Evolution of Declarative Languages

G. Watts (UW/Seattle)

2024-03-14

What are declarative languages?

What are declarative languages

Goal: Reduce Boiler Plate Code!

```
void MyAnalysisAlgorithm::execute()
{
    // Retrieve the JetContainer from the xAOD
    const xAOD::JetContainer* jetContainer = nullptr;
    if (!evtStore()->retrieve(jetContainer, "MyJetContainer").isSuccess())
    {
        ATH_MSG_ERROR("Failed to retrieve JetContainer");
        return;
    }

    // Vector to store the pt's of the jets
    std::vector<float> jetPtVector;

    // Loop over all the jets in the JetContainer
    for (const auto& jet : *jetContainer)
    {
        // Push the pt of the jet onto the vector
        jetPtVector.push_back(jet->pt());
    }

    // Do something with the jetPtVector...
}
```

What are declarative languages

Goal: Reduce Boiler Plate Code!

```
df_future = ds \
        .SelectMany('lambda e: e.Jets("AntiKt4EMTopoJets")') \
        .Select('lambda j: j.pt()/1000.0') \
        .AsPandasDF('JetPt')
```

func_adl

What are declarative languages

Goal: Reduce Boiler Plate Code!

"Write code to extract all jet pt's from dataset X"

- Our LLM Future

What are declarative languages

Goal: Reduce Boiler Plate Code!

Even a compiler is a **declarative language**: it translate intent into assembly language!

What Drives Interest?

Complexity of Software

- Deriving scale factors and systematic errors often requires large amount of configuration
- Accessing data can be complex and require a lot of knowledge
- Software is designed to be general and foundational
  - Access data of type “jet”
- However, analysis needs specifics
  - Access the particle flow jets

Complexity of Hardware

- Scale Out
  - DASK, batch, etc.
- Data Location
  - GRID, local disk, rucio
- Facilities
  - Different facilities have different “setups” and magic configuration commands.

What Drives Interest?

Complexity of Software

- Deriving scale factors and systematic errors often requires large amount of configuration
- Accessing data can be complex and require a lot of knowledge
- Software is designed t
  - Access data of typ
- However, analysis nee
  - Access the partic

None of this is really the **physics**!

Complexity of Hardware

- Scale Out
  - DASK, batch, etc.
- Data Location
  - GRID, local disk, rucio
- Facilities
  - Different facilities have different “setups” and magic configuration commands.

What Drives Interest?

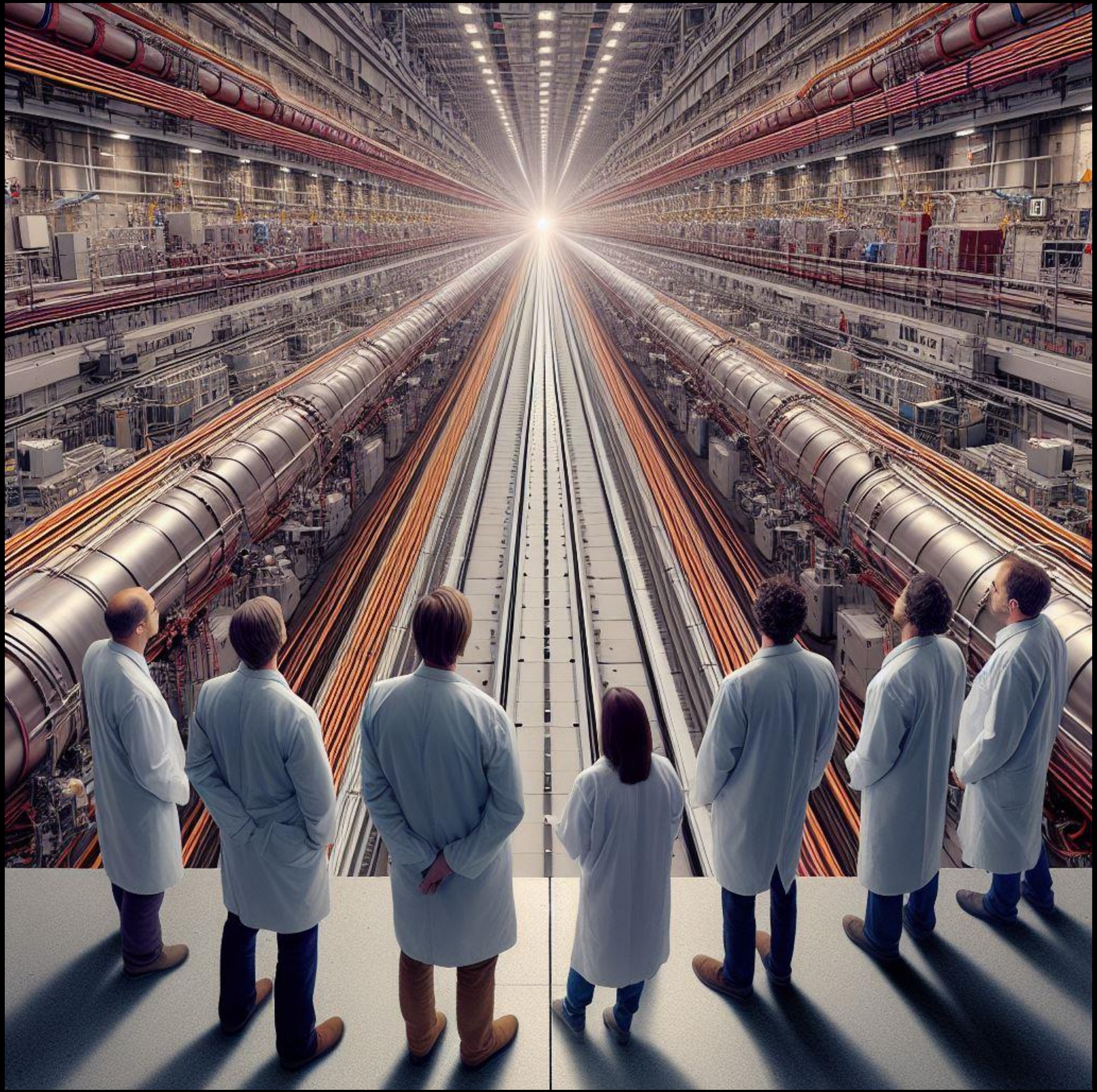
Physics

Selection cuts, ML, mass, statistical models, etc.

Infrastructure

Submit batch job, interactive scale-out, temporary results storage, dataset locations, plot storage, versioning, etc.

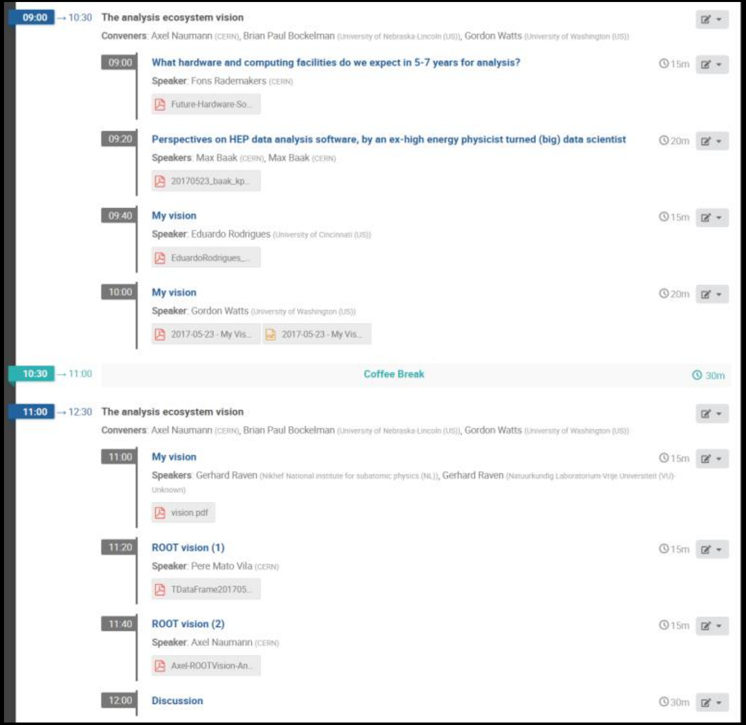
History



Amsterdam (2017)

indico

HEP analysis ecosystem workshop



09:00 → 10:30 The analysis ecosystem vision

Conveners: Axel Naumann (CERN), Brian Paul Bockelman (University of Nebraska-Lincoln (US)), Gordon Watts (University of Washington (US))

- 09:00 **What hardware and computing facilities do we expect in 5-7 years for analysis?** — 15m
  Speaker: Fons Rademakers (CERN)
  Future-Hardware-So...
- 09:20 **Perspectives on HEP data analysis software, by an ex-high energy physicist turned (big) data scientist** — 20m
  Speakers: Max Baak (CERN), Max Baak (CERN)
  20170523_baak_kp...
- 09:40 **My vision** — 15m
  Speaker: Eduardo Rodrigues (University of Cincinnati (US))
  EduardoRodrigues_...
- 10:00 **My vision** — 20m
  Speaker: Gordon Watts (University of Washington (US))
  2017-05-23 - My Vis... 2017-05-23 - My Vis...

10:30 → 11:00 Coffee Break — 30m

11:00 → 12:30 The analysis ecosystem vision

Conveners: Axel Naumann (CERN), Brian Paul Bockelman (University of Nebraska-Lincoln (US)), Gordon Watts (University of Washington (US))

- 11:00 **My vision** — 15m
  Speakers: Gerhard Raven (Nikhef National institute for subatomic physics (NL)), Gerhard Raven (Natuurkundig Laboratorium-Vrije Universiteit (VU)-Unknown)
  vision.pdf
- 11:20 **ROOT vision (1)** — 15m
  Speaker: Pere Mato Vila (CERN)
  TDataFrame201705...
- 11:40 **ROOT vision (2)** — 15m
  Speaker: Axel Naumann (CERN)
  Axel-ROOTVision-An...
- 12:00 **Discussion** — 30m

On the worst ennemies of analysts

(Eduardo Rodrigues)

1. Lack of knowledge and/or incompetence
   Solution: "You might want to look into …", a British would say
2. Inertia – the thinking "This is not working well but I will keep working this way rather than spending time investigating (potentially rewarding) alternatives!"
   Solution: the hard-to-trigger shift in culture (problem)
3. Inappropriate and/or not timely available software working environment
   - Broad subject touching many points discussed here (SW stacks, notebooks, etc.)
   Solution: I rely on a great core software team
4. Inappropriate analysis software ecosystem
   Solution: We are here to discuss this … ;-)
5. Bad distributed computing model - user interaction with the Grid just isn't great, let's be honest
   - For example LHCb has Ganga, very much underpowered and often problematic
   Solution: we probably need a common tool (sustainability & maintainability ✓)
6. Amount of data and the complexity of analyses (tasks)
   - But this should actually be seen as a good thing ;-)
   Solution? I do not want one. I rather want the "problem" of more data!

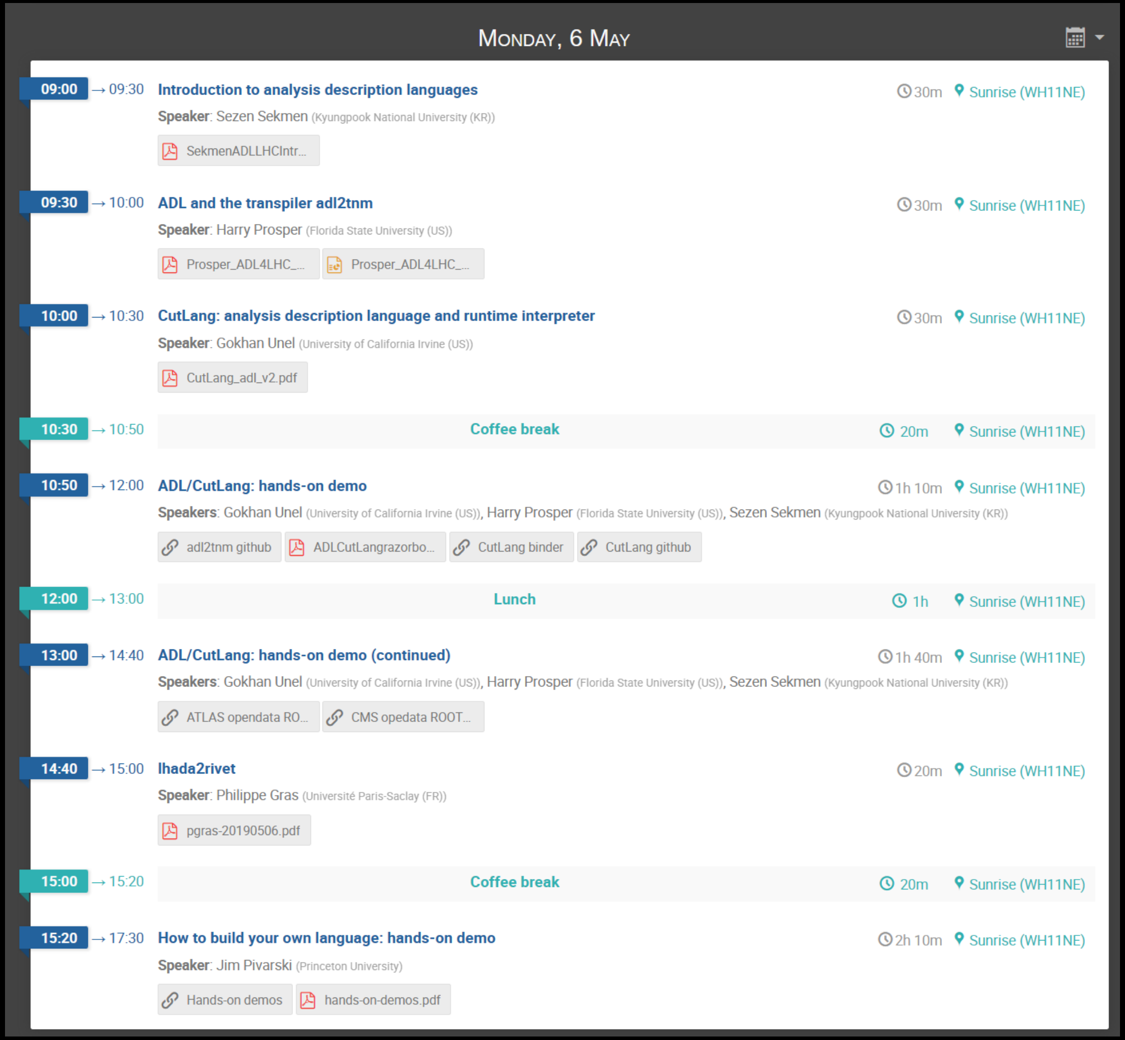
Amsterdam

From the report:

- A frequent observation during the workshop is that **these steps** often **involve semantics that can be expressed in a declarative programming style**.
- The use of declarative interfaces may allow a hiding of the concurrency complexity.
- Declarative- or functional-like models are a particularly advantageous way HEP analysts could focus on describing “what they want to do” as opposed to “how to do it.”
- Another recurring theme was functional / declarative style programming, **decoupling problem description from implementation**.
- Functional, declarative programming was a prominent theme. **Declarative avoids over-specifying the order of operations. Functional avoids over-specifying sequential/parallel**. This was part of a more general theme of splitting the “what I want” from the “how it’s produced,” which avoids over-specifying incidental details in general.

Analysis Description Languages for the LHC (2019)

indico



- Included a survey of many languages
  - ADL (Analysis Description Language)
  - CutLang
  - Lhada2rivet
  - C#'s LINQ
  - NAIL (Natural Analysis Implementation Language)
  - YAML as an ADL
  - TTreeFormula
  - AEACUS and RHADAMANTUS
- Covered topics like preservation, what is missing, etc.
- Event included a talk by Jim Pivarski *How to build your own language: hands-on demo"*

Types of Declarative Languages

Domain Specific Language

Use an existing (general purpose) programming language. Use language features to add analysis semantics.

Confined to the language's syntax, but can take advantage of all the existing knowledge and syntax (and sometimes libraries)

Hard to shift to a different backend language (e.g. if you need C++ as part of your backend, or batch jobs)

Standalone

Everything is under the control of the language author

Syntax can be purpose suited to HEP analysis, but must re-invent/code general purpose features (e.g. expressions)

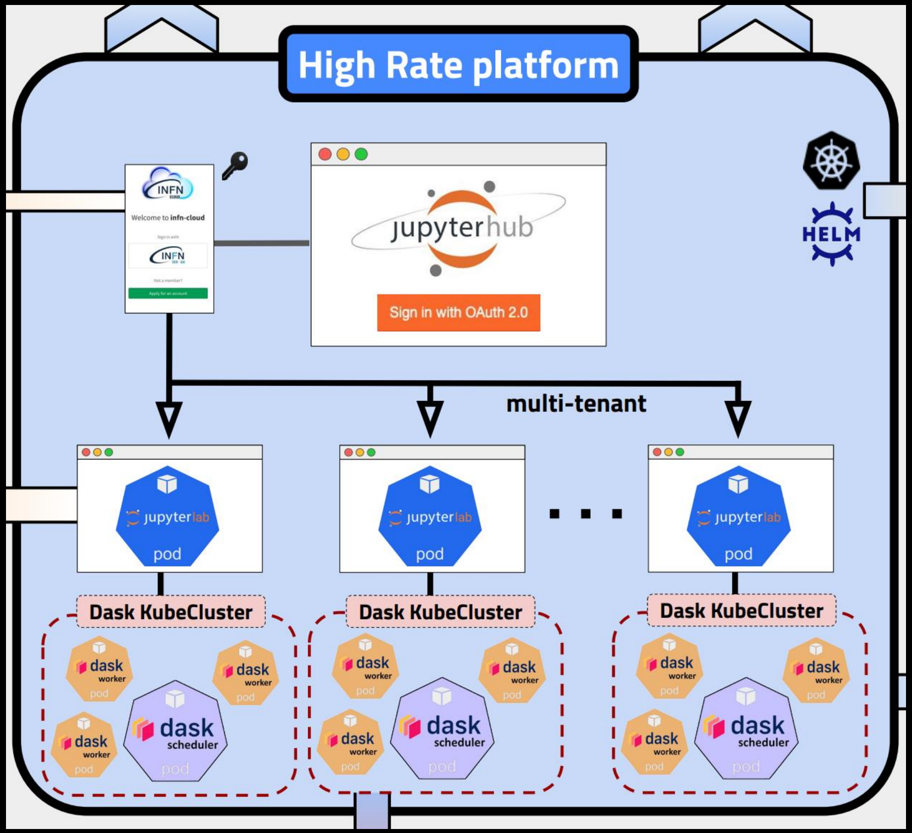
Relatively easy to move between different backends. But also difficult to use external libraries written by others.

Interactive Analysis

An old new use case

Language must accommodate the **development** phase of an analysis

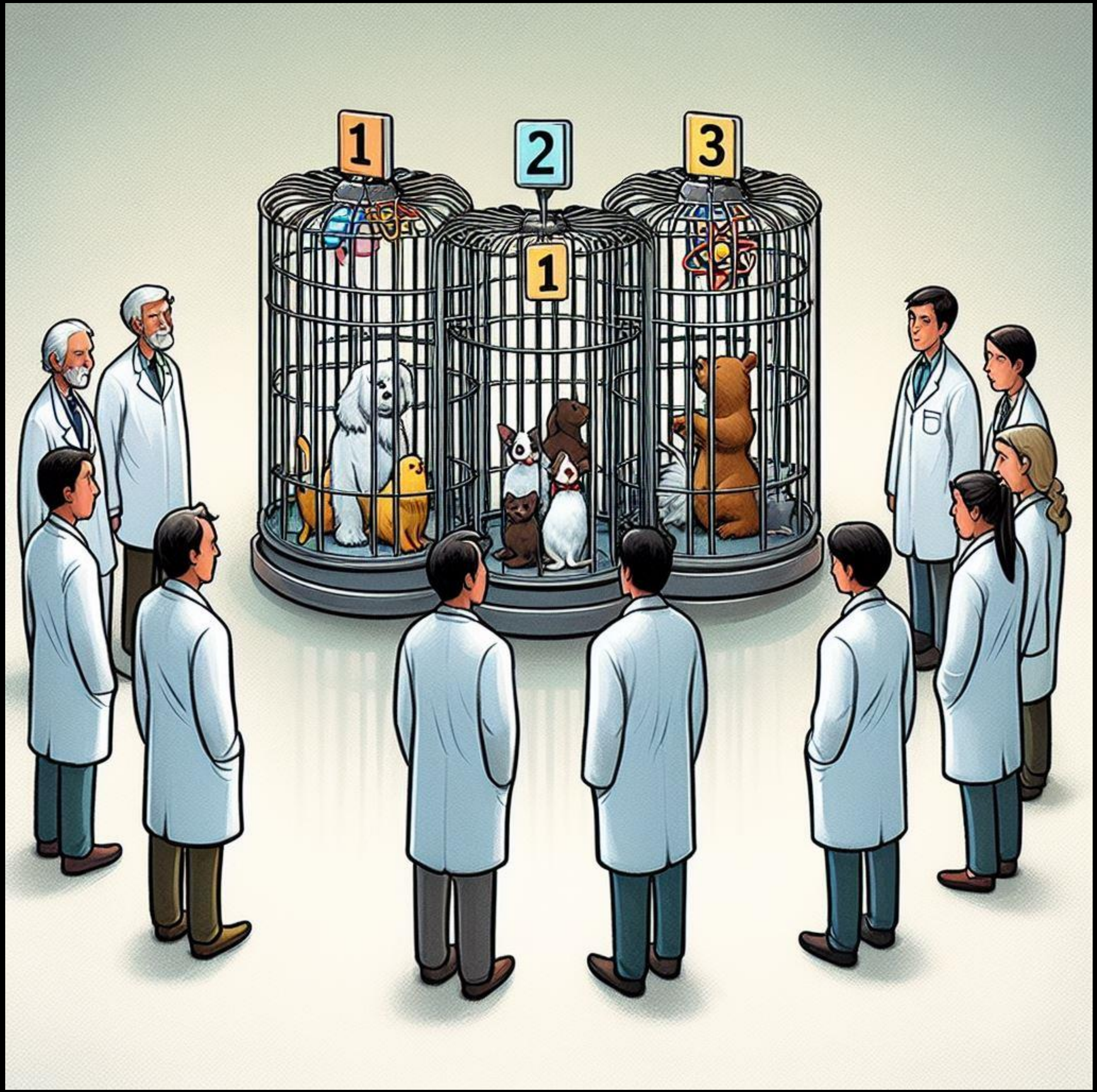
- Want to use commonly defined virtual datasets (e.g. pre-selection, selection, control region A, etc.)
- Investigate plots that are never used in the final analysis note
- Build ML training samples
- Re-run a single plot many times
- Easily adjust selection cuts
- Active code, plots, and text combined (notebooks)



From the Poster "Quasi interactive analysis of High Energy Physics big data with high throughput" by Tommaso Tedeschi

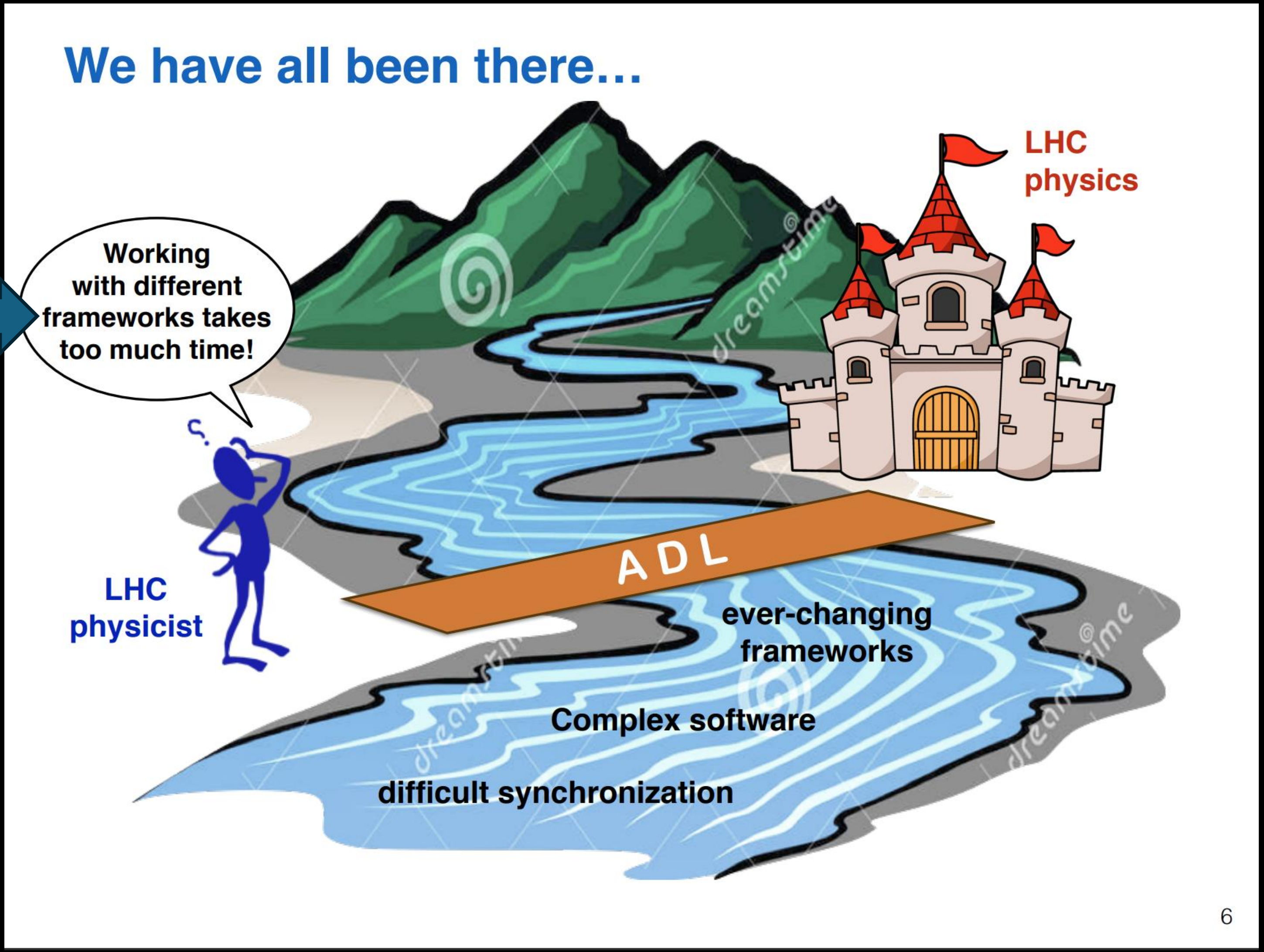
Many Declarative Analysis Languages are not built to support this!
(though they can)

A few Examples



(ADL Proposal)

ADL



ADL

"Standalone"

```
info analysis
  experiment      ATLAS
  id              EXOT-2016-32
  publication     Eur.Phys.J. C77 (2017) no.6, 393
  sqrtS           13.0 # TeV
  lumi            36.1 # 1/fb
  arXiv           1704.03848
  hepdata         https://www.hepdata.net/record/ins1591328
  doi             10.1140/epjc/s10052-017-4965-8
```

- Based around text files
  - Declaration Bocks
- Aimed at a full analysis chain
- Based off underlying experiment definition
  - E.g. "Muon"
- Designed to work with multiple frameworks
- Aimed at letting both experimentalists and theorists complete an analysis
- Works in several experiment contexts
- Very successful in helping students get started

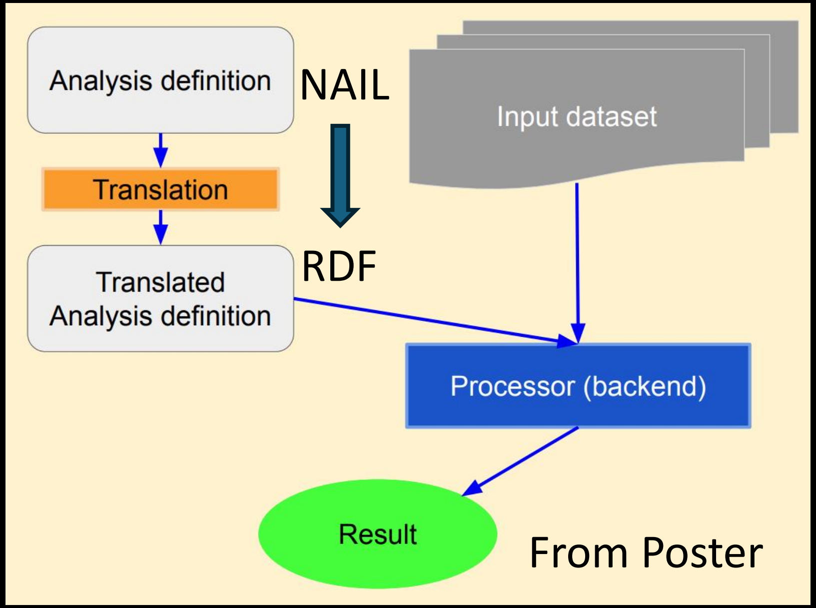
```
object muonsVeto
  take Muon
  select pt > 5
  select |eta| < 2.4
  select softId == 1
  select miniPFRelIso_all < 0.2
  select |dxy| < 0.2
  select |dz| < 0.5

# jets - no photon
object AK4jetsNopho
  take AK4jets j
  reject dR(j, photons) < 0.4 and
           photons.pt/j.pt [] 0.5 2.0
```

See ACAT Poster!!

Example: NAIL "DSL"

Natural Analysis Implementation Language

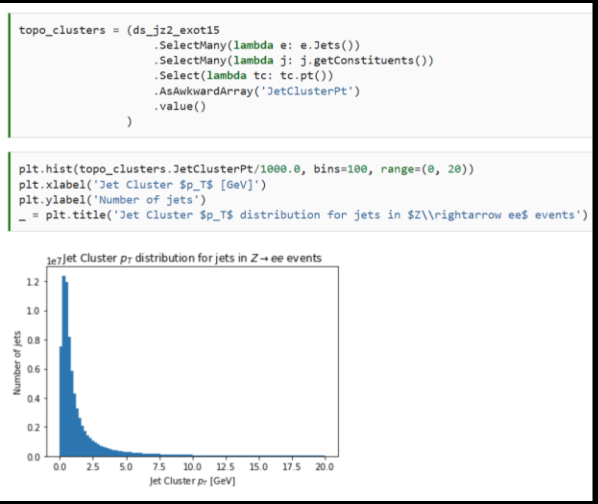
- Python-based DSL
- Backend is RDataFrame
  - Semantics are a loose wrapper around RDF
- Can make histograms or output files (for training!)




```python
from nail import *
import ROOT

flow = SampleProcessing(
    "Simple Test", "/scratch/arizzi/0088F3A1-0457-AB4D-836B-AC3022A0E34F.root")


flow.SubCollection("GenMuon", "GenPart", "abs(GenPart_pdgId)==13")
flow.MatchDeltaR("GenMuon", "GenJet")
flow.Define("GenJet_goodGenMuonIdx",
            "ROOT::VecOps::Where(GenJet_GenMuonDr<0.4,GenJet_GenMuonIdx,-1)")

nthreads = 10
histos = {}
targets = ["nGenJet", "GenJet_pt", "GenJet_eta", "GenJet_phi", "GenJet_GenMuonDr",
           "GenJet_GenMuonIdx", "nGenMuon", "GenMuon_pt", "GenMuon_eta", "GenMuon_phi", "GenJet_goodGenMuonIdx"]
processor = flow.CreateProcessor(
    "eventProcessor", targets, histos, [], "", nthreads)


rdf = ROOT.RDataFrame(
    "Events", "/scratch/arizzi/0088F3A1-0457-AB4D-836B-AC3022A0E34F.root")
result = processor(rdf)

processed_rdf = result.rdf.find("").second
processed_rdf.Snapshot("Events", "out.root", targets)
```

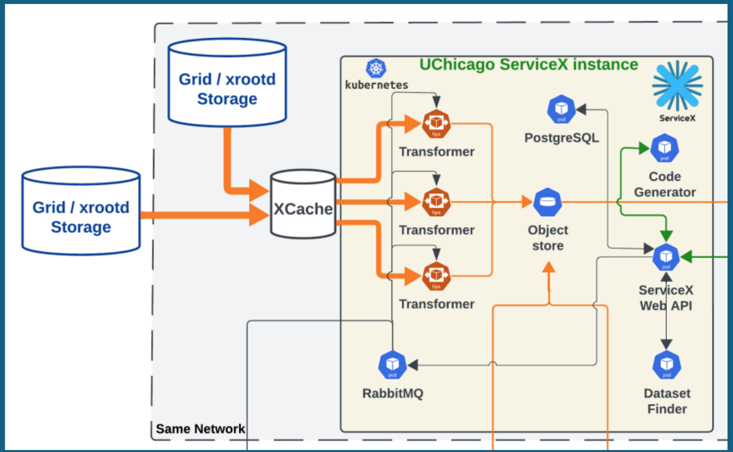
Sample generates a small output flat root-tuple from an input file

func_adl (Functional ADL) “DSL”



- Data Selection looks a lot like a SQL query
    - Use a *functional* form of SQL (from C# LINQ research)
- Built for data delivery
    - Not designed for histograms
- Builds a computational graph in python
    - Translates it to C++
- Translate from a proprietary format to flat ntuples or parquet files
    - ATLAS xAOD’s, for example
    - Demo on CMS miniaod
- Supplies data to awkward and pandas
    - Prototype of a RDF source
- Primary implementation uses ServiceX as backend
    - See [Poster here at ACAT]

ServiceX



From [poster]

DASK-Awkward Array “DSL”

awkard is the numpy of the python ecosystem for HEP data

```
from pathlib import Path
import awkward as ak

file = Path("data.00.json")
x = ak.from_json(file, line_delimited=Tru
x = x[ak.num(x.foo) > 2]
```

```
import dask_awkward as dak

# dask-awkward only supports line-delimite
x = dak.from_json("data.*.json")
x = x[dak.num(x.foo) > 2]

# With Dask we have to ask for the result
x = x.compute()
```

awkward

- Immediate/eager execution
- Run locally on-machine in memory

Allows for:

- Translation to other backends
- Editing the compute graph after creation
- Optimization of graph can be performed

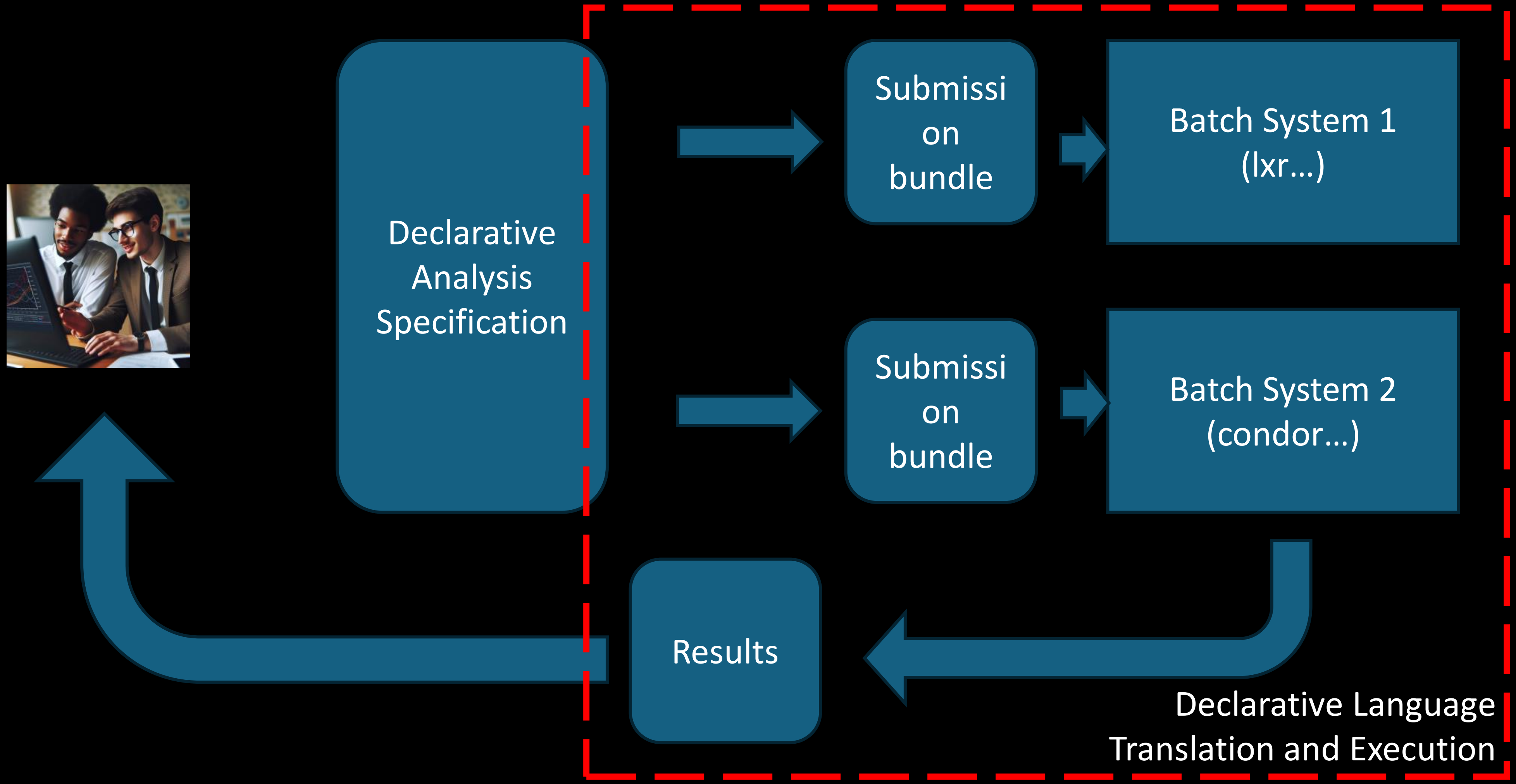
dask-awkward

- Build compute graph
- The .compute() runs the data
- Distributed (across cluster)

Challenges
&
Opportunities

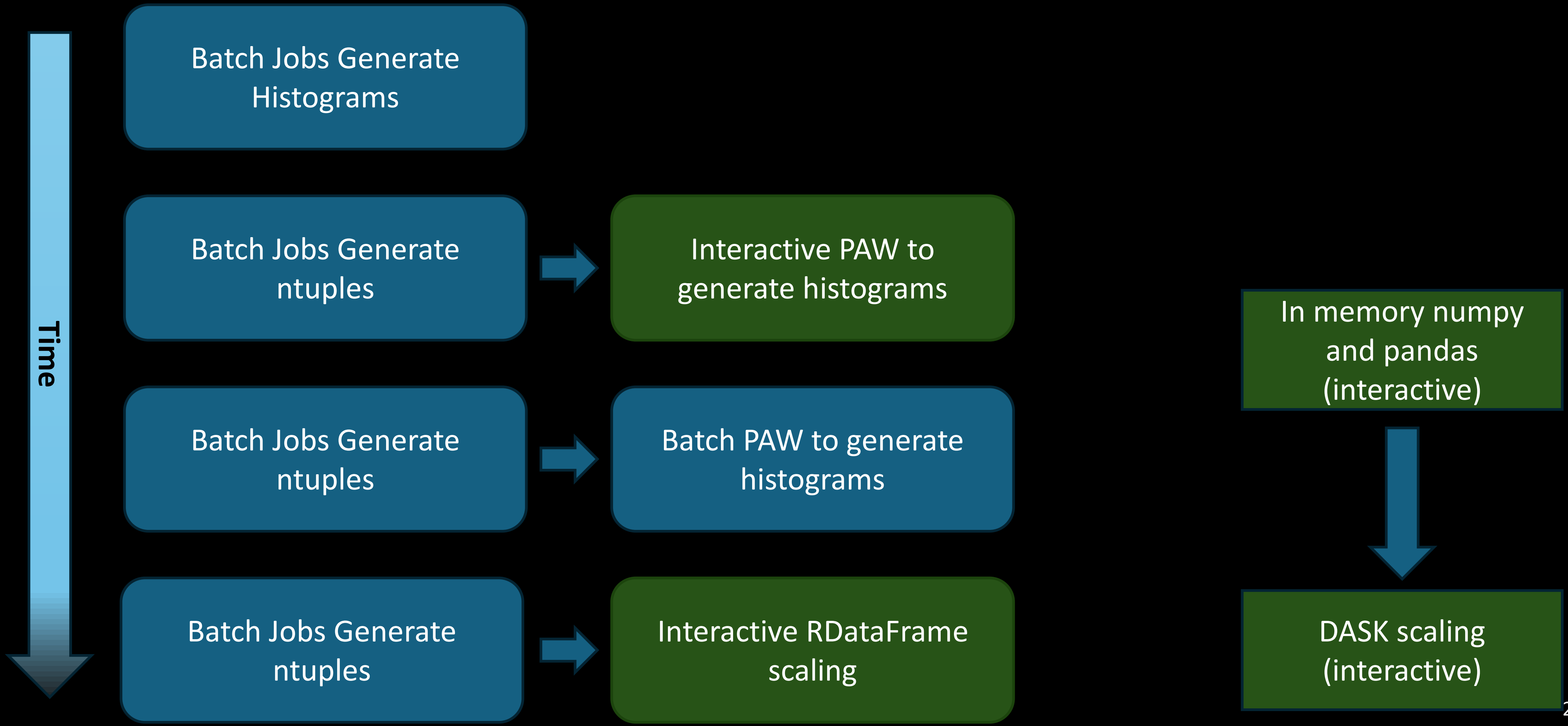
The Scalability Problem

Scaling systems aren’t written by DL authors!



The Scalability Problem

Interactivity isn't a new use case
It is just a new possibility

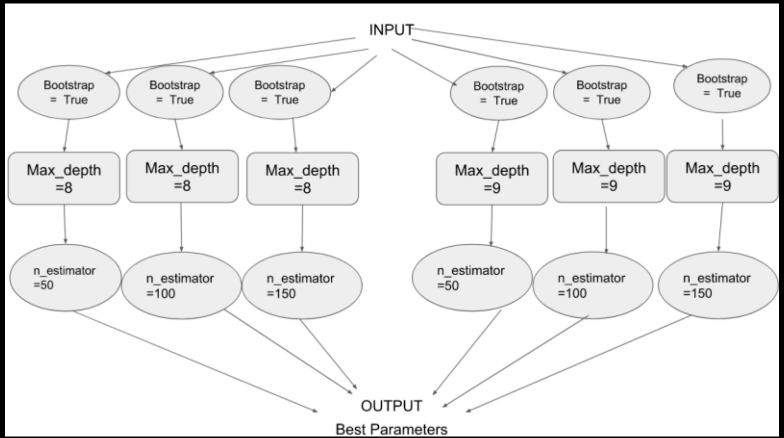


Possible Convergence

In the dask-awkward world…

- The DAG is known *before* any calculation is done
- Anyone can build it, or even edit it!

New opportunities for integration between languages and infrastructure.



Directed Acyclic Graph (DAG) of computations

- Integration with other ADL's
  - Selection cut text
  - Vertex building
  - Sample and control region definitions
- Unique optimizations
- Cross platform possibilities

Which Declarative Language Will Win?

Declarative Languages for Analysis

Compelling goals

- Reduce boiler plate
- Reduce time-to-insight
- Hide complexity of software and hardware

Field is moving away from boiler-plate and complexity

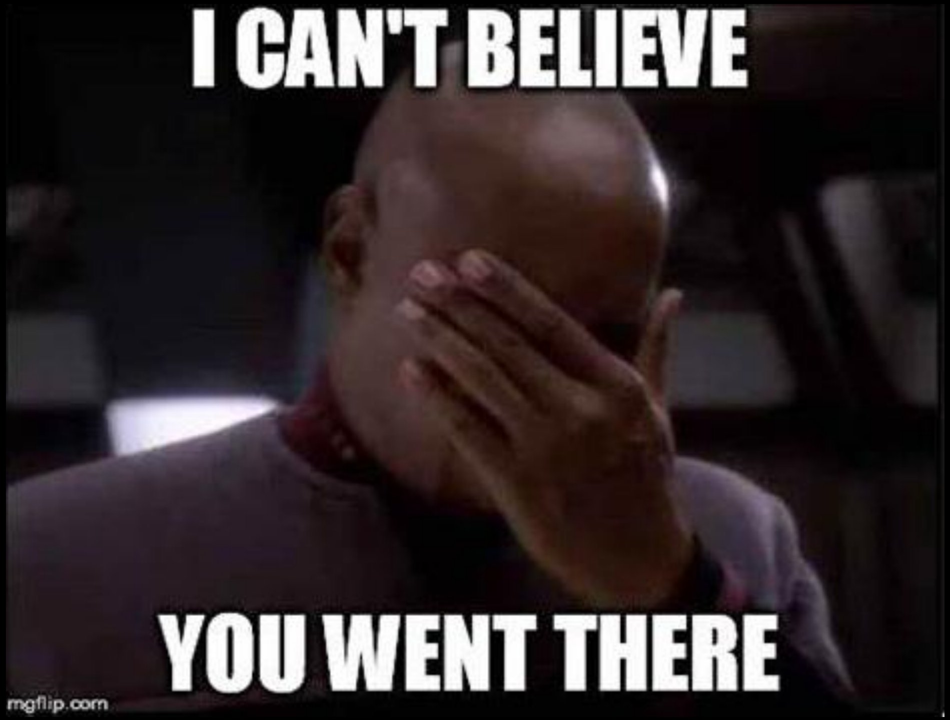
- It isn't just declarative languages that are pushing this!

Paths Forward

- Supporting Interactive Use Cases
- Deeper integration with abstract DAG's that are portable across machines
- Will this lead to deep multi-language integration with efficient execution?
  - I hope so!

Bonus: Large Language Models

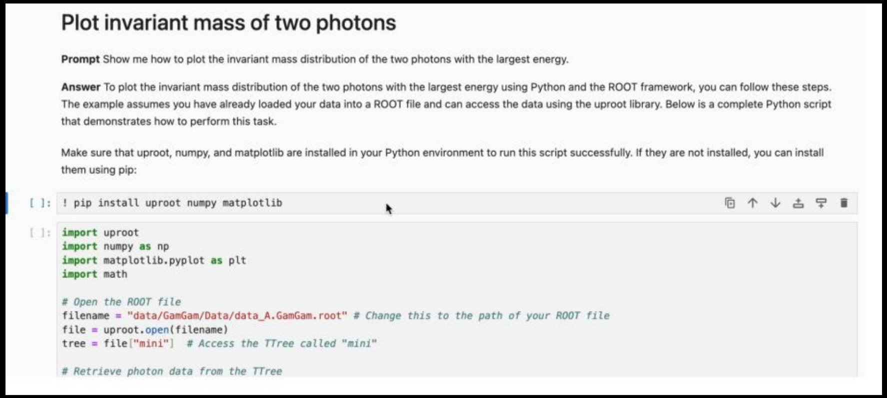
- Everything is in place
  - Code on slide ~3 was written by ChatGPT
  - Ability for the LLM to execute the code in a sandbox to test it
  - Integrated into a code editor
- So… we are just waiting…



Bonus: Large Language Models

- Everything is in place
  - Code on slide ~3 was written by ChatGPT
  - Ability for the LLM to execute the code in a sandbox to test it
  - Integrated into a code editor
- So… we are just waiting…

Educational Outreach with AI-Assisted CERN Open Data Analysis



Plot invariant mass of two photons

Prompt Show me how to plot the invariant mass distribution of the two photons with the largest energy.

Answer To plot the invariant mass distribution of the two photons with the largest energy using Python and the ROOT framework, you can follow these steps. The example assumes you have already loaded your data into a ROOT file and can access the data using the uproot library. Below is a complete Python script that demonstrates how to perform this task.

Make sure that uproot, numpy, and matplotlib are installed in your Python environment to run this script successfully. If they are not installed, you can install them using pip:

```
[ ]: ! pip install uproot numpy matplotlib
```

```
[ ]: import uproot
import numpy as np
import matplotlib.pyplot as plt
import math

# Open the ROOT file
filename = "data/GamGam/Data/data_A.GamGam.root" # Change this to the path of your ROOT file
file = uproot.open(filename)
tree = file["mini"]  # Access the TTree called "mini"

# Retrieve photon data from the TTree
```