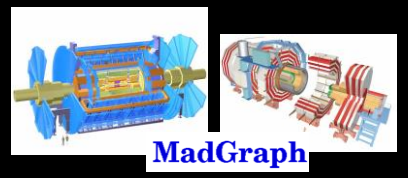
Using Declarative Languages for Analysis at the LHC

G. Watts (UW)

Institute for Research & Innovation in Software for High Energy Physics

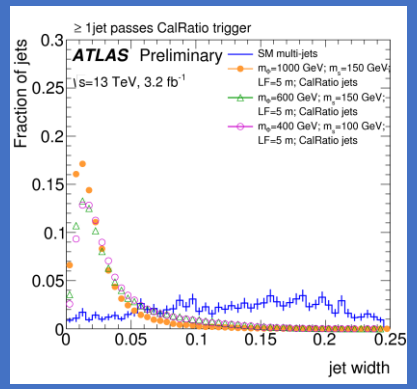
What is an Analysis Language?

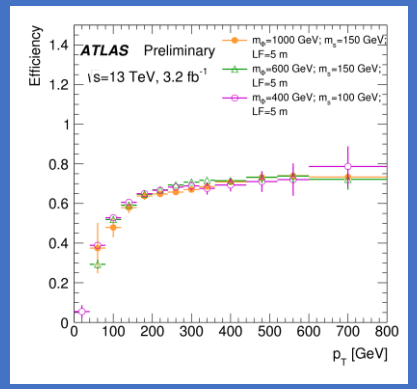


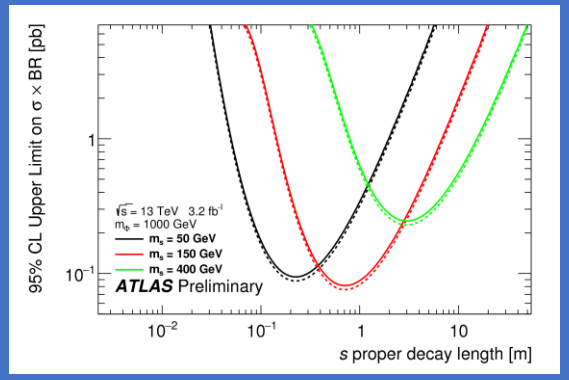
Raw Data Objects

Event Summary Data (ESD)

Analysis Object Data (AOD)







Analysis Language Tasks:

- Generate plots from a dataset
- Ratios from different datasets (efficiencies, etc.)
- Statistical Analysis (limit plots)
- Machine Learning
- Tables of numbers for publication

In use today:

- C++, ROOT
- Python, data science tools (including ROOT)
- Long tail: C#, Go, DSL's etc.

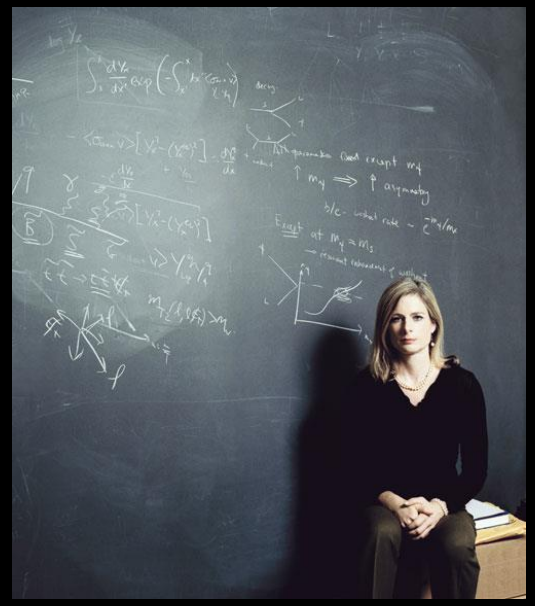
Can We Do Better?

What would a language look like explicitly designed for particle physics data?

A language that explicitly supported **both fast exploration and slower production?**

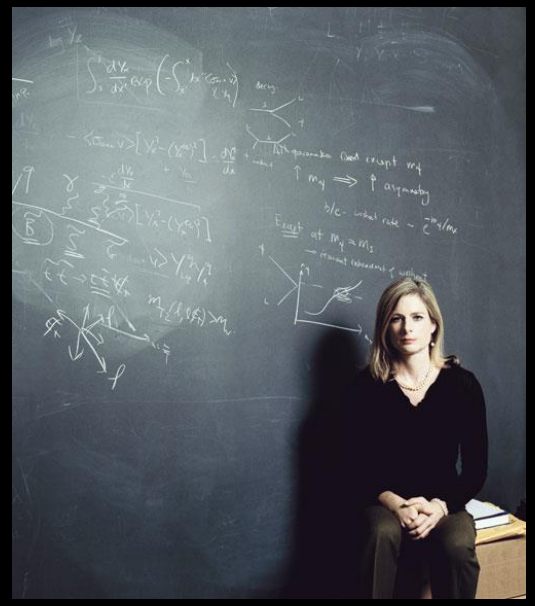
A language that could easily **scale from your laptop to a cluster** with minimal change (or knowledge) by the analyzer?

What is Declarative?



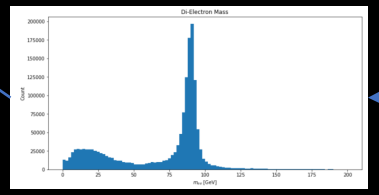
I want to see the invariant mass spectra of an e^+e^- pair from the $Z \rightarrow e^+e^-$ MC dataset

What is Declarative?



I want to see the invariant mass spectra of an e^+e^- pair from the $Z \rightarrow e^+e^-$ MC dataset

Magic Programming System



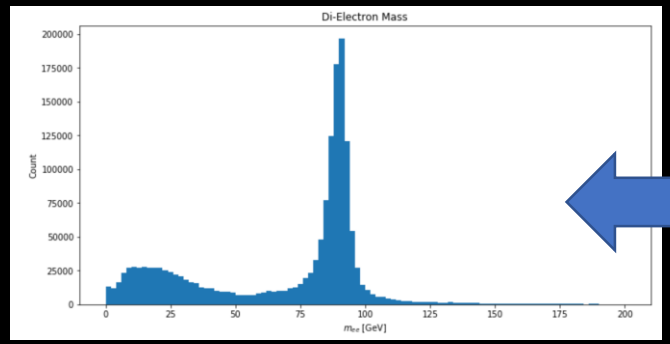
What is Declarative?

No loops, no boilerplate…

Fetching the data

github

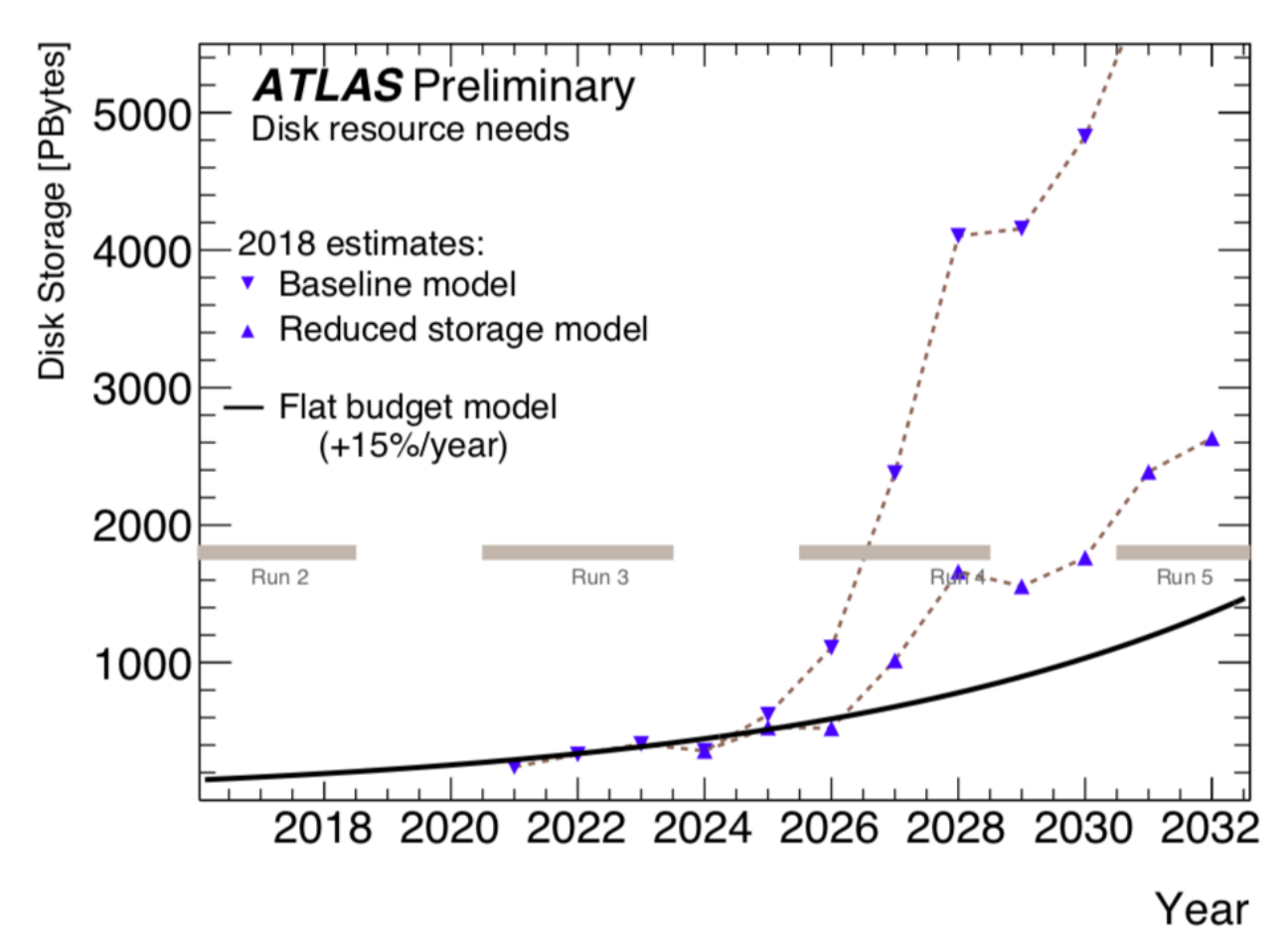
```
In [6]: ds = EventDataset('localds://mc15_13TeV.361106.PowhegPythia8EvtGen_AZNLOCTEQ6L1_Zee.merge.DAOD_STDM3.e3601_s2576_s2132_r6630_r6264_p2363_tid05630052_00')

In [7]: leptons_per_event_as = ds \
            .Select('lambda e: (e.Electrons("Electrons"), e.Muons("Muons"))') \
            .Select('lambda ls: (ls[0].Select(lambda e: e.pt()), ls[0].Select(lambda e: e.eta()), ls[0].Select(lambda e: e.phi()), ls[0].Select(lambda e: e.e()),ls[1].Select(lambda m: m.pt()), ls[1].Select(lambda m: m.eta()), ls[1].Select(lambda m: m.phi()), ls[1].Select(lambda m: m.e()))') \
            .AsAwkwardArray(('ElePt', 'EleEta', 'ElePhi', 'EleE', 'MuPt', 'MuEta', 'MuPhi', 'MuE')) \
            .future_value(executor=lambda a: use_exe_func_adl_server(a, node=end_point, quiet=False))

In [8]: leptons_per_event = await leptons_per_event_as
```




```
In [15]: v_particles = uproot_methods.TLorentzVectorArray.from_ptetaphi(
             leptons_per_event[b'ElePt'], leptons_per_event[b'EleEta'],
             leptons_per_event[b'ElePhi'], leptons_per_event[b'EleE'],
             )

In [17]: v_particles = v_particles[v_particles.counts >= 2]
         diparticles = v_particles[:, 0] + v_particles[:, 1]

In [39]: plt.figure(figsize=(12, 6))
         plt.hist(diparticles.mass/1000.0, bins=100, range=(0,200))
         plt.title('Di-Electron Mass')
         plt.xlabel('$m_{ee}$ [GeV]')
         plt.ylabel('Count')
         plt.show()
```

What is Declarative?

$$\frac{\text{Physics Content}}{\text{Boiler Plate}} \ll \text{Whatever It Is Now}$$


ATLAS Preliminary
Disk resource needs
2018 estimates:
Baseline model
Reduced storage model
Flat budget model
(+15%/year)
Disk Storage [PBytes]
5000
4000
3000
2000
1000
Run 2
Run 3
Run 4
Run 5
2018
2020
2022
2024
2026
2028
2030
2032
Year

Roles of the Host And Query Languages

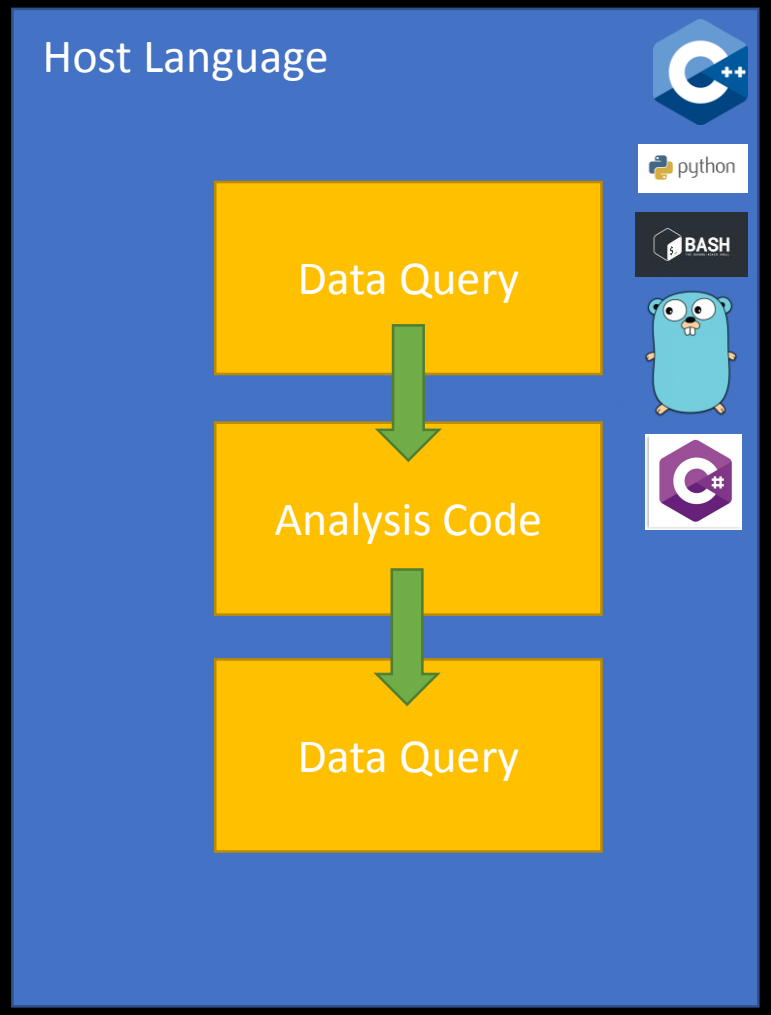
Data Query Language/Data Query Code Examples

- From Dataset #4553
- Select all electrons
- Form pairs
- Calculate invariant mass

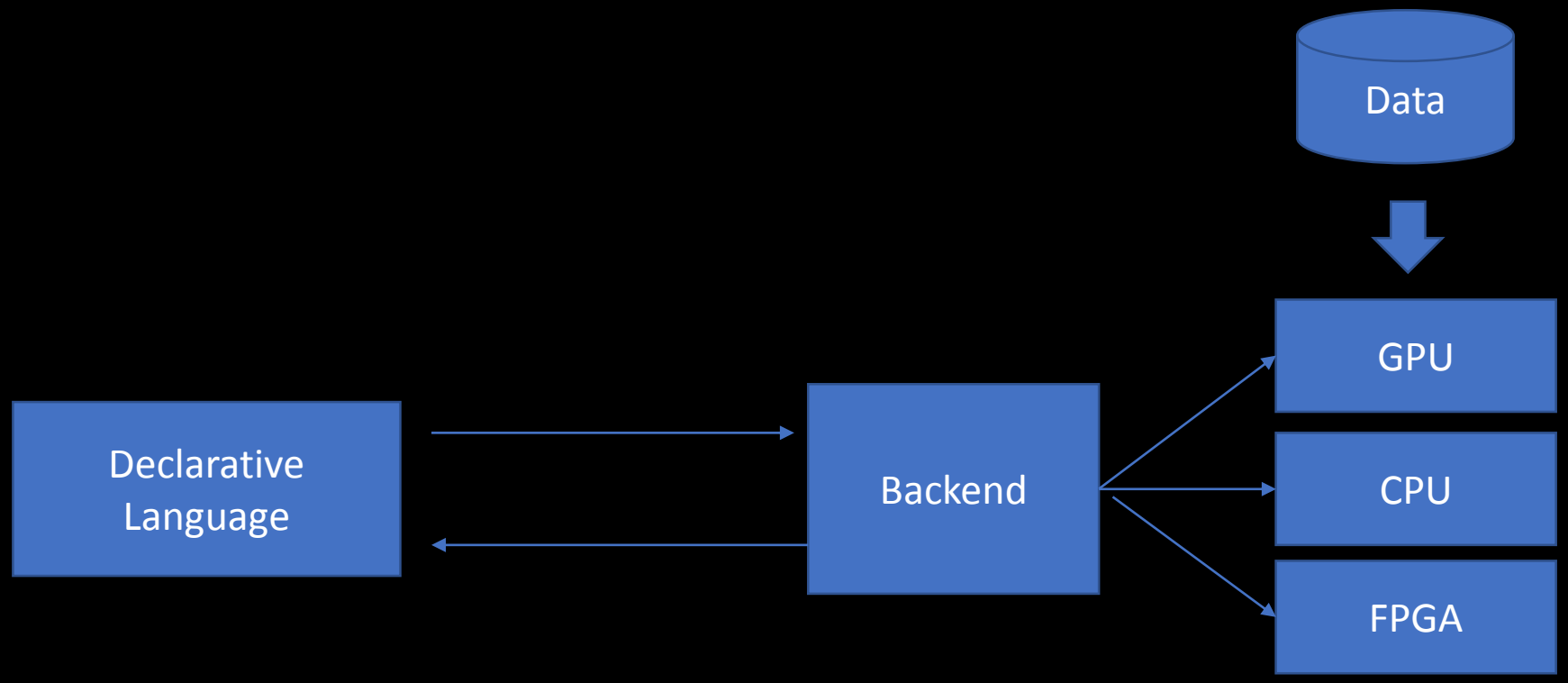
Host Language/Analysis Code Examples

- If statement on what defines a good/loose electron
- The η cut for electrons.
- Manipulation of final $Z \rightarrow ee$ invariant mass histogram

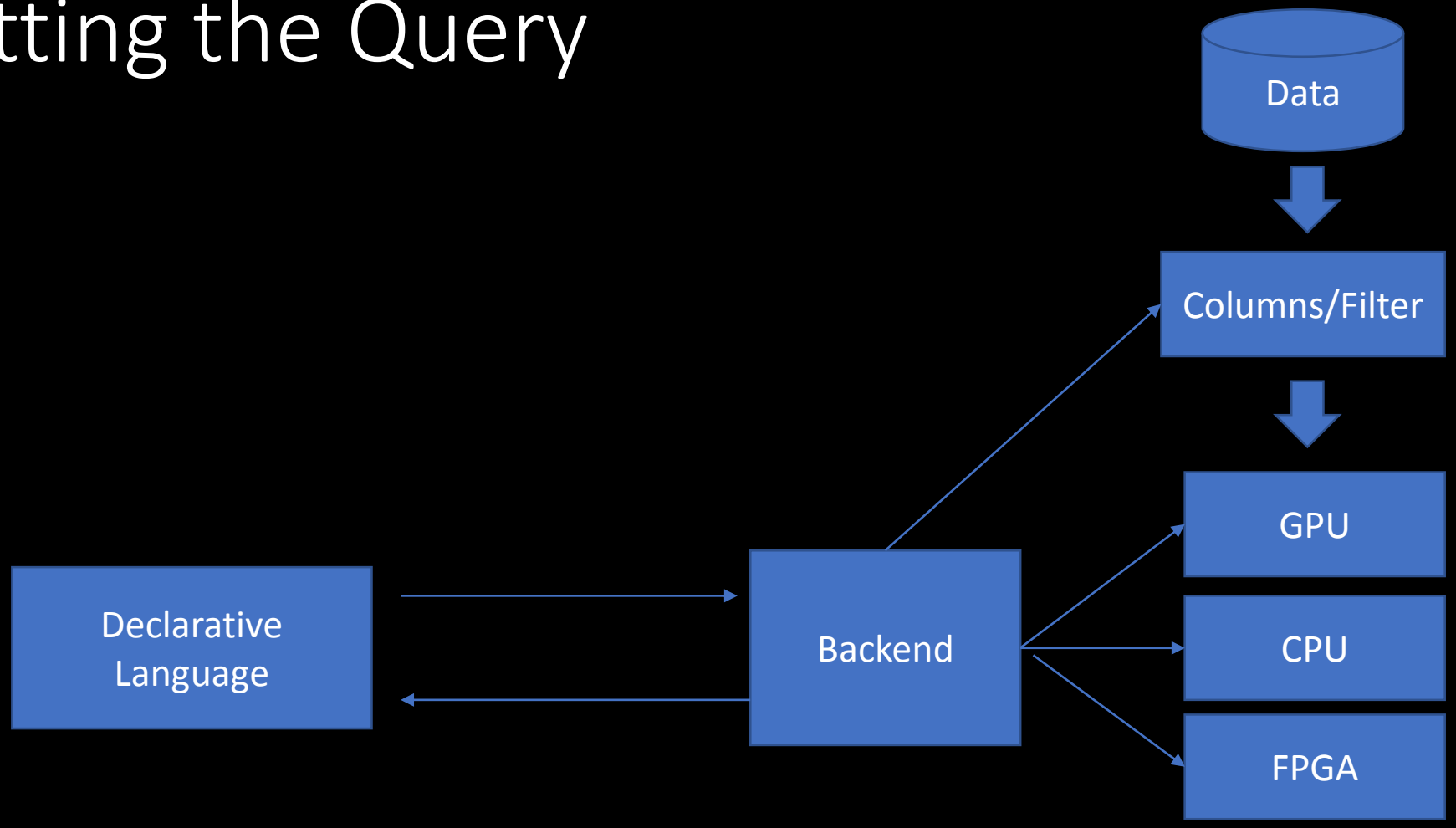
Text strings or part of the language?



Translating the Query

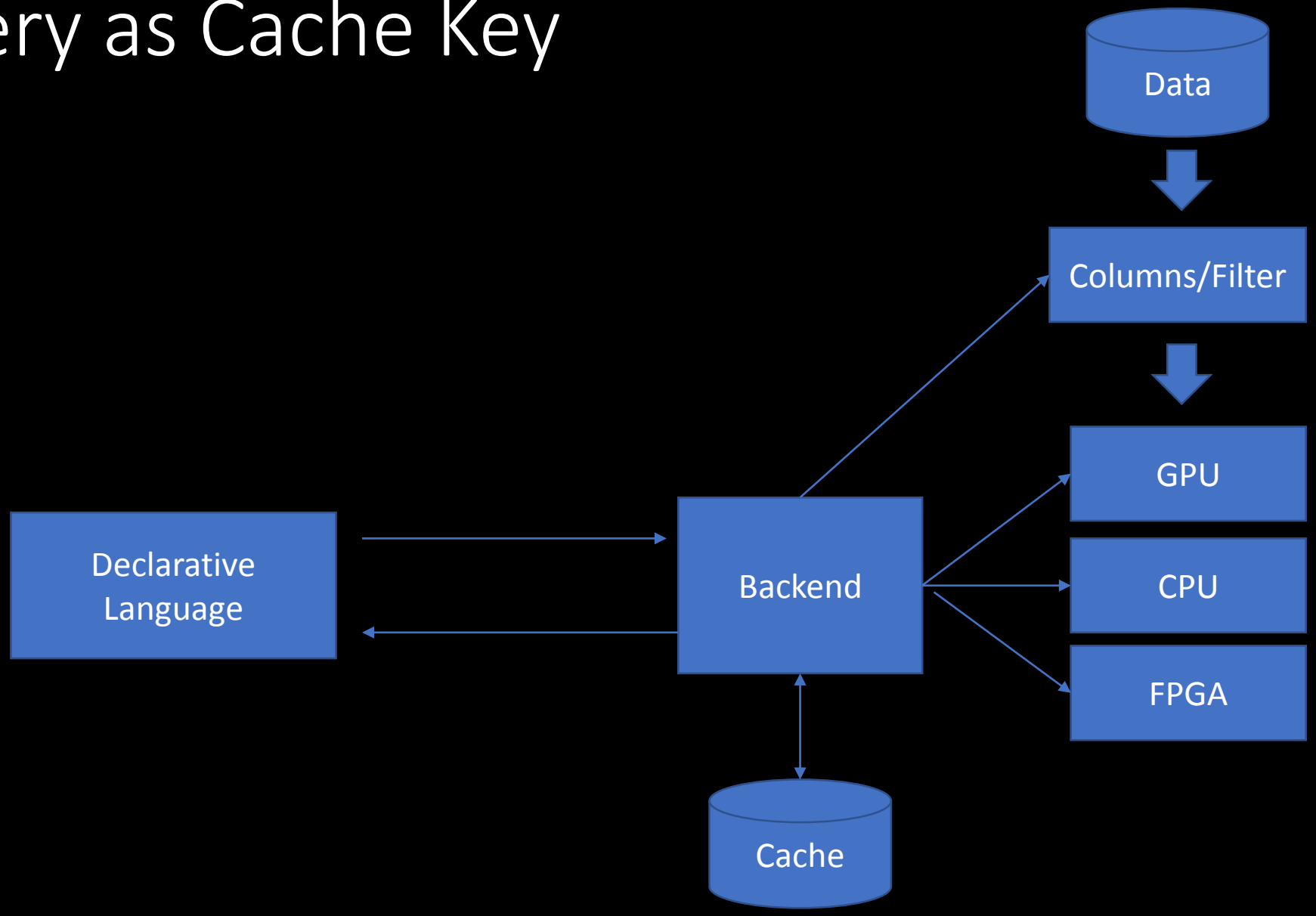


Splitting the Query



(ServiceX)

Query as Cache Key



(ServiceX)

Is Numpy Declarative?

```
jets[jets.pt > 50]
```

Slice operation

Good:
- No explicit loops
- Concise and clear

Not As Good:
- Immediate Execution
- All jets in dataset considered as one (not the way a physicist thinks)

Is Numpy Declarative?

```
jets[jets.pt > 50]
```

Slice operation

Good:
- No explicit loops
- Concise and clear

Not As Good:
- Immediate Execution
- All jets in dataset considered as one (not the way a physicist thinks)

Is Numpy Declarative?

```
jets[jets.pt > 50]
```

Slice operation

Good:
- No explicit loops
- Concise and clear

Not As Good:
- Immediate Execution
- All jets in dataset considered as one (not the way a physicist thinks)

HEP Data Query Challenges
(Poster, Thursday)

Data Query Language Must…

Set of common challenges that languages can implement.

Linked from the github repo are repos containing implementations for:

- RDF
- NAIL
- groot (Go)
- coffea

Functionality benchmarks

Github

1. Plot the missing E_T of all events.
2. Plot p_T of all jets in all events.
3. Plot p_T of jets with $|\eta| < 1$.
4. Plot the missing E_T of events that have at least two jets with $p_T > 40$ GeV.
5. Plot the missing E_T of events that have an opposite-sign muon pair with an invariant mass between 60 and 120 GeV.
6. Plot p_T of the trijet system with the mass closest to 172.5 GeV in each event and plot the maximum b-tagging discriminant value among the jets in the triplet.
7. Plot the sum of p_T of jets with $p_T > 30$ GeV that are not within 0.4 in ΔR of any lepton with $p_T > 10$ GeV.
8. For events with at least three leptons and a same-flavor opposite-sign lepton pair, find the same-flavor opposite-sign lepton pair with the mass closest to 91.2 GeV and plot the transverse mass of the missing energy and the leading other lepton.

Challenge 8

For events with at least three leptons and a same-flavor opposite-sign lepton pair, find the same-flavor opposite-sign lepton pair with the mass closest to 91.2 GeV and plot the transverse mass of the missing energy and the leading other lepton.

Root Data Frame (RDF)

Declarative

```
void rdataframe() {
    ROOT::EnableImplicitMT();
    ROOT::RDataFrame df("Events", "root://eospublic.cern.ch//eos/root-eos/benchmark/Run2012B_SingleMu.root");
    auto df2 = df.Filter("nElectron + nMuon > 2", "At least three leptons")
                 .Define("Lepton_pt", "Concatenate(Muon_pt, Electron_pt)")
                 .Define("Lepton_eta", "Concatenate(Muon_eta, Electron_eta)")
                 .Define("Lepton_phi", "Concatenate(Muon_phi, Electron_phi)")
                 .Define("Lepton_mass", "Concatenate(Muon_mass, Electron_mass)")
                 .Define("Lepton_charge", "Concatenate(Muon_charge, Electron_charge)")
                 .Define("Lepton_flavour", "Concatenate(ROOT::RVec<int>(nMuon, 0), ROOT::RVec<int>(nElectron, 1))")
                 .Define("AdditionalLepton_pt", additional_lepton_pt,
                         {"Lepton_pt", "Lepton_eta", "Lepton_phi", "Lepton_mass", "Lepton_charge", "Lepton_flavour"})
                 .Filter("AdditionalLepton_pt != -999", "No valid lepton pair found.");
    auto h1 = df2.Histo1D({"", ";MET (GeV);N_{Events}", 100, 0, 2000}, "MET_sumet");
    auto h2 = df2.Define("foo", "42")
                 .Histo1D({"", ";Lepton p_{T} (GeV);N_{Events}", 100, 15, 60}, "AdditionalLepton_pt");

    TCanvas c;
    c.Divide(2, 1);
    c.cd(1);
    h1->Draw();
    c.cd(2);
    h2->Draw();
    c.SaveAs("8_rdataframe.png");
}
```

Imperative

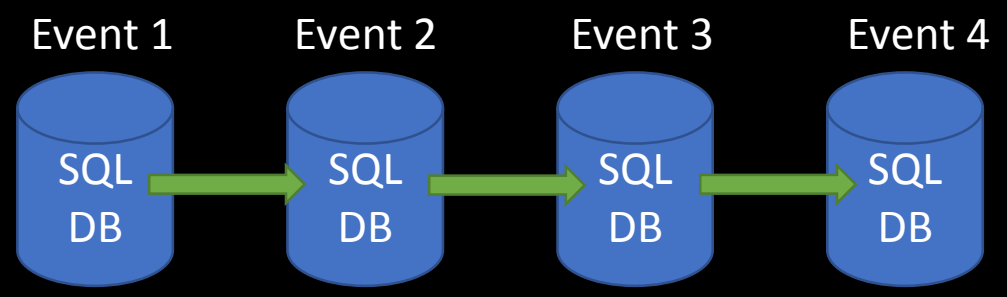
```
float additional_lepton_pt(Vec<float> pt, Vec<float> eta, Vec<float> phi, Vec<float> mass, Vec<int> charge, Vec<int> flavour)
{
    const auto c = Combinations(pt, 2);
    float lep_pt = -999;
    float best_mass = 99999;
    int best_i1 = -1;
    int best_i2 = -1;
    const auto z_mass = 91.2;
    const auto make_p4 = [&](std::size_t idx) {
        return ROOT::Math::PtEtaPhiMVector(pt[idx], eta[idx], phi[idx], mass[idx]);
    };

    for (auto i = 0; i < c[0].size(); i++) {
        const auto i1 = c[0][i];
        const auto i2 = c[1][i];
        if (charge[i1] == charge[i2]) continue;
        if (flavour[i1] != flavour[i2]) continue;
        const auto p1 = make_p4(i1);
        const auto p2 = make_p4(i2);
        const auto tmp_mass = (p1 + p2).mass();
        if (std::abs(tmp_mass - z_mass) < std::abs(best_mass - z_mass)) {
            best_mass = tmp_mass;
            best_i1 = i1;
            best_i2 = i2;
        }
    }

    if (best_i1 == -1) return lep_pt;

    for (auto i = 0; i < pt.size(); i++) {
        if (i != best_i1 && i != best_i2 && pt[i] > lep_pt) lep_pt = pt[i];
    }

    return lep_pt;
}
```

C++ Code

Belle II Decay Reconstruction Declarative Language

```
In [ ]: import basf2
        from modularAnalysis import *
        from variables.utils import create_aliases_for_selected
        file_name = "http://ekpwww.physik.uni-karlsruhe.de/~nbraun/B2JpsiKs_mu_BGx0_r1000.root"
        filelistSIG = [file_name]

        main_path = create_path()
        queue = handler.create_queue()

        inputMdstList("default", filelist=filelistSIG, path=main_path)

        fillParticleList('pi+:loose', 'piid > 0.01', path=main_path)
        fillParticleList('mu+:loose', 'muid > 0.01', path=main_path)

        # reconstruct Ks -> pi+ pi- decay
        # keep only candidates with 0.4 < M(pipi) < 0.6 GeV
        reconstructDecay('K_S0:pipi -> pi+:loose pi-:loose', '0.4 < M < 0.6', path=main_path)

        # reconstruct J/psi -> mu+ mu- decay
        # keep only candidates with 3.0 < M(mumu) < 3.2 GeV
        reconstructDecay('J/psi:mumu -> mu+:loose mu-:loose', '3.0 < M < 3.2', path=main_path)

        # reconstruct B0 -> J/psi Ks decay
        # keep only candidates with 5.2 < M(J/PsiKs) < 5.4 GeV
        reconstructDecay('B0:jspiks -> J/psi:mumu K_S0:pipi', '5.2 < M < 5.4', path=main_path)

        # perform B0 kinematic vertex fit using only the mu+ mu-
        # keep candidates only passing C.L. value of the fit > 0.0 (no cut)
        vertexRave('B0:jspiks', 0.0, 'B0 -> [J/psi -> ^mu+ ^mu-] K_S0', path=main_path)

        # build the rest of the event associated to the B0
        buildRestOfEvent('B0:jspiks', path=main_path)

        # perform MC matching (MC truth asociation). Always before TagV
```

Query In Python



A sequence of small “event” databases

Stream Processing

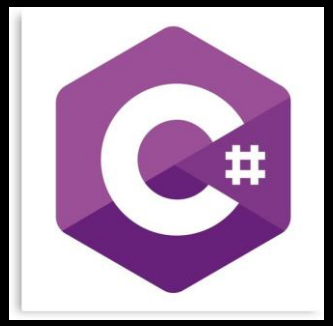
```
    event_info = events \
        .Select("lambda e: (e.EventInfo('EventInfo'), e.Jets('AntiKt4EMTopoJets'), e.TruthParticles('TruthParticles').Where(lambda t
p1: tp1.pdgId() == 35))")
    jet_info = event_info \
        .SelectMany(f'lambda ev: ev[1].Select(lambda j1: (ev[0], j1, ev[2].Where(lambda tp2: DeltaR(tp2.eta(), tp2.phi(), j1.eta(),
j1.phi()) < {deltar_llp})))')
```

Jets, with a MC particle with $\Delta R < 0.4$

- No explicit loops
- Explicit declaration of each relationship
- SQL-like language isn’t the best match for python

func_adl

Industry



Built SQL into the language as LINQ

- Functional implementation of SQL
- Code as Data
- Runs on all platforms
- Integrates data query and host language

(Embedded DSL)

Amazon's

PartiQL

- Standalone language
- SQL backwards compatible
- Explicitly supports nested data (unlike SQL)
- Syntax is… SQL, with a small number of extensions
- Designed to be agnostic to backend

(External DSL)

Roll Your Own!

Jim Pivarski's PartiQL

This is the same challenge #8

Note how concise this is

```
leptons = electrons union muons

cut count(leptons) >= 3 named "three_leptons" {

   Z = electrons as (lep1, lep2) union muons as (lep1, lep2)
         where lep1.charge != lep2.charge
         min by abs(mass(lep1, lep2) - 91.2)

   third = leptons except [Z.lep1, Z.lep2] max by pt

   hist third.pt by regular(100, 0, 250) named "third_pt" titled "Leading other lepton pT"
}
```

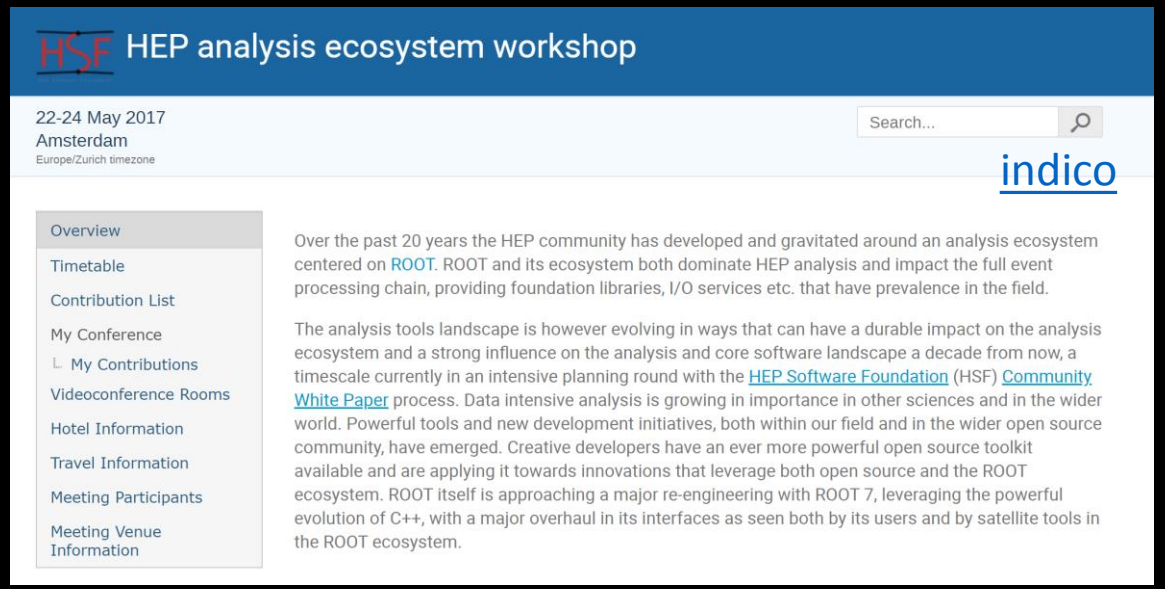
Workshops

Two large workshops:

- ADL, CutLang, lhada2rivit, LINQ, Yaml as an ADL, NAIL, TTreeFormual, AEACUS and RHADAMANTUS

Here at CHEP:

- F.A.S.T. (yaml) – Monday, Nov 4th
- COFFEA - Columnar Object Framework For Effective Analysis – This session
- A Functional Declarative Analysis Language in Python – Tuesday Poster
- HEP Data Query Challenges – Thursday Poster
- Striped Data Analysis Framework – Thursday Poster
- See my talk at the WLCG/HSF workshop on Analysis Eco-systems challenges



iris hep
Institute for Research & Innovation in Software for High Energy Physics

Spring 2020?

Conclusions

- Lots to potentially gain from a well implement declarative language
  - Improve time-to-insight
  - Potentially burry experiment’s data access issues
- Your host language needs to support the data query language directly
  - Alternative is text strings
  - Lack of composition (text strings!?)
  - Languages support the features needed to varying degrees
  - C# is great, python is ok, and C++ is tough
- Concepts go hand-in-hand for HL-LHC

Backup

Go

Not declarative… (though the language could support that)

Inherently distributed!

Event Loop is Explicit

```go
package main

import (
    "fmt"

    "github.com/pkg/errors"
    "go-hep.org/x/hep/groot"
    "go-hep.org/x/hep/groot/rtree"
    "go-hep.org/x/hep/hbook"
    "go-hep.org/x/hep/hplot"
    "gonum.org/v1/plot/vg"
)

// basic1 plots the MET in an event.
func basic1(fname string) error {
    f, err := groot.Open(fname)
    if err != nil {
        return errors.Wrap(err, "could not open ROOT file")
    }
    defer f.Close()

    o, err := f.Get("Events")
    if err != nil {
        return errors.Wrap(err, "could not retrieve tree")
    }

    tree := o.(rtree.Tree)
    fmt.Printf("tree: %d entries\n", tree.Entries())

    var (
        hmet = hbook.NewH1D(100, 0, 2000)
    )

    sc, err := rtree.NewTreeScannerVars(tree, rtree.ScanVar{Name: "MET_sumet"})
    if err != nil {
        return errors.Wrap(err, "could not create scanner")
    }
    defer sc.Close()

    for sc.Next() {
        var met float32
        err := sc.Scan(&met)
        if err != nil {
            return errors.Wrap(err, "error during scan")
        }

        hmet.Fill(float64(met), 1)
    }

    if err := sc.Err(); err != nil {
        return errors.Wrap(err, "could not scan whole file")
    }

    fmt.Printf("hmet: %v\n", hmet.SumW())
```

Challenge 1

Plot the missing E_T of all events.

Root Data Frame (RDF)

```
void rdataframe() {
    ROOT::EnableImplicitMT();
    ROOT::RDataFrame df("Events",
        "root://eospublic.cern.ch//eos/root-eos/benchmark/Run2012B_SingleMu.root");
    auto h = df.Histo1D({"", ";MET (GeV);N_{Events}", 100, 0, 2000}, "MET_sumet");

    TCanvas c;
    h->Draw();
    c.SaveAs("1_rdataframe.png");
}
```

Text leaf names

Implied Loops

After the workshop, people sent me a few more items to include. The slides after this one are the results of that.

Common Analysis Framework (CAF)

From ATLAS

I just listened to your talk on declarative languages in HEP. I found it super interesting and I agree that this is exactly what we need.

During my PhD, I was working on the ATLAS HWW analysis, where we developed a very general analysis software framework that we call "CAF"
(Common Analysis Framwork). This is now used by quite a few analyes in ATLAS (HWW, Htautau, SMWW, and a few smaller analyses in the exotics
group) The idea behind that framework is to basically allow users to to everything they want by just editing config files, without ever having to touch the backend C++ or python.

Of course, our config file syntax is basically a declarative DSO. As I have been one of the main developers of the framework, I'm happy to provide more details if you are interested - so please feel free to follow up. Here's a pointer to our documentation:

https://atlas-caf-tutorial.web.cern.ch/

Regards,

Carsten

LHCb Trigger Language

https://indico.cern.ch/event/773049/contributions/3473309/attachments/1937519/3211534/CHEP_configuration_talk_1.pdf

In particular slide 13 for the selections. Obviously currently targeting reconstruction and trigger selections, but easily extendable

Implementation of cuts and loops is c++ functors with some magic to interface to python.

Marco

Trigger lines and control flow

```
@register_line_builder(all_lines)
def dzero2Kpi_line(name='Hlt2CharmHadD02KmPipLine'):
    long_tracks = make_long_tracks()
    kaons = make_kaons(long_tracks, Cut=PID_K > 3)
    pions = make_pions(long_tracks, Cut=PID_K < 3)
    dzeros = make_dzeros(
        particles=[kaons, pions],
        decays=['[D0 -> K- pi+]cc'])
    return HltLine(
        name=name,
        children=[gec, kaons, pions, dzeros],
        prescale=1)
```

Select Kaons & Pions

Defines a line

Combine them to D^0 mesons

Control flow: Short circuited AND
If one selects 0 particles
→ we stop right there