Analysis user experience & declarative languages

Summary (https://indico.cern.ch/e/aew2)

Session conveners:
Jonas Rembser (CERN)
Alexander Held (UW-Madison)

Teaching programming vs simplifying tools

- Big discussion topic on Tuesday: **educate analysers in software engineering** techniques **vs. sufficiently simplify tools** (-> also ADLs)
  - Users frequently report relatively basic problems they run into
- **How high-level** should the **interfaces** for analysis be?
  - High-level: we can replace backends with newer ones without having to rely on user adoption
  - Lower lever hooks a must for implementing more complex analysis
- On **performance**: how to know if your analysis has obvious bottlenecks?
  - We need easy tools to at least check if it’s IO limited
  - Are users aware of profiling tools?
- **Large interest in training courses** (e.g. HSF C++)

Pain points in analysis user experience, ordered

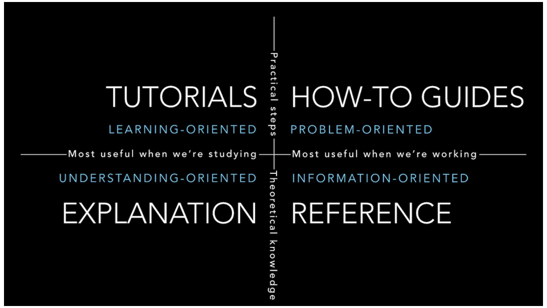
1. **Systematics**
   - Recurring topic throughout this workshop: this is not solved

2. **Metadata**
   - Finding & handling information

3. **Scale-out**
   - Prototyping vs scale-out, different implementations / details on different sites
   - Need for consistent environments across all resources

https://documentation.divio.com/

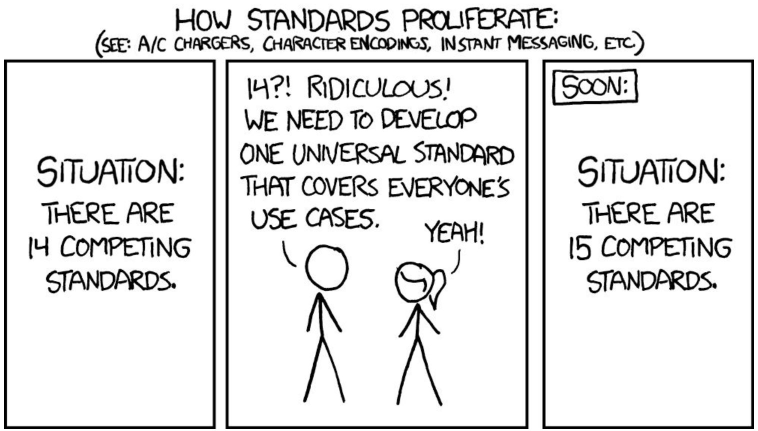
Onboarding people into the ecosystem



- Identified **success stories**
  - LHCb Starterkit highlighted as success story
    - Regularly updated
  - Discussion forums / Slack / Mattermost
  - Compared to AEW1: things are better!
- Writing **tutorials** etc. can be difficult: what should they address? Importance of **user input to inform design**
  - Feedback loop between user support / discussion platforms and good how-to guides: Questions to the developers motivate them to write documentation which increases adoption by new users that ask for support
  - Bringing together users and developers to work on documentation (hackathons?)

Interoperability

- Identified **histograms** as key area for improving interoperability
- We should ensure that **Python bindings** are interoperable
- Demand for general **statistical model format** based on JSON
- Discussion about **data interoperability** at the level of individual **columns** (also in memory)



Towards a better future - what would we like?

- Automatic (graph-based) **optimization** (à la RDF Vary()) to enable users to focus on physics instead of optimization
- **Grouping of columns into objects** to allow physics reasoning
- **Factoring out** the **analysis chores** to mini-frameworks / libraries
- More **documentation and learning material**

Thank you!

- For all your **contributions**
- For the **very interesting discussions**
- For **taking notes**
- For making this workshop a **great experience**!

Backup

Analysis user experience

- Do we understand “average” analyzer experience?
  - Do we only know from a particular group / are biased
  - Work towards a survey? Other ways of gathering feedback?

- Are analyzers using new approaches we may recommend?
  - What can we do to help with adoption?
  - Is there anything missing in the workflows we are envisioning?

- Are there specific requirements from small / new experiments?

New programming languages and ADLs

- What is the future of Julia in HEP?
  - Other languages to keep track of?
  - Lessons to be learnt from interest in Julia?

- Analysis description languages (ADLs)
  - How does the future of ADLs look like?
  - What are potential barriers to adoption?

Sezen Sekmen's talk

```
# OBJECTS
object goodMuons
  take muon
  select pT(muon) > 20
  select abs(eta(muon)) < 2.4

object goodEles
  take ele
  select pT(ele) > 20
  select abs(eta(ele)) < 2.5

object goodLeps
  take union(goodEles, goodMuons)

object goodJets
  take jet
  select pT(jet) > 30
  select abs(eta(jet)) < 2.4
  reject dR(jet, goodLeps) < 0.4
```

```
# EVENT VARIABLES
define HT = sum(pT(goodJets))
define MTl = Sqrt( 2*pT(goodLeps[0]) * MET*(1-cos(phi(METLV[0]) - phi(goodLeps[0]) )))

# EVENT SELECTION
region baseline
  select size(jets) >= 2
  select HT > 200
  select MET / HT <= 1

region signalregion
  baseline
  select Size(goodLeps) == 0
  select dphi(METLV[0], jets[0]) > 0.5
  histo hMET , "met (GeV)", 40, 200, 1200, MET

region controlregion
  baseline
  select size(goodLeps) == 1
  select MTl < 120
```

- Organized structuring of the analysis helps easy overview.
- ADL implementations of numerous public LHC analyses exist and more implementations ongoing.

Documentation, examples, benchmarking, performance

- Documentation, examples -> next slide
  - State of documentation / what and how to improve?
  - Incentives: funding, dedicated positions, …
  - UX of analyzers going from simple tutorials to full-scale analyses?

- Benchmarking & performance
  - Time to write code vs time to run code

Nick Manganelli

- 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)

Nick Smith

Solutions must be:

Easy to use

Scalable

Fast

Practical steps

TUTORIALS | HOW-TO GUIDES

LEARNING-ORIENTED | PROBLEM-ORIENTED

Most useful when we're studying | Most useful when we're working

UNDERSTANDING-ORIENTED | INFORMATION-ORIENTED

EXPLANATION | REFERENCE

Theoretical knowledge

https://documentation.divio.com/

Interoperability

- Interoperability (e.g. ROOT <-> Python HEP data science world) is crucial
  - Where do we stand?
  - Which improvements are needed?
- Status of interoperability with other ecosystems?

Slides from Monday's plenary

- Summary of the ROOT workshop with highlight on status and plans
  Axel Naumann
- Analysis user experience with the Python HEP ecosystem
  Jim Pivarski
- Declarative languages overview
  Sezen Sekmen

Easy to use

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

Nick Smith's talk emphasizing "ease of use" - NanoEvents

NanoEvents debut

```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))
)
```

```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

16 May 23, 2022 Nick Smith I Analysis User Experience

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 → 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

[Stephan Hageböck's talk](#) on systematics

Core Software to the Rescue? – The Future

```
ROOT::RDataFrame df("Events", "root://eospublic.cern.ch//eos/opendata/cms/derived-data/AOD2NanoAODOutreachTool/Run2012BC_DoubleMuParked_Muons.root");

MuonCalibrationTool calibrationTool{};
auto df_calib = df.Redefine("Muon_pt",
   [&](RVecF const & pts, RVecF const & etas) {
     return calibrationTool.calibratePTs(pts, etas, Sys::Nominal);
   }, {"Muon_pt", "Muon_eta"});

df_calib = df_calib.Vary("Muon_pt",
   [&](RVecF const & pts, RVecF const & etas) {
     RVecF down = calibrationTool.calibratePTs(pts, etas, Sys::MomentumScaleDo);
     RVecF up   = calibrationTool.calibratePTs(pts, etas, Sys::MomentumScaleUp);
     return RVec{down, up};
   }, {"Muon_pt", "Muon_eta"},
   {"Do", "Up"},
   "MuonMomentumScale"
);

// Start analysis selection
auto df_2mu = df_calib.Filter("nMuon == 2", "Events with exactly two muons");
auto df_os = df_2mu.Filter("Muon_charge[0] != Muon_charge[1]", "Muons with opposite charge");
```

Apply calibration

Apply systematic

[RDF Talk - CMS Analysis Tools Task Force](#)