# AF/UX/ADL/ML

*Toward more efficient end-user analysis on exabyte scale*

Parallel session at [https://indico.cern.ch/e/aew2](https://indico.cern.ch/e/aew2)

## Discussion topics

- "Interactive" analysis workflows? Access methods/interfaces (notebooks, vscode, ssh, "offline" submission)?
  - Analysts - what would it take you to move to an AF?
  - What is "interactive"? Turnaround time of ~minutes?
- AF admins:
  - what would it take to share components (e.g. "Zero-to-Analysis Facility" Helm chart)
  - Gordon's Question: Is Kubernetes it? How do we deploy it in our systems? Admin Training?
  - Can we rely on VO-wide and token-based authentication for AFs like for the Grid?
- Environment setup: how to ensure reproducible environments, and allowing users to specify what they need exactly
- What does Analysis Preservation look like in a non-Batch analysis model (Dask etc)? What are reliable APIs to code against?

## Discussion topics

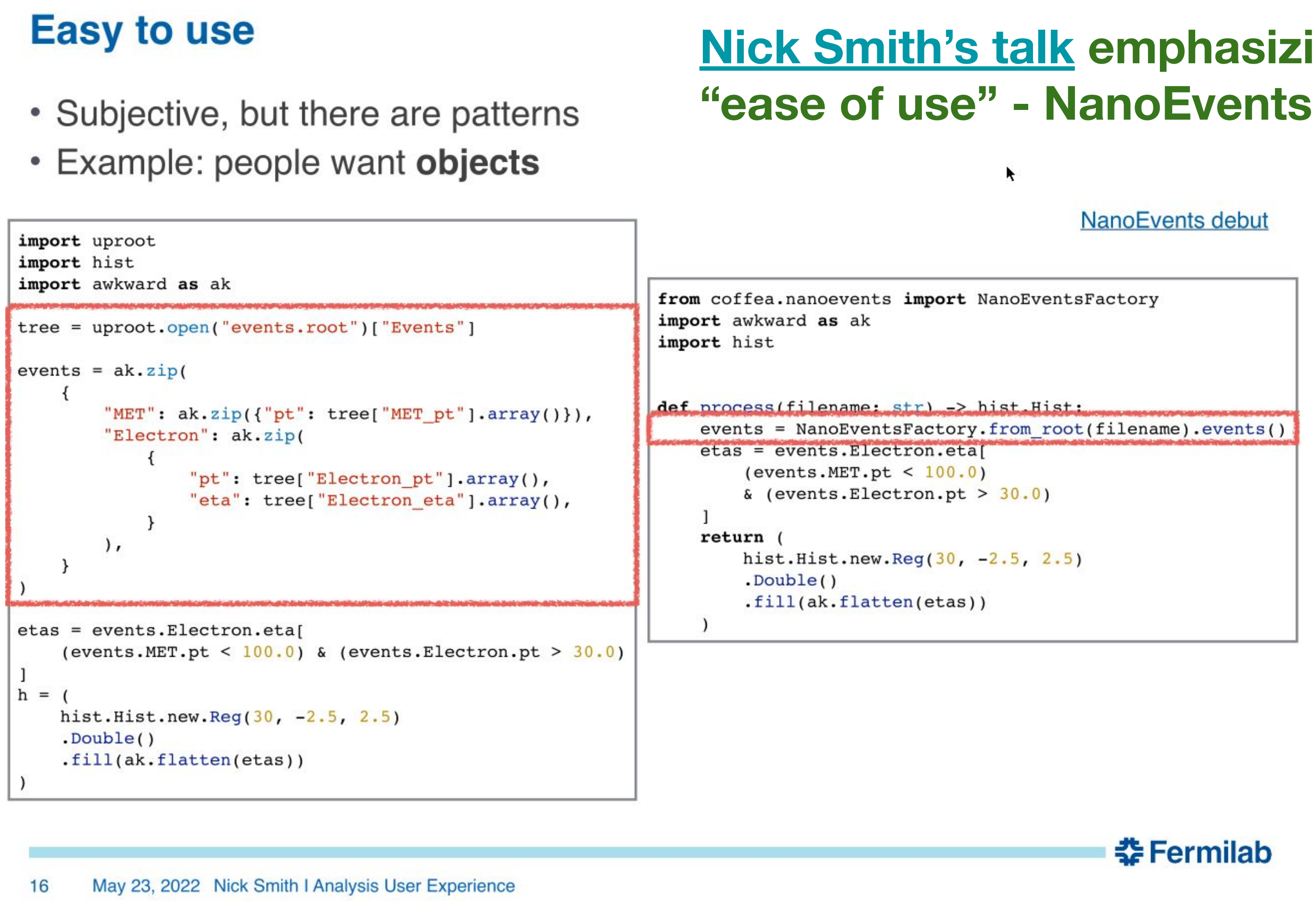
- How can we make the ML development experience better
    - Experiment Tracking, HP optimization, Data preprocessing, GPU profiling?
    - Relationship between “ML Platforms” to AF facilities (KubeFlow, MLFlow, …) ?
    - Can we learn things about metric tracking also for instrumenting non ML code?
- Common Infrastructure:
    - What new Community Datasets / Simulators are needed?
    - Is access to Hardware an issue that we can solve as a community (→ fed. AF access)?
    - Do we expect shared and fine-tunable models like in industry (BERT, GPT-3,...)?
- Differentiable Programming:
    - Can we find examples where downstream feedback meaningfully changes upstream algo?
    - What are prime infrastructure targets for DP? Diffable RDF / Awkward?
      Stats seems clearly within reach & useful
    - Sensitivity Analysis for Systematics

## Interoperability

- Interoperability
  - e.g. ROOT <-> Python HEP data science world
  - Local prototyping vs running on institute Tier-3 vs running on AF vs grid vs …
- Status of interoperability with other ecosystems?
- How do we ensure sustainability within ecosystems (e.g. libraries analysts can rely on long-term)

## Other slides from yesterday relevant for UX / ADL

## Nick Smith’s talk emphasizing “ease of use” - NanoEvents



## Easy to use

- Subjective, but there are patterns
- Example: people want **objects**

```python
import uproot
import hist
import awkward as ak

tree = uproot.open("events.root")["Events"]

events = ak.zip(
    {
        "MET": ak.zip({"pt": tree["MET_pt"].array()}),
        "Electron": ak.zip(
            {
                "pt": tree["Electron_pt"].array(),
                "eta": tree["Electron_eta"].array(),
            }
        ),
    }
)

etas = events.Electron.eta[
    (events.MET.pt < 100.0) & (events.Electron.pt > 30.0)
]
h = (
    hist.Hist.new.Reg(30, -2.5, 2.5)
    .Double()
    .fill(ak.flatten(etas))
)
```

NanoEvents debut

```python
from coffea.nanoevents import NanoEventsFactory
import awkward as ak
import hist


def process(filename: str) -> hist.Hist:
    events = NanoEventsFactory.from_root(filename).events()
    etas = events.Electron.eta[
        (events.MET.pt < 100.0)
        & (events.Electron.pt > 30.0)
    ]
    return (
        hist.Hist.new.Reg(30, -2.5, 2.5)
        .Double()
        .fill(ak.flatten(etas))
    )
```

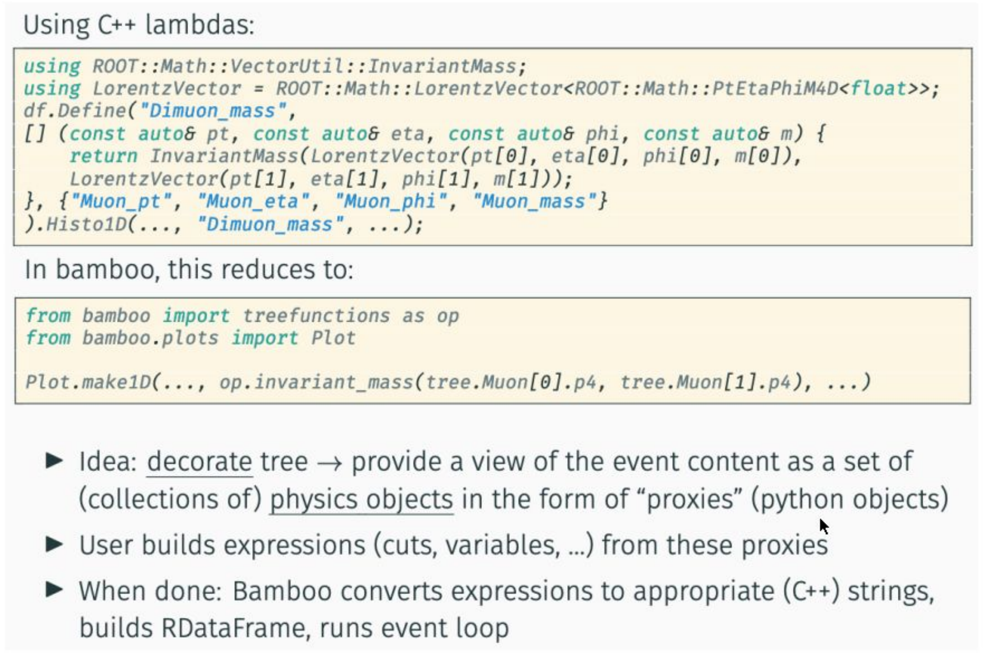
Fermilab

## Easy to use

- Subjective, but there are patterns
- Example: people want **objects**

**Nick Smith's talk emphasizing "ease of use" - Objects in bamboo**

S. Wertz



Using C++ lambdas:

```
using ROOT::Math::VectorUtil::InvariantMass;
using LorentzVector = ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float>>;
df.Define("Dimuon_mass",
[] (const auto& pt, const auto& eta, const auto& phi, const auto& m) {
    return InvariantMass(LorentzVector(pt[0], eta[0], phi[0], m[0]),
    LorentzVector(pt[1], eta[1], phi[1], m[1]));
}, {"Muon_pt", "Muon_eta", "Muon_phi", "Muon_mass"}
).Histo1D(..., "Dimuon_mass", ...);
```

In bamboo, this reduces to:

```
from bamboo import treefunctions as op
from bamboo.plots import Plot

Plot.make1D(..., op.invariant_mass(tree.Muon[0].p4, tree.Muon[1].p4), ...)
```

- Idea: decorate tree $\rightarrow$ provide a view of the event content as a set of (collections of) physics objects in the form of "proxies" (python objects)
- User builds expressions (cuts, variables, ...) from these proxies
- When done: Bamboo converts expressions to appropriate (C++) strings, builds RDataFrame, runs event loop