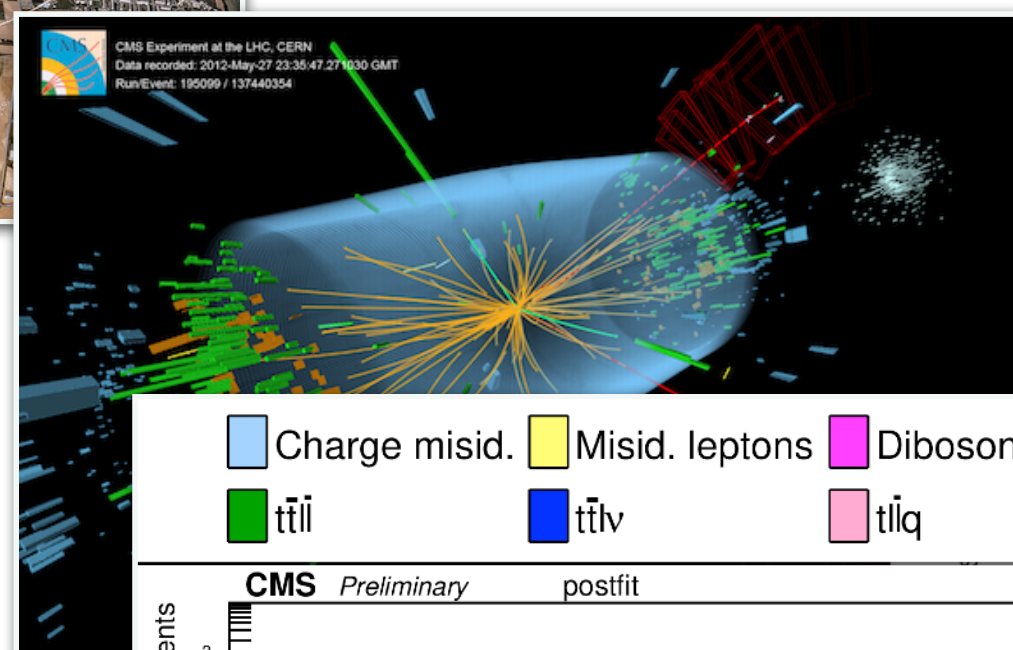


HEP Experiment: three easy steps



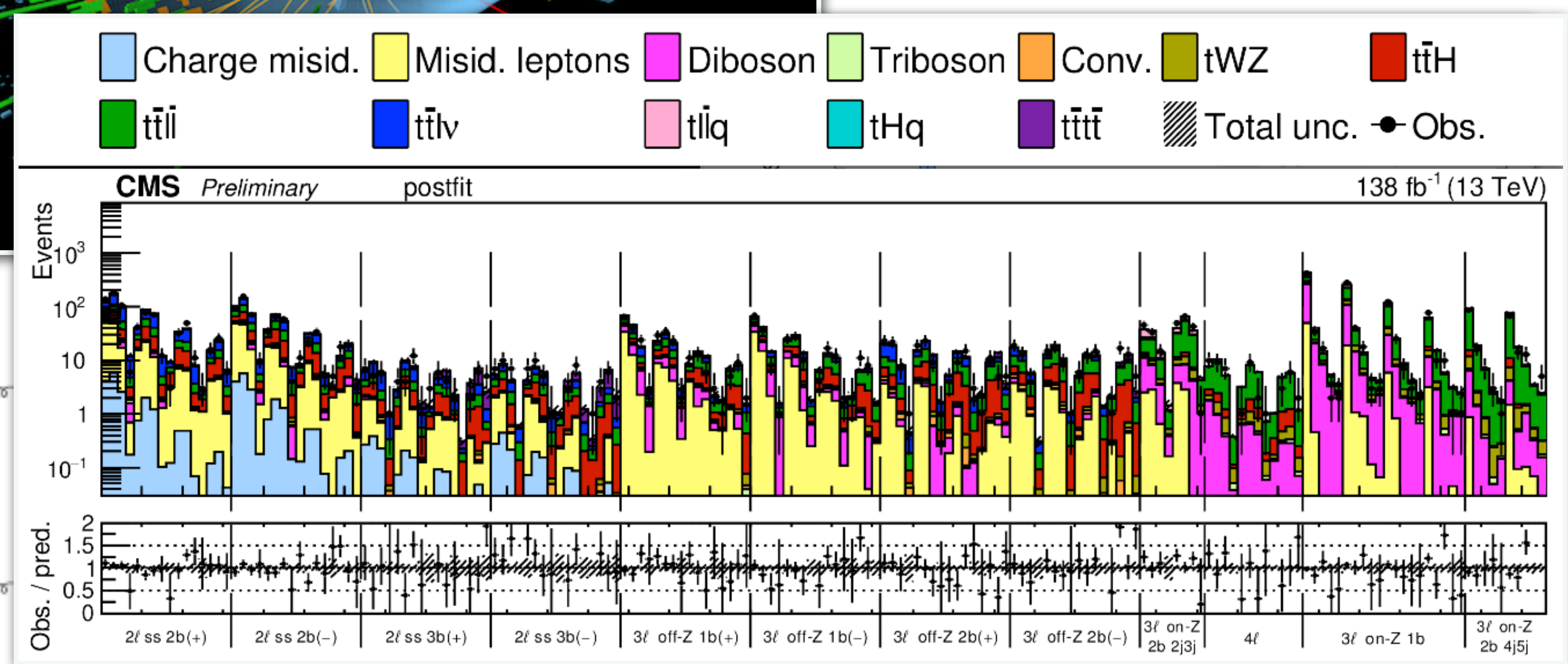
1. Collide particles

2. Take pictures



3. Infer parameters

~1MB / event
~100B events



Conclusion

Conclusion

- A new kind of scaling challenge faces us
  - Complexity of data analysis

Conclusion

- A new kind of scaling challenge faces us
  - Complexity of data analysis

- If we are to make the most of our data, we need innovations in *design*
  - Composable libraries and shared interfaces

Conclusion

- A new kind of scaling challenge faces us
  - Complexity of data analysis
- If we are to make the most of our data, we need innovations in *design*
  - Composable libraries and shared interfaces
- To build a software detector, we need a software culture
  - Build an expert community, and keep it