Declarative framework for analysis definition and implementation

Alberto Annovi, Tommaso Boccali,
Paolo Mastrandrea, Andrea Rizzi

(INFN & Università di Pisa)

ICSC-S2 Annual Meeting - Catania, 11/12/2024

Target: improve (HEP) data analysis tasks

Analysis DEVELOPMENT

Speed
fast analysis development

Portability
same analysis for *different* datasets / experiments

Preservation
reproducibility of the results

Analysis PERFORMANCE

Flexibility
simple adoption of cutting-edge tools / languages / submission systems

Parallelization
simple parallelization of the tasks

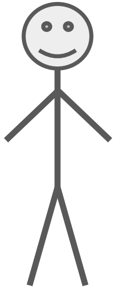
Optimization
support for automatic (technical) optimization

Deeper decoupling
between algorithm and implementation

Better scaling
with algorithm complexity and data size

Declarative paradigms

How: framework structure



Frontend : Analysis flow

*Definition of the task using a standard set of **declarations**.*

*Analysis flow described with a **Directed Acyclic Graph** (**DAG**).*

example

- `Select 2 muons`
- `Get the dimuon invariant mass distribution`

Flow definition
(one JSON file)

Backend : Processor building

Generation of the analysis implementation for specific:

- ***tool / language / system***
- ***dataset / experiment***

example

`RDF processor for NANOAOD samples`

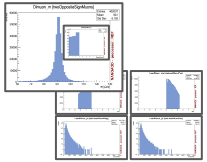
Input dataset

Submission system

example

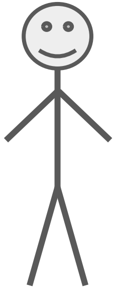
`Local`
`Grid`
`Dask`
`…`

Processor



How: framework structure

Speed



Frontend : Analysis flow

Definition of the task using a standard set of ***declarations****.*

Analysis flow described with a ***Directed Acyclic Graph*** *(****DAG****).*

example

- *Select 2 muons*
- *Get the dimuon invariant mass distribution*

Flow definition

(one JSON file)

Backend : Processor building

Generation of the analysis implementation for specific:

- ***tool / language / system***
- ***dataset / experiment***

example

RDF *processor for* ***NANOAOD*** *samples*

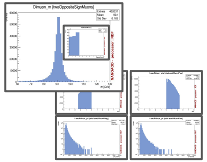
Input dataset

Submission system

example

Local
Grid
Dask
…

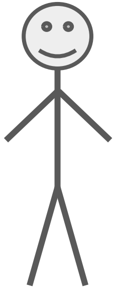
Processor



Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA | ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

How: framework structure

Speed



Frontend : Analysis flow

Definition of the task using a standard set of ***declarations****.*

Analysis flow described with a ***Directed Acyclic Graph*** *(****DAG****).*

example

- *Select 2 muons*
- *Get the dimuon invariant mass distribution*

Portability

Flow definition
(one JSON file)

Preservation

Backend : Processor building

Generation of the analysis implementation for specific:

- ***tool / language / system***
- ***dataset / experiment***

example

RDF *processor for* ***NANOAOD*** *samples*

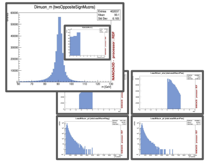
Input dataset

Submission system

example

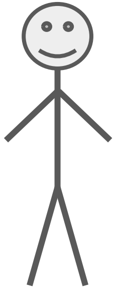
Local
Grid
Dask
…

Processor



How: framework structure

Speed



Frontend : Analysis flow

Definition of the task using a standard set of ***declarations****.*

Analysis flow described with a ***Directed Acyclic Graph*** *(****DAG****).*

example

- *Select 2 muons*
- *Get the dimuon invariant mass distribution*

Portability

Flow definition
(one JSON file)

Preservation

Optimization

Backend : Processor building

Generation of the analysis implementation for specific:

- ***tool / language / system***
- ***dataset / experiment***

example

RDF *processor for* ***NANOAOD*** *samples*

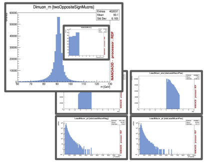
Input dataset

Submission system

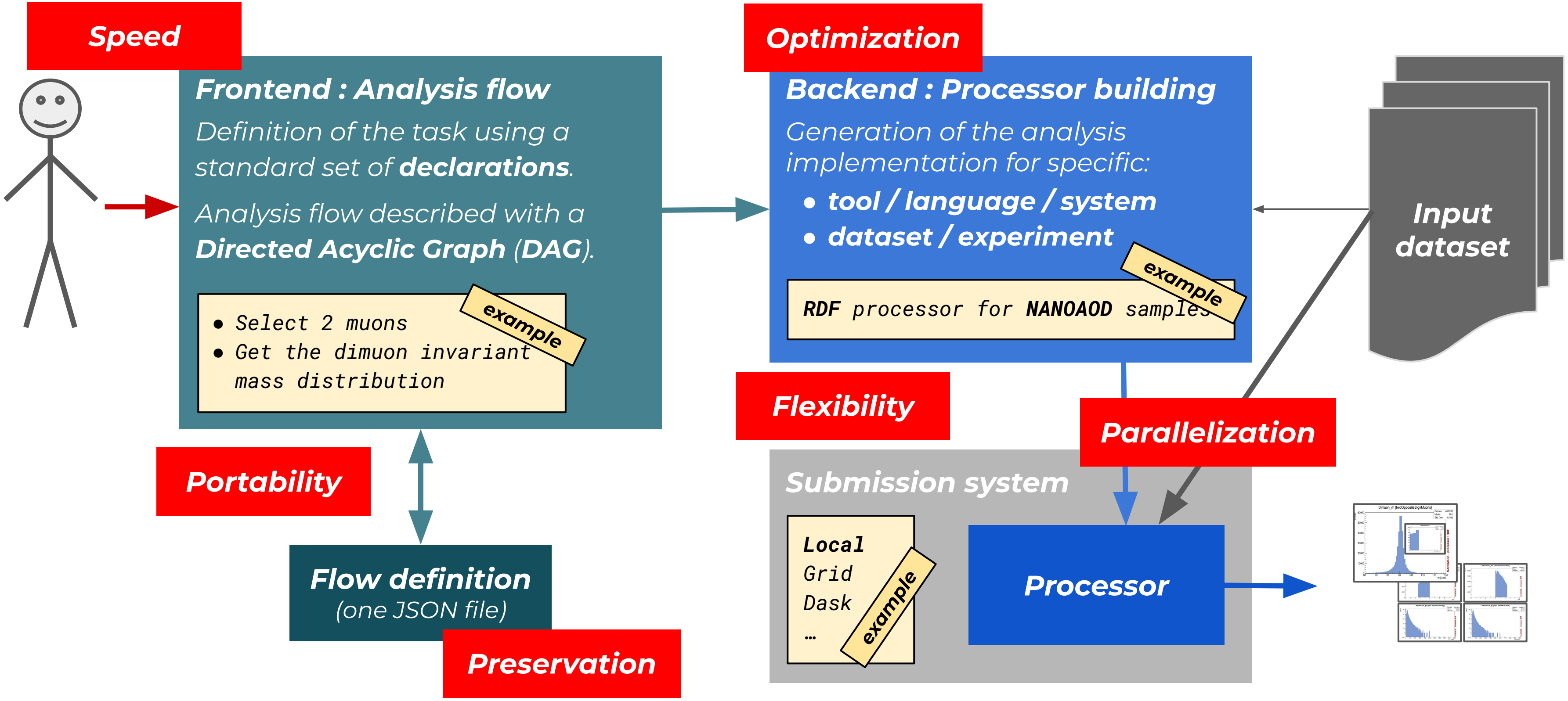
example

Local
Grid
Dask
…

Processor



How: framework structure



Demonstrator development

- ***Toolbox*** supporting declarative approach for HEP analysis
- re-implementation from [***NAIL***] (improved modularity)
- stand-alone Python package:
  - ***DAG handling***
  - ***Sample Processing*** : event loop ***flow*** definition
  - ***Interface Dictionary*** : translation of input naming
  - Backend ***processors*** (for event loop):
    1. Basic ***Loop*** *processor* (C++ compiled)
    2. ***RDF****-based processor* (C++ compiled - Multi-thread support)
    3. ***Direct python*** processor

(***Available*** / ***Under development***)

Extension →

Multiple-input data formats

Same flow on different-format datasets (input naming translation)

Supported data-formats:

- **NANOAOD** (CMS) ***Full***
- **PHYSLITE** (ATLAS) ***Preliminary***

Extension ↓

Full analysis chain

Extend the flow definition to procedure incorporating all the steps needed to extract the result of a complex analysis task

Demonstrator github repository

https://github.com/ICSC-Spoke2-repo/nail-dev

Definition set example

Example of a set of definitions for an "event loop" analysis flow

Define	define a new variable based on available inputs
SubCollection	define all the variables in a collection for selected candidates
Distinct	define pairs from a collection
TakePair	define a specific pair from a set of pairs from "Distinct" definition
ObjectAt	define an object from a collection (specific index)
Selection	define a selection (T/F value) - "Regions" as a group of Selections
DefineEventWeight	define a multiplicative event weight - applied for events in a region
DefineHisto1D	define a 1D histogram

To be added: definitions for handling of systematic uncertainties

Example code - flow

- Definition of tools/base variables
- Selection of 2 opposite-charge muons
- Evaluation of the dimuon invariant mass
- Definition of 2 regions:
  - Leading-muon eta > 0
  - Leading-muon eta <= 0
- Distributions:
  - # selected muons
  - dimuon invariant mass
  - Leading-muon p_T (per region)
  - Leading-muon eta (per region)
- Generate the processor & run

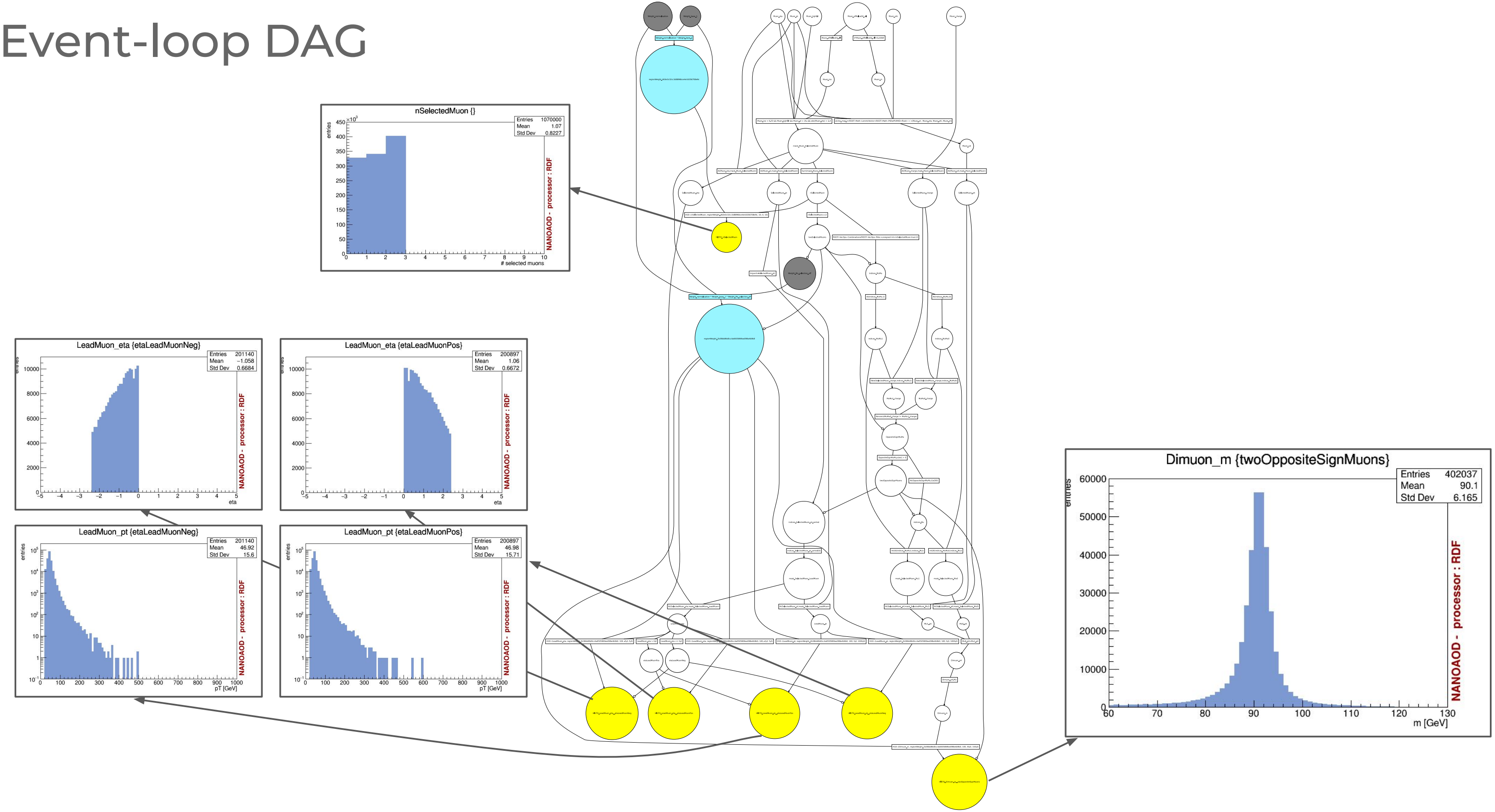
```
from eventFlow import *
from processorRDF import *

flow = SampleProcessing("flowTest", 'dictionaries/nanoAOD_nanoAOD_id_OpenData.json')

flow.DefineEventWeight("Weight_normalisation",    "1.0f")
flow.DefineEventWeight("Weight_base_1",           "1.0f")

flow.Define("Muon_m",    "0*Muon_pfRelIso04_all+0.1056f")
flow.Define("Muon_p4",
            "vector_map_t<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > >(Muon_pt, Muon_eta, Muon_phi, Muon_m)")
flow.Define("Muon_iso", "Muon_pfRelIso04_all")

flow.SubCollection("SelectedMuon", "Muon", sel="Muon_iso < 0.25 && Muon_tightId && Muon_pt > 20. && abs(Muon_eta) < 2.4")

flow.Selection("twoSelectedMuons", "nSelectedMuon==2")

flow.DefineEventWeight("Weight_Mu_selection_eff", "0.95f", requires=["twoSelectedMuons"])

flow.Distinct("MuMu", "SelectedMuon", requires=["twoSelectedMuons"])

flow.Define("OppositeSignMuMu", "Nonzero(MuMu0_charge != MuMu1_charge)", requires=["twoSelectedMuons"])

flow.Selection("twoOppositeSignMuons", "OppositeSignMuMu.size() > 0")

flow.TakePair("Mu", "SelectedMuon", "MuMu", "At(OppositeSignMuMu,0,-200)", requires=["twoOppositeSignMuons"])

flow.Define("Dimuon_p4", "Mu0_p4+Mu1_p4")
flow.Define("Dimuon_m",  "Dimuon_p4.M()")

flow.Define("indices_SelectedMuon_pt_sorted", "Argsort(-SelectedMuon_pt)", requires=["twoOppositeSignMuons"])
flow.ObjectAt("LeadMuon", "SelectedMuon", "indices_SelectedMuon_pt_sorted[0]")
flow.ObjectAt("SubMuon",  "SelectedMuon", "indices_SelectedMuon_pt_sorted[1]")

flow.Selection("etaLeadMuonPos", "LeadMuon_eta > 0.0")
flow.Selection("etaLeadMuonNeg", "LeadMuon_eta <= 0.0")

flow.DefineHisto1D("nSelectedMuon", [], 10, 0, 10)
flow.DefineHisto1D("Dimuon_m", ["twoOppositeSignMuons"], 100, 50.0, 150.0)

flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonPos'], 100, 0.0, 1000.0)
flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonNeg'], 100, 0.0, 1000.0)

flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonPos'], 100, -5.0, 5.0)
flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonNeg'], 100, -5.0, 5.0)

flow.BuildFlow()

targetList = ["HISTO_nSelectedMuon",
              "HISTO_Dimuon_m__twoOppositeSignMuons",
              "HISTO_LeadMuon_pt__etaLeadMuonPos",
              "HISTO_LeadMuon_pt__etaLeadMuonNeg",
              "HISTO_LeadMuon_eta__etaLeadMuonPos",
              "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)

pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```

Example code - flow

- Definition of tools/base variables
- Selection of 2 opposite-charge muons
- Evaluation of the dimuon invariant mass
- Definition of 2 regions:
  - Leading-muon eta > 0
  - Leading-muon eta <= 0
- Distributions:
  - # selected muons
  - dimuon invariant mass
  - Leading-muon p_T (per region)
  - Leading-muon eta (per region)
- Generate the processor & run

```
from eventFlow import *
from processorRDF import *

flow = SampleProcessing("flowTest", 'dictionaries/nanoAOD_nanoAOD_id_OpenData.json')

flow.DefineEventWeight("Weight_normalisation",   "1.0f")
flow.DefineEventWeight("Weight_base_1",          "1.0f")

flow.Define("Muon_m",   "0*Muon_pfRelIso04_all+0.1056f")
flow.Define("Muon_p4",
            "vector_map_t<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > >(Muon_pt, Muon_eta, Muon_phi, Muon_m)")
flow.Define("Muon_iso", "Muon_pfRelIso04_all")

flow.SubCollection("SelectedMuon", "Muon", sel="Muon_iso < 0.25 && Muon_tightId && Muon_pt > 20. && abs(Muon_eta) < 2.4")

flow.Selection("twoSelectedMuons", "nSelectedMuon==2")

flow.DefineEventWeight("Weight_Mu_selection_eff", "0.95f", requires=["twoSelectedMuons"])

flow.Distinct("MuMu", "SelectedMuon", requires=["twoSelectedMuons"])

flow.Define("OppositeSignMuMu", "Nonzero(MuMu0_charge != MuMu1_charge)", requires=["twoSelectedMuons"])

flow.Selection("twoOppositeSignMuons", "OppositeSignMuMu.size() > 0")

flow.TakePair("Mu", "SelectedMuon", "MuMu", "At(OppositeSignMuMu,0,-200)", requires=["twoOppositeSignMuons"])

flow.Define("Dimuon_p4", "Mu0_p4+Mu1_p4")
flow.Define("Dimuon_m",  "Dimuon_p4.M()")

flow.Define("indices_SelectedMuon_pt_sorted", "Argsort(-SelectedMuon_pt)", requires=["twoOppositeSignMuons"])
flow.ObjectAt("LeadMuon", "SelectedMuon", "indices_SelectedMuon_pt_sorted[0]")
flow.ObjectAt("SubMuon",  "SelectedMuon", "indices_SelectedMuon_pt_sorted[1]")

flow.Selection("etaLeadMuonPos", "LeadMuon_eta > 0.0")
flow.Selection("etaLeadMuonNeg", "LeadMuon_eta <= 0.0")

flow.DefineHisto1D("nSelectedMuon", [], 10, 0, 10)
flow.DefineHisto1D("Dimuon_m", ["twoOppositeSignMuons"], 100, 50.0, 150.0)

flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonPos'], 100, 0.0, 1000.0)
flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonNeg'], 100, 0.0, 1000.0)

flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonPos'], 100, -5.0, 5.0)
flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonNeg'], 100, -5.0, 5.0)

flow.BuildFlow()

targetList = ["HISTO_nSelectedMuon",
              "HISTO_Dimuon_m__twoOppositeSignMuons",
              "HISTO_LeadMuon_pt__etaLeadMuonPos",
              "HISTO_LeadMuon_pt__etaLeadMuonNeg",
              "HISTO_LeadMuon_eta__etaLeadMuonPos",
              "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)

pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```

Event-loop DAG



Example code - flow

- Definition of tools and first derived vars
- Selection of 2 opposite-charge muons
- Evaluation of the dimuon invariant mass
- Definition of 2 regions:
  - Leading-muon eta > 0
  - Leading-muon eta <= 0
- Distributions:
  - Leading-muon p_T (per region)
  - Leading-muon eta (per region)
- Generate the processor & run

```
from eventFlow import *
from processorRDF import *

flow = SampleProcessing("flowTest", 'dictionaries/nanoAOD_nanoAOD_id_OpenData.json')

flow.DefineEventWeight("Weight_normalisation",    "1.0f")
flow.DefineEventWeight("Weight_base_1",           "1.0f")

flow.Define("Muon_m",    "0*Muon_pfRelIso04_all+0.1056f")
flow.Define("Muon_p4",
             "vector_map_t<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > >(Muon_pt, Muon_eta, Muon_phi, Muon_m)")
flow.Define("Muon_iso",  "Muon_pfRelIso04_all")

flow.SubCollection("SelectedMuon", "Muon", sel="Muon_iso < 0.25 && Muon_tightId && Muon_pt > 20. && abs(Muon_eta) < 2.4")

flow.Selection("twoSelectedMuons", "nSelectedMuon==2")

flow.DefineEventWeight("Weight_Mu_selection_eff", "0.95f", requires=["twoSelectedMuons"])

flow.Distinct("MuMu", "SelectedMuon", requires=["twoSelectedMuons"])

flow.Define("OppositeSignMuMu", "Nonzero(MuMu0_charge != MuMu1_charge)", requires=["twoSelectedMuons"])

flow.Selection("twoOppositeSignMuons", "OppositeSignMuMu.size() > 0")

flow.TakePair("Mu", "SelectedMuon", "MuMu", "At(OppositeSignMuMu,0,-200)", requires=["twoOppositeSignMuons"])

flow.Define("Dimuon_p4", "Mu0_p4+Mu1_p4")
flow.Define("Dimuon_m",  "Dimuon_p4.M()")

flow.Define("indices_SelectedMuon_pt_sorted", "Argsort(-SelectedMuon_pt)", requires=["twoOppositeSignMuons"])
flow.ObjectAt("LeadMuon", "SelectedMuon", "indices_SelectedMuon_pt_sorted[0]")
flow.ObjectAt("SubMuon",  "SelectedMuon", "indices_SelectedMuon_pt_sorted[1]")

flow.Selection("etaLeadMuonPos", "LeadMuon_eta > 0.0")
flow.Selection("etaLeadMuonNeg", "LeadMuon_eta <= 0.0")

flow.BuildFlow()

targetList = ["HISTO_nSelectedMuon",
              "HISTO_Dimuon_m__twoOppositeSignMuons",
              "HISTO_LeadMuon_pt__etaLeadMuonPos",
              "HISTO_LeadMuon_pt__etaLeadMuonNeg",
              "HISTO_LeadMuon_eta__etaLeadMuonPos",
              "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)

pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```


```
pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```

Example code - processor Loop

Loop processor code generated from the flow described in the previous slides

Code structure:

1. Functions declarations (for all derived variables)
2. Processing tool function declaration
   a. TTree access (via TTreeReader)
   b. Variables declarations (input and derived)
   c. Loop over events:
      i. Variables initialization
      ii. Input variable reading
      iii. Derived variables evaluation
   d. Get the results!

```
#include <vector>
//...
#include <ROOT/RVec.hxx>

struct Result {
  Result() {}
  std::map<std::string, TH1D> histos;
};

float func__Weight_normalisation(
                ) {
       return 1.0f;
}
float func__Weight_base_1(
                ) {
       return 1.0f;
}
ROOT::VecOps::RVec<float> func__Muon_iso(
               ROOT::VecOps::RVec<float> Muon_pfRelIso04_all) {
       return Muon_pfRelIso04_all;
}
//...
ROOT::VecOps::RVec<int> func__mask_Muon_SelectedMuon(
               ROOT::VecOps::RVec<float> Muon_iso,
               ROOT::VecOps::RVec<float> Muon_pt_GeV,
               ROOT::VecOps::RVec<float> Muon_eta) {
       return Muon_iso < 0.25 && Muon_pt_GeV > 20. && abs(Muon_eta) < 2.4;
}
//...
float func__LeadMuon_pt_GeV(
               ROOT::VecOps::RVec<float> SelectedMuon_pt_GeV,
               unsigned long mask_SelectedMuon_LeadMuon) {
       return At(SelectedMuon_pt_GeV,mask_SelectedMuon_LeadMuon);
}
//...
float func__Dimuon_m(
               ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > Dimuon_p4) {
       return Dimuon_p4.M();
}
```

`//… = similar lines omitted`

Example code - processor Loop

Loop processor code generated from the flow described in the previous slides

Code structure:

1. Functions declarations (for all derived variables)
2. **Processing tool function declaration**
   a. TTree access (via TTreeReader)
   b. Variables declarations (input and derived)
   c. Loop over events:
      i. Variables initialization
      ii. Input variable reading
      iii. Derived variables evaluation
   d. Get the results!

```
Result event_processorLoop() {

  Result r;

  TFile* input_file = TFile::Open("../test_data/OpenData_ATLAS-DAOD_PHYSLITE.37621409._000041.pool.root.1");
  TTree* input_tree = (TTree*)input_file->Get("CollectionTree");

  TTreeReader reader(input_tree);
  reader.Restart();

  const float Weight_normalisation           = func__Weight_normalisation();
  //...

  TTreeReaderArray<float>         ra_Muon_pt(reader, "AnalysisMuonsAuxDyn.pt");
  //...
  TTreeReaderArray<float>         ra_Muon_charge(reader, "AnalysisMuonsAuxDyn.charge");

  ROOT::VecOps::RVec<float>       Muon_pt;
  ROOT::VecOps::RVec<float>       Muon_charge;
  float                           regionWeight_463e3c32cc3d8896bce4e16356708e9c;
  ROOT::VecOps::RVec<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > > Muon_p4;
  int                             nSelectedMuon;
  bool                            twoOppositeSignMuons;
  //...
  float                           Dimuon_m;

  r.histos[std::string("HISTO_nSelectedMuon")]                  =
    TH1D("HISTO_nSelectedMuon",                   "nSelectedMuon {}",                   10 ,     0 ,    10);
  //...
  r.histos[std::string("HISTO_LeadMuon_eta__etaLeadMuonNeg")]   =
    TH1D("HISTO_LeadMuon_eta__etaLeadMuonNeg",    "LeadMuon_eta {etaLeadMuonNeg}",     100 ,  -5.0 ,   5.0);
  r.histos[std::string("HISTO_Dimuon_m__twoOppositeSignMuons")] =
    TH1D("HISTO_Dimuon_m__twoOppositeSignMuons",  "Dimuon_m {twoOppositeSignMuons}",   100 ,  50.0 , 150.0);

  TH1D* HISTO_nSelectedMuon                    = &(r.histos[std::string("HISTO_nSelectedMuon")]);
  //...
  TH1D* HISTO_LeadMuon_eta__etaLeadMuonNeg     = &(r.histos[std::string("HISTO_LeadMuon_eta__etaLeadMuonNeg")]);
  TH1D* HISTO_Dimuon_m__twoOppositeSignMuons   = &(r.histos[std::string("HISTO_Dimuon_m__twoOppositeSignMuons")]);
```

`//… = similar lines omitted`

Example code - processor Loop

Loop processor code generated from the flow described in the previous slides

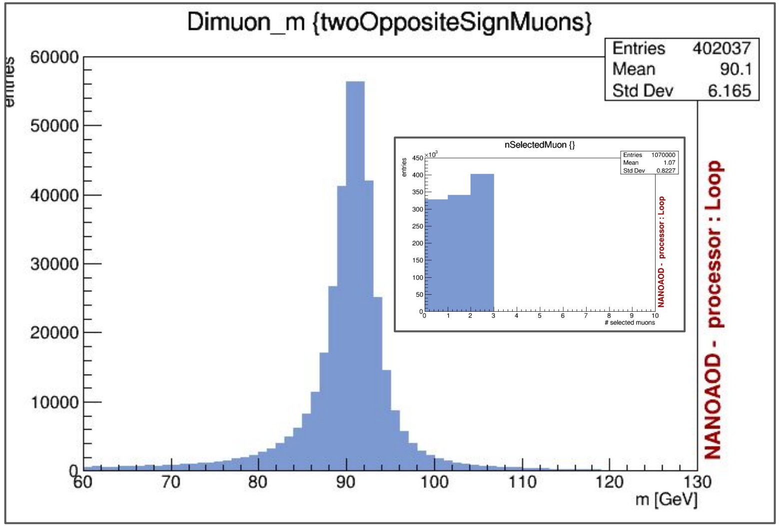
Code structure:

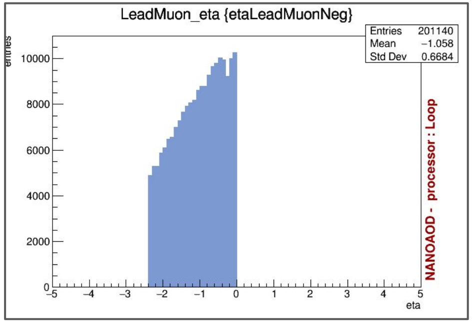
1. Functions declarations (for all derived variables)
2. Processing tool function declaration
   a. TTree access (via TTreeReader)
   b. Variables declarations (input and derived)
   c. Loop over events:
      i. Variables initialization
      ii. Input variable reading
      iii. Derived variables evaluation
   d. Get the results!

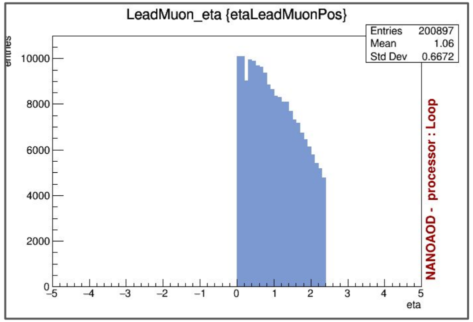
```
while (reader.Next()) {

  twoSelectedMuons                = false;
  //...
  etaLeadMuonNeg                  = false;

  Muon_pt                         = ROOT::VecOps::RVec<float>(ra_Muon_pt.begin() , ra_Muon_pt.end() );
  //...
  Muon_charge                     = ROOT::VecOps::RVec<float>(ra_Muon_charge.begin() , ra_Muon_charge.end() );

  regionWeight_463e3c32cc3d8896bce4e16356708e9c      =
    func__regionWeight_463e3c32cc3d8896bce4e16356708e9c(Weight_normalisation, Weight_base_1);
  Muon_p4                                            = func__Muon_p4(Muon_pt_GeV, Muon_eta, Muon_phi, Muon_m);
  nSelectedMuon                                      = func__nSelectedMuon(mask_Muon_SelectedMuon);
  //...
  twoSelectedMuons                                   = func__twoSelectedMuons(nSelectedMuon);

  if (twoSelectedMuons) {
    indices_MuMu                                     = func__indices_MuMu(nSelectedMuon);
    //...
    twoOppositeSignMuons                             = func__twoOppositeSignMuons(OppositeSignMuMu);
  }

  if (twoSelectedMuons and twoOppositeSignMuons) {
    indices_Mu                                       = func__indices_Mu(OppositeSignMuMu);
    //...
    etaLeadMuonNeg                                   = func__etaLeadMuonNeg(LeadMuon_eta);
    Dimuon_m                                         = func__Dimuon_m(Dimuon_p4);
    HISTO_Dimuon_m__twoOppositeSignMuons             -> Fill(Dimuon_m, regionWeight_2b38bb86d0cc4a6505869ee098e4b9b0);
  }

  if (twoSelectedMuons and twoOppositeSignMuons and etaLeadMuonPos) {
    HISTO_LeadMuon_pt_GeV__etaLeadMuonPos            -> Fill(LeadMuon_pt_GeV, regionWeight_2b38bb86d0cc4a6505869ee098e4b9b0);
    HISTO_LeadMuon_eta__etaLeadMuonPos               -> Fill(LeadMuon_eta, regionWeight_2b38bb86d0cc4a6505869ee098e4b9b0);
  }

  if (twoSelectedMuons and twoOppositeSignMuons and etaLeadMuonNeg) {
    HISTO_LeadMuon_pt_GeV__etaLeadMuonNeg            -> Fill(LeadMuon_pt_GeV, regionWeight_2b38bb86d0cc4a6505869ee098e4b9b0);
    HISTO_LeadMuon_eta__etaLeadMuonNeg               -> Fill(LeadMuon_eta, regionWeight_2b38bb86d0cc4a6505869ee098e4b9b0);
  }
}

return r;
}
```

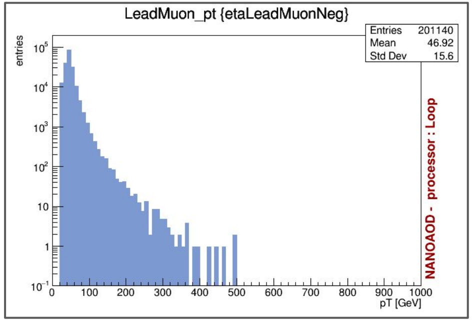
`//… = similar lines omitted`

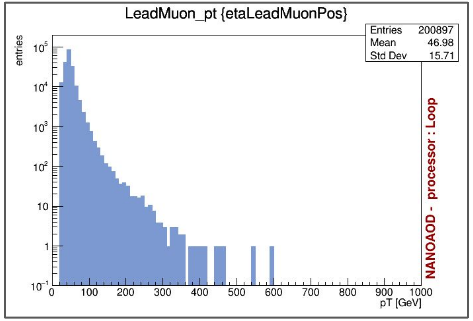
NANOAOD - processor Loop





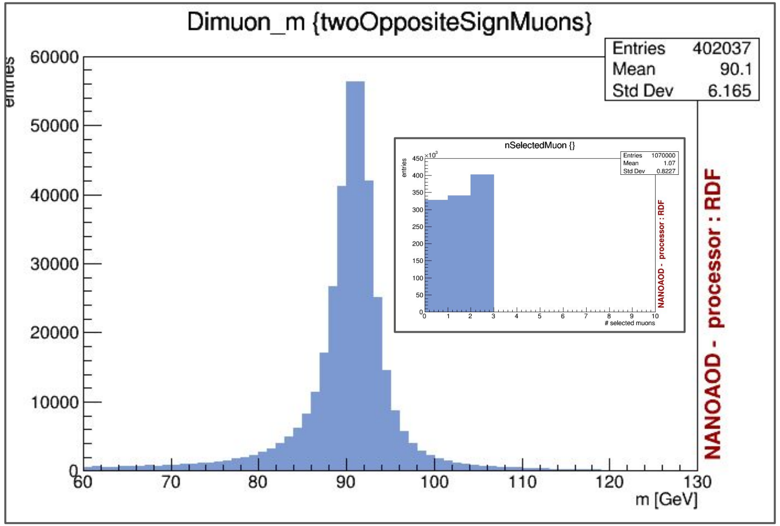


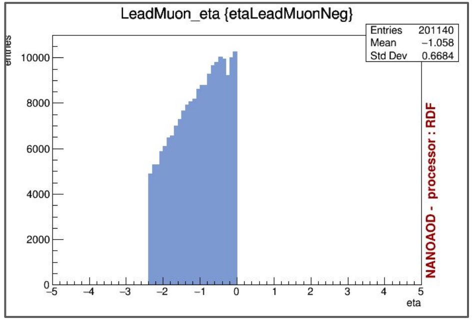


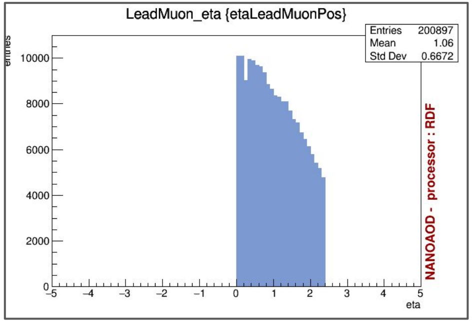


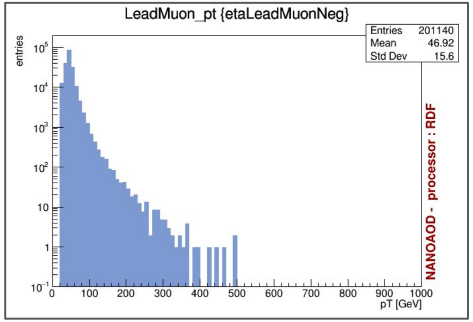
CMS OpenData: https://opendata.cern.ch/record/75482 ([root file](#))
Simulated Z ➛μμ for pp collisions @ 13 TeV - 1.070.000 events

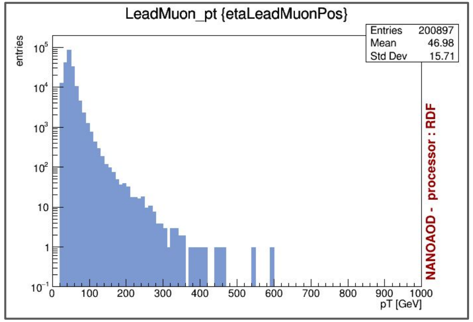
NANOAOD - processor RDF





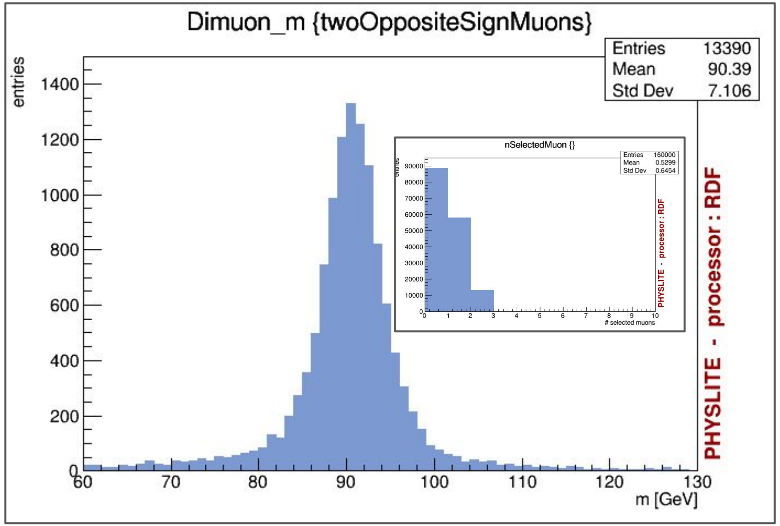


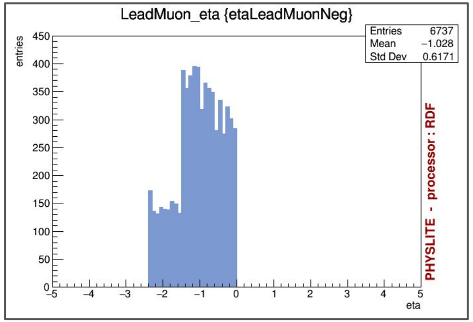


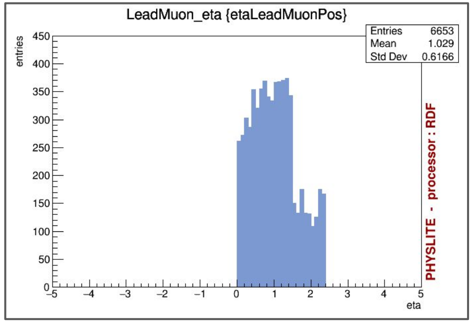


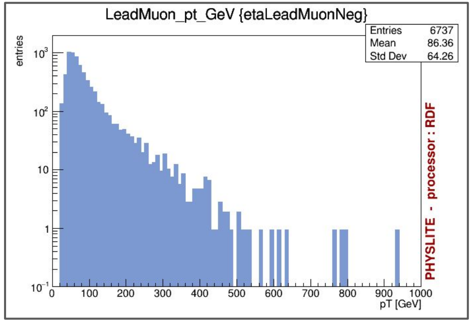
CMS OpenData: https://opendata.cern.ch/record/75482 (root file)
Simulated Z ➛μμ for pp collisions @ 13 TeV - 1.070.000 events

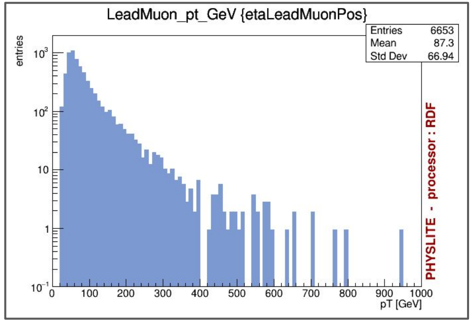
PHYSLITE - processor RDF









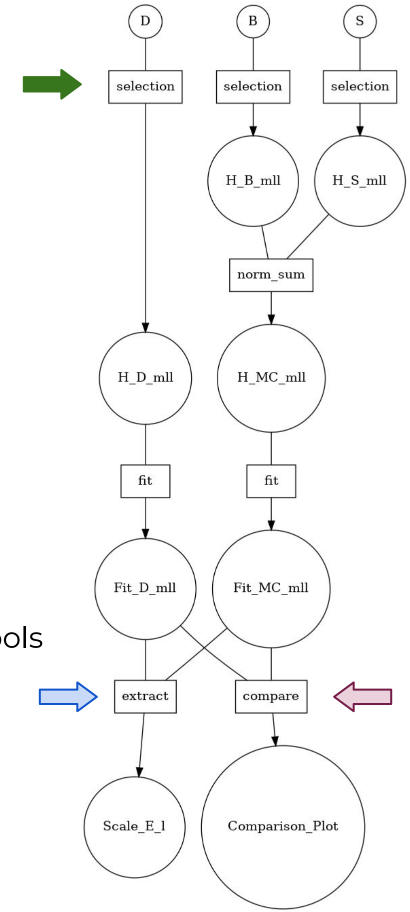


ATLAS OpenData: https://opendata.cern.ch/record/80010 (root file)
Simulated Z ➛μμ for pp collisions @ 13 TeV - 160.000 events

Same analysis flow definition respect to CMS NANOAOD sample - but no requirements on muon quality.

Extension to full analysis chain

- Example of a full analysis chain (dummy-style calibration task):
  - Sample preparation
  - **Event Loop (NAIL - fully re-implemented now)**
  - Snapshot/data reduction
  - Combination / comparison of distributions
  - Statistical analysis / Extraction of results
  - Selective / incremental execution
- Target:
  - Definition and implementation of the full analysis chain via declarative tools
  - Improved result preservation
  - Automatic optimization

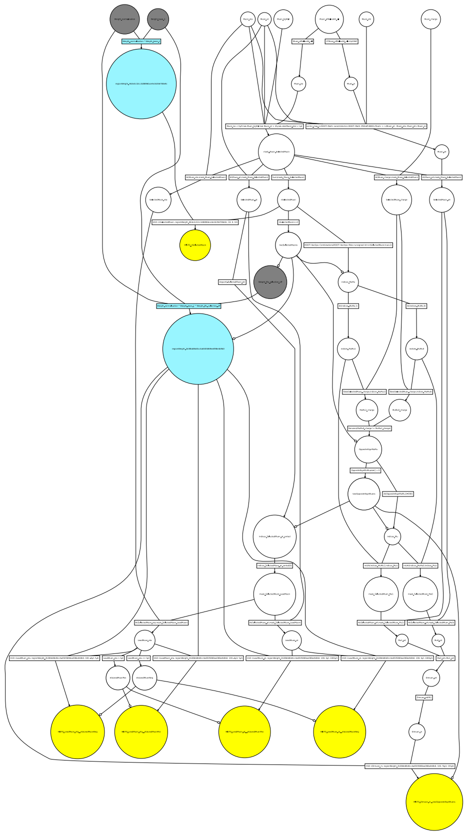


Example: (over-) simplified energy calibration scheme

Summary

- *The application of (more)* ***declarative paradigm*** *in analysis description and implementation can boost*
  - ***DEVELOPMENT : Speed, Portability, Preservation***
  - ***PERFORMANCE : Flexibility, Parallelization, Optimization***

- ***Demonstrator*** *in development*
  - implements core features and handles for extensions
  - two main extensions:
    - **data-format interface** (different experiments)
    - **full analysis chain** (expanded tasks)
  - will benefit from cutting-edge HW technologies and community feedback for test and optimization phases



Backup

Analysis paradigm: declarative vs imperative

- These is a correlation between the paradigm used for the description of the analysis algorithm and the <u>*programming paradigm*</u> used for its implementation in a *software program*.

Main paradigm approaches

(from Wikipedia)

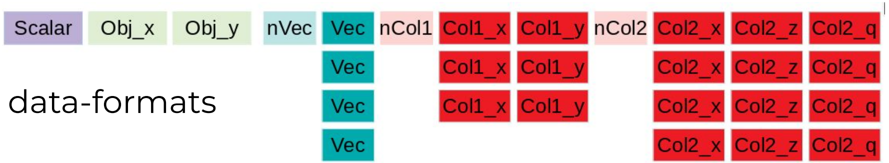
There are two main approaches to programming:[1]

- Imperative programming – focuses on how to execute, defines control flow as statements that change a program state.
- Declarative programming – focuses on what to execute, defines program logic, but not detailed control flow.

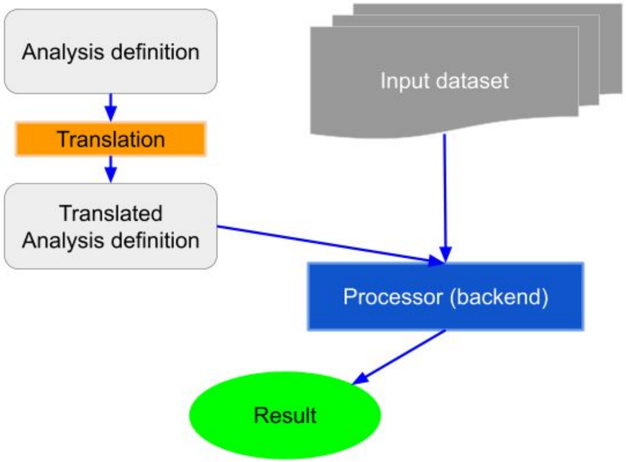
- So far mainly imperative paradigms have been used for analysis description and implementation
  - More straightforward application for “simple” tasks and linear/serial tools

- What has changed in the last decade?
  - **HW** : parallelism/multithreading
  - **SW** : more expressive programming languages (Python, C++ 17/20/23)
  - **Tasks** : increased complexity, increased data size (analyses, combinations)

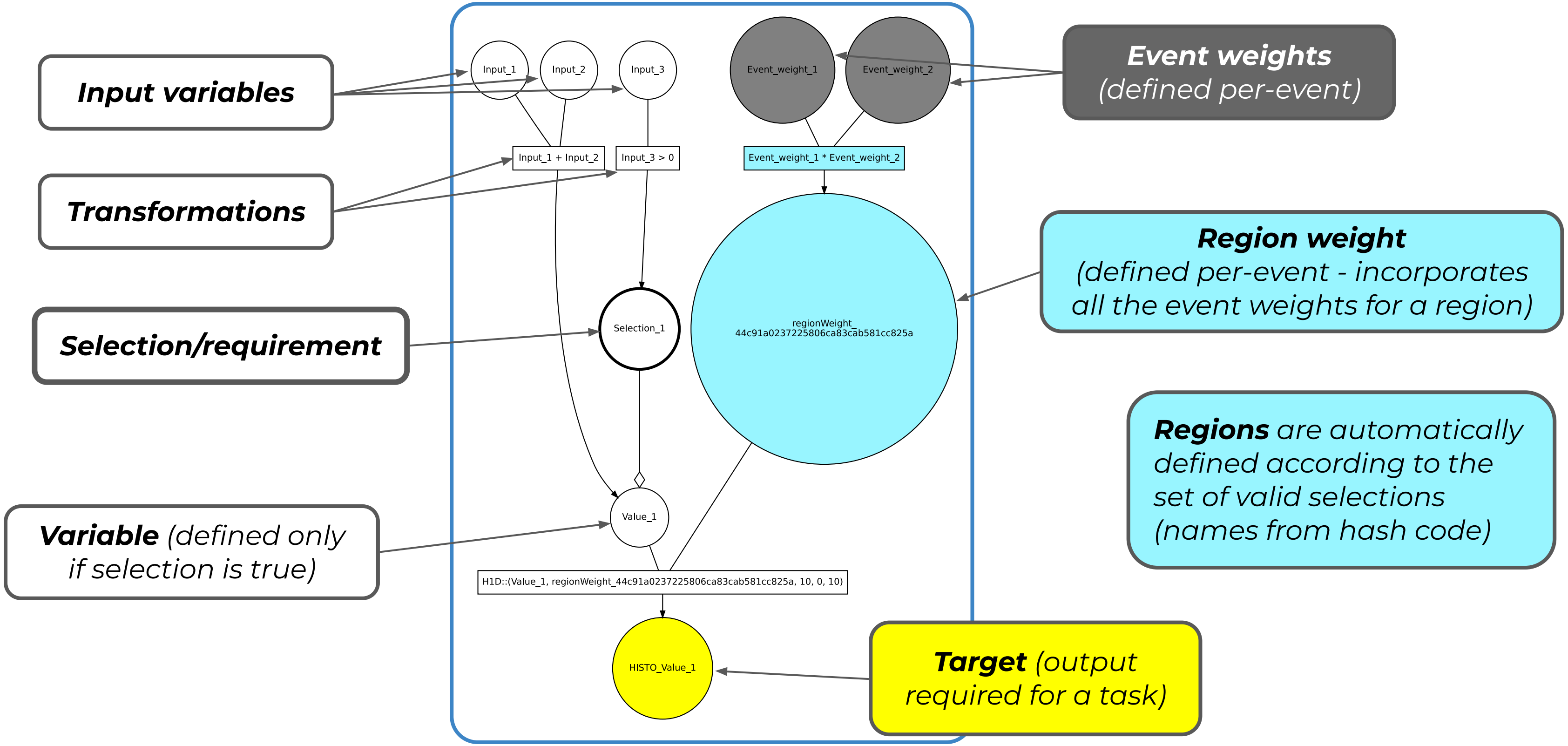
Data-format interface



- In principle 3 *equivalent* - but in general distinct - data-formats are involved in an analysis definition:
  - **a.** inside the framework for *variables manipulation*
  - **b.** in the *description of the algorithm* by the user
  - **c.** in the *encoding of the input* data to be processed
- **a** and **b** can - in principle - be unified for most applications
- **c** is *experiment dependent* : a translation is needed **a ↔ c**
- Strategy implemented for the demonstrator:
  - a. Translation via a configurable dictionary tool (Python)
  - b. Encode all the data-format specific information (and configurations needed) in a dictionary (JSON file)



DAG example



***Regions** are automatically defined according to the set of valid selections (names from hash code)*

Example code - flow

- Definition of tools and first derived vars

```python
from eventFlow import *
from processorRDF import *

flow = SampleProcessing("flowTest", 'dictionaries/nanoAOD_nanoAOD_id_OpenData.json')

flow.DefineEventWeight("Weight_normalisation",    "1.0f")
flow.DefineEventWeight("Weight_base_1",           "1.0f")

flow.Define("Muon_m",    "0*Muon_pfRelIso04_all+0.1056f")
flow.Define("Muon_p4",
            "vector_map_t<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > >(Muon_pt, Muon_eta, Muon_phi, Muon_m)")
flow.Define("Muon_iso", "Muon_pfRelIso04_all")
flow.SubCollection("SelectedMuon", "Muon", sel="Muon_iso < 0.25 && Muon_tightId && Muon_pt > 20. && abs(Muon_eta) < 2.4")
```

```python
flow.DefineHisto1D("Dimuon_m", ["twoOppositeSignMuons"], 100, 50.0, 150.0)

flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonPos'], 100, 0.0, 1000.0)
flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonNeg'], 100, 0.0, 1000.0)

flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonPos'], 100, -5.0, 5.0)
flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonNeg'], 100, -5.0, 5.0)

flow.BuildFlow()

targetList = ["HISTO_nSelectedMuon",
              "HISTO_Dimuon_m__twoOppositeSignMuons",
              "HISTO_LeadMuon_pt__etaLeadMuonPos",
              "HISTO_LeadMuon_pt__etaLeadMuonNeg",
              "HISTO_LeadMuon_eta__etaLeadMuonPos",
              "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)

pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```


```python
from eventFlow import *
from processorRDF import *

flow = SampleProcessing("flowTest", 'dictionaries/nanoAOD_nanoAOD_id_OpenData.json')

flow.DefineEventWeight("Weight_normalisation",    "1.0f")
flow.DefineEventWeight("Weight_base_1",           "1.0f")

flow.Define("Muon_m",    "0*Muon_pfRelIso04_all+0.1056f")
flow.Define("Muon_p4",
            "vector_map_t<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > >(Muon_pt, Muon_eta, Muon_phi, Muon_m)")
flow.Define("Muon_iso", "Muon_pfRelIso04_all")
```

Example code - flow

- Definition of tools and first derived vars
- Selection of 2 opposite-charge muons

```
from eventFlow import *
from processorRDF import *

flow = SampleProcessing("flowTest", 'dictionaries/nanoAOD_nanoAOD_id_OpenData.json')

flow.DefineEventWeight("Weight_normalisation",    "1.0f")
flow.DefineEventWeight("Weight_base_1",           "1.0f")

flow.Define("Muon_m",    "0*Muon_pfRelIso04_all+0.1056f")
flow.Define("Muon_p4",
            "vector_map_t<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > >(Muon_pt, Muon_eta, Muon_phi, Muon_m)")
flow.Define("Muon_iso", "Muon_pfRelIso04_all")
flow.SubCollection("SelectedMuon", "Muon", sel="Muon_iso < 0.25 && Muon_tightId && Muon_pt > 20. && abs(Muon_eta) < 2.4")

flow.Selection("twoSelectedMuons", "nSelectedMuon==2")

flow.DefineEventWeight("Weight_Mu_selection_eff", "0.95f", requires=["twoSelectedMuons"])

flow.Distinct("MuMu", "SelectedMuon", requires=["twoSelectedMuons"])

flow.Define("OppositeSignMuMu", "Nonzero(MuMu0_charge != MuMu1_charge)", requires=["twoSelectedMuons"])

flow.Selection("twoOppositeSignMuons", "OppositeSignMuMu.size() > 0")

flow.TakePair("Mu", "SelectedMuon", "MuMu", "At(OppositeSignMuMu,0,-200)", requires=["twoOppositeSignMuons"])

flow.Define("Dimuon_p4", "Mu0_p4+Mu1_p4")
...
              "HISTO_LeadMuon_eta__etaLeadMuonPos",
              "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)

pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```


```
flow.SubCollection("SelectedMuon", "Muon", sel="Muon_iso < 0.25 && Muon_tightId && Muon_pt > 20. && abs(Muon_eta) < 2.4")

flow.Selection("twoSelectedMuons", "nSelectedMuon==2")

flow.DefineEventWeight("Weight_Mu_selection_eff", "0.95f", requires=["twoSelectedMuons"])

flow.Distinct("MuMu", "SelectedMuon", requires=["twoSelectedMuons"])

flow.Define("OppositeSignMuMu", "Nonzero(MuMu0_charge != MuMu1_charge)", requires=["twoSelectedMuons"])

flow.Selection("twoOppositeSignMuons", "OppositeSignMuMu.size() > 0")

flow.TakePair("Mu", "SelectedMuon", "MuMu", "At(OppositeSignMuMu,0,-200)", requires=["twoOppositeSignMuons"])
```

Example code - flow

- Definition of tools and first derived vars
- Selection of 2 opposite-charge muons
- **Evaluation of the dimuon invariant mass**

```
from eventFlow import *
from processorRDF import *

flow = SampleProcessing("flowTest", 'dictionaries/nanoAOD_nanoAOD_id_OpenData.json')

flow.DefineEventWeight("Weight_normalisation",   "1.0f")
flow.DefineEventWeight("Weight_base_1",          "1.0f")

flow.Define("Muon_m",    "0*Muon_pfRelIso04_all+0.1056f")
flow.Define("Muon_p4",
            "vector_map_t<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > >(Muon_pt, Muon_eta, Muon_phi, Muon_m)")
flow.Define("Muon_iso", "Muon_pfRelIso04_all")

flow.SubCollection("SelectedMuon", "Muon", sel="Muon_iso < 0.25 && Muon_tightId && Muon_pt > 20. && abs(Muon_eta) < 2.4")

flow.Selection("twoSelectedMuons", "nSelectedMuon==2")

flow.DefineEventWeight("Weight_Mu_selection_eff", "0.95f", requires=["twoSelectedMuons"])

flow.Distinct("MuMu", "SelectedMuon", requires=["twoSelectedMuons"])

flow.Define("OppositeSignMuMu", "Nonzero(MuMu0_charge != MuMu1_charge)", requires=["twoSelectedMuons"])

flow.Selection("twoOppositeSignMuons", "OppositeSignMuMu.size() > 0")

flow.TakePair("Mu", "SelectedMuon", "MuMu", "At(OppositeSignMuMu,0,-200)", requires=["twoOppositeSignMuons"])

flow.Define("Dimuon_p4",  "Mu0_p4+Mu1_p4")
flow.Define("Dimuon_m",   "Dimuon_p4.M()")

flow.Define("indices_SelectedMuon_pt_sorted", "Argsort(-SelectedMuon_pt)", requires=["twoOppositeSignMuons"])
flow.ObjectAt("LeadMuon", "SelectedMuon", "indices_SelectedMuon_pt_sorted[0]")
flow.ObjectAt("SubMuon",  "SelectedMuon", "indices_SelectedMuon_pt_sorted[1]")
```


```
flow.Define("Dimuon_p4",  "Mu0_p4+Mu1_p4")
flow.Define("Dimuon_m",   "Dimuon_p4.M()")
```


```
flow.DefineHisto1D("LeadMuon_pt",   ['etaLeadMuonPos'], 100, 0.0, 1000.0)
flow.DefineHisto1D("LeadMuon_pt",   ['etaLeadMuonNeg'], 100, 0.0, 1000.0)

flow.DefineHisto1D("LeadMuon_eta",  ['etaLeadMuonPos'], 100, -5.0, 5.0)
flow.DefineHisto1D("LeadMuon_eta",  ['etaLeadMuonNeg'], 100, -5.0, 5.0)

flow.BuildFlow()

targetList = ["HISTO_nSelectedMuon",
              "HISTO_Dimuon_m__twoOppositeSignMuons",
              "HISTO_LeadMuon_pt__etaLeadMuonPos",
              "HISTO_LeadMuon_pt__etaLeadMuonNeg",
              "HISTO_LeadMuon_eta__etaLeadMuonPos",
              "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)

pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```

Example code - flow

- Definition of tools and first derived vars
- Selection of 2 opposite-charge muons
- Evaluation of the dimuon invariant mass
- Definition of 2 regions:
  - Leading-muon eta > 0
  - Leading-muon eta <= 0

```
from eventFlow import *
from processorRDF import *

flow = SampleProcessing("flowTest", 'dictionaries/nanoAOD_nanoAOD_id_OpenData.json')

flow.DefineEventWeight("Weight_normalisation",   "1.0f")
flow.DefineEventWeight("Weight_base_1",          "1.0f")

flow.Define("Muon_m",   "0*Muon_pfRelIso04_all+0.1056f")
flow.Define("Muon_p4",
            "vector_map_t<ROOT::Math::LorentzVector<ROOT::Math::PtEtaPhiM4D<float> > >(Muon_pt, Muon_eta, Muon_phi, Muon_m)")
flow.Define("Muon_iso", "Muon_pfRelIso04_all")

flow.SubCollection("SelectedMuon", "Muon", sel="Muon_iso < 0.25 && Muon_tightId && Muon_pt > 20. && abs(Muon_eta) < 2.4")

flow.Selection("twoSelectedMuons", "nSelectedMuon==2")

flow.DefineEventWeight("Weight_Mu_selection_eff", "0.95f", requires=["twoSelectedMuons"])

flow.Distinct("MuMu", "SelectedMuon", requires=["twoSelectedMuons"])

flow.Define("OppositeSignMuMu", "Nonzero(MuMu0_charge != MuMu1_charge)", requires=["twoSelectedMuons"])

flow.Selection("twoOppositeSignMuons", "OppositeSignMuMu.size() > 0")

flow.TakePair("Mu", "SelectedMuon", "MuMu", "At(OppositeSignMuMu,0,-200)", requires=["twoOppositeSignMuons"])

flow.Define("Dimuon_p4", "Mu0_p4+Mu1_p4")
flow.Define("Dimuon_m",  "Dimuon_p4.M()")

flow.Define("indices_SelectedMuon_pt_sorted", "Argsort(-SelectedMuon_pt)", requires=["twoOppositeSignMuons"])
flow.ObjectAt("LeadMuon", "SelectedMuon", "indices_SelectedMuon_pt_sorted[0]")
flow.ObjectAt("SubMuon",  "SelectedMuon", "indices_SelectedMuon_pt_sorted[1]")

flow.Selection("etaLeadMuonPos", "LeadMuon_eta > 0.0")
flow.Selection("etaLeadMuonNeg", "LeadMuon_eta <= 0.0")
```



```
flow.Define("indices_SelectedMuon_pt_sorted", "Argsort(-SelectedMuon_pt)", requires=["twoOppositeSignMuons"])
flow.ObjectAt("LeadMuon", "SelectedMuon", "indices_SelectedMuon_pt_sorted[0]")
flow.ObjectAt("SubMuon",  "SelectedMuon", "indices_SelectedMuon_pt_sorted[1]")

flow.Selection("etaLeadMuonPos", "LeadMuon_eta > 0.0")
flow.Selection("etaLeadMuonNeg", "LeadMuon_eta <= 0.0")
```



```
               "HISTO_LeadMuon_pt__etaLeadMuonPos",
               "HISTO_LeadMuon_pt__etaLeadMuonNeg",
               "HISTO_LeadMuon_eta__etaLeadMuonPos",
               "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)

pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```

Example code - flow

- Definition of tools and first derived va
- Selection of 2 opposite-charge muons
- Evaluation of the dimuon invariant m
- Definition of 2 regions:
  - Leading-muon eta > 0
  - Leading-muon eta <= 0
- Distributions:
  - # selected muons
  - dimuon invariant mass
  - Leading-muon p_T (per region)
  - Leading-muon eta (per region)

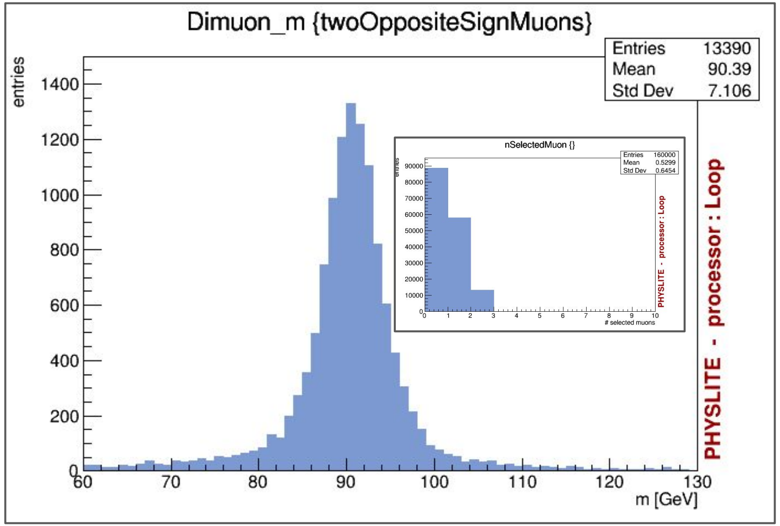
```
flow.DefineHisto1D("nSelectedMuon", [], 10, 0, 10)
flow.DefineHisto1D("Dimuon_m", ["twoOppositeSignMuons"], 100, 50.0, 150.0)

flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonPos'], 100, 0.0, 1000.0)
flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonNeg'], 100, 0.0, 1000.0)

flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonPos'], 100, -5.0, 5.0)
flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonNeg'], 100, -5.0, 5.0)

flow.BuildFlow()

targetList = ["HISTO_nSelectedMuon",
              "HISTO_Dimuon_m__twoOppositeSignMuons",
              "HISTO_LeadMuon_pt__etaLeadMuonPos",
              "HISTO_LeadMuon_pt__etaLeadMuonNeg",
              "HISTO_LeadMuon_eta__etaLeadMuonPos",
              "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)
```

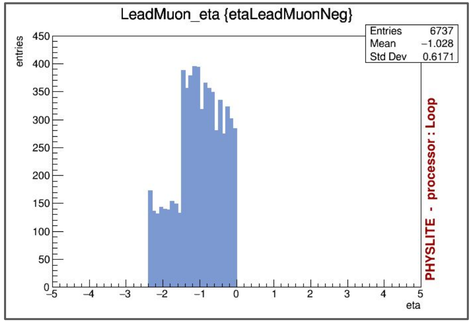
```
flow.Selection("etaLeadMuonPos", "LeadMuon_eta > 0.0")
flow.Selection("etaLeadMuonNeg", "LeadMuon_eta <= 0.0")
```

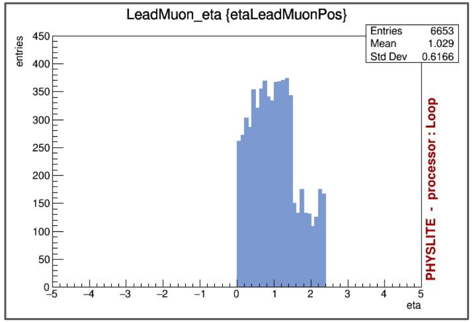

```
flow.DefineHisto1D("nSelectedMuon", [], 10, 0, 10)
flow.DefineHisto1D("Dimuon_m", ["twoOppositeSignMuons"], 100, 50.0, 150.0)

flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonPos'], 100, 0.0, 1000.0)
flow.DefineHisto1D("LeadMuon_pt",  ['etaLeadMuonNeg'], 100, 0.0, 1000.0)

flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonPos'], 100, -5.0, 5.0)
flow.DefineHisto1D("LeadMuon_eta", ['etaLeadMuonNeg'], 100, -5.0, 5.0)

flow.BuildFlow()

targetList = ["HISTO_nSelectedMuon",
              "HISTO_Dimuon_m__twoOppositeSignMuons",
              "HISTO_LeadMuon_pt__etaLeadMuonPos",
              "HISTO_LeadMuon_pt__etaLeadMuonNeg",
              "HISTO_LeadMuon_eta__etaLeadMuonPos",
              "HISTO_LeadMuon_eta__etaLeadMuonNeg"]

flow.SetTargets(targetList)
```

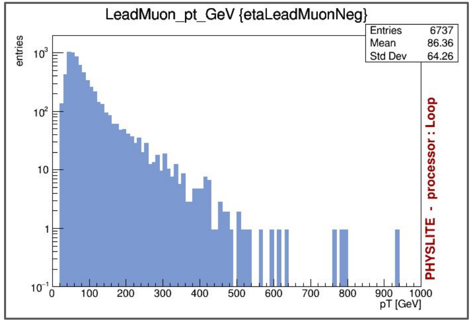

```
pRDF = Processor_RDF("pRDF", flow, "../test_data/OpenData_CMS-DA1BF301-762C-5048-A9EB-AB534069FB4B.root", "Events")

pRDF.RunProcessor()
```

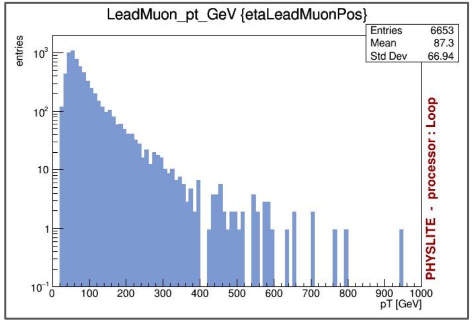
PHYSLITE - processor Loop











ATLAS OpenData: https://opendata.cern.ch/record/80010 (root file)
Simulated Z ➝μμ for pp collisions @ 13 TeV - 160.000 events

Same analysis flow definition respect to CMS NANOAOD sample - but no requirements on muon quality.

Performance

- ***Qualitative*** comparison (OpenData & typical 2023 laptop):
  a. Analysis flow definition
  b. Backend (C++) generation & compilation
  c. Execution in a Python session

NANOAOD *sample:*
CMS OpenData: https://opendata.cern.ch/record/75482 (root file)
Simulated Z → μμ for pp collisions @ 13 TeV - 1.070.000 events

PHYSLITE *sample:*
ATLAS OpenData: https://opendata.cern.ch/record/80010 (root file)
Simulated Z → μμ for pp collisions @ 13 TeV - 160.000 events

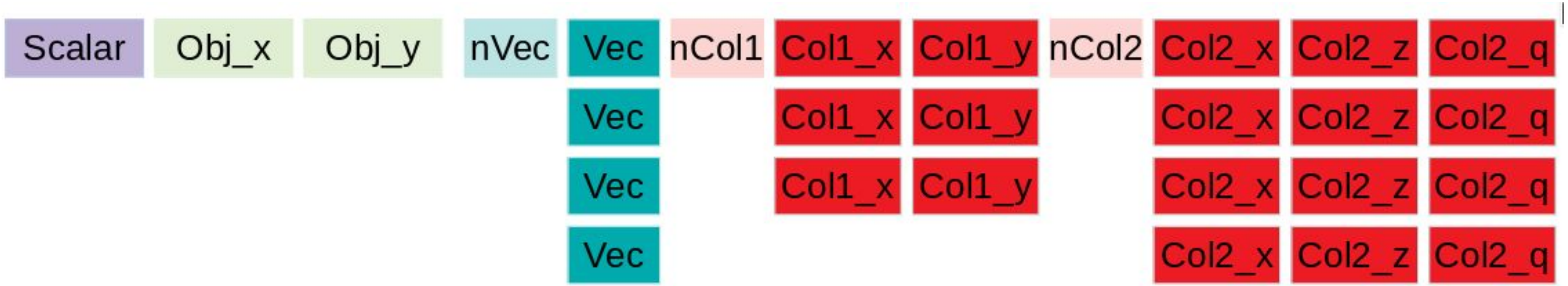
processor	***NANOAOD*** (1M events)		***PHYSLITE*** (160k events)	
	t(a+b+c) [s]	t(c) [s]	t(a+b+c) [s]	t(c) [s]
Loop	14.7	6.9	10.5	2.6
RDF	14.0	6.1	10.1	2.1
RDF (8 threads)	10.5	3.2	9.8	2.4

Plain Loop more readable than RDF / RDF ~30% faster processing

NAIL (Natural Analysis Implementation Language)

- *“NAIL is an analysis language that should allow to define in an abstract way a data analysis of a typical HEP experiment such as CMS or ATLAS. NAIL assumes an input data model for the event to process (…) and allow to specify the event by event processing actions in a declarative form. Analysis variations for optimizations and systematics do not need to be explicitly coded but are automatically derived from the event processing computational graph. Currently ROOT's RDataFrame is used as backend for a concrete implementation of the event processing as it allows parallelization and lazy evaluation.”* (from the README file of the NAIL package)

- Developed in the CMS collaboration, main developer Andrea Rizzi
- Based on CMS’ **nanoAOD** (*columnar*) data format, written in Python, heavy lift in C++ (RDataFrame)



AoS vs SoA

- From Wikipedia : *"In computing, an **array of structures (AoS)**, **structure of arrays (SoA)** … are contrasting ways to arrange a sequence of records in memory, with regard to interleaving, and are of interest in SIMD and SIMT programming."*

AoS

```c
struct point3D {
    float x;
    float y;
    float z;
};
struct point3D points[N];
float get_point_x(int i) { return points[i].x; }
```

SoA

```c
struct pointlist3D {
    float x[N];
    float y[N];
    float z[N];
};
struct pointlist3D points;
float get_point_x(int i) { return points.x[i]; }
```

- CMS: SoA (e.g. nanoAOD)
- ATLAS: AoS interface with SoA memory storage (e.g. xAOD, PHYSLITE)

Where the increased speed comes from?

- **RVec**
  - *“A "std::vector"-like collection of values implementing handy operation to analyse them.”*
  - [Documentation]
  - Optimized for **speed**
  - Its storage is contiguous in memory
  - **Automatic internal loop**

```
std::vector<float> goodMuons_pt;
const auto size = mu_charge.size();
for (size_t i=0; i < size; ++i) {
   if (mu_pt[i] > 10 && abs(mu_eta[i]) <= 2. &&  mu_charge[i] == -1) {
      goodMuons_pt.emplace_back(mu_pt[i]);
   }
}
```

```
auto goodMuons_pt = mu_pt[ (mu_pt > 10.f && abs(mu_eta) <= 2.f && mu_charge == -1) ]
```

Where the increased speed comes from?

- **RDataFrame**
  - *"ROOT's RDataFrame offers a modern, high-level **interface** for analysis of data stored in TTree , CSV and other data formats, in C++ or Python.*

    In addition, multi-threading and other low-level optimizations allow users to exploit all the resources available on their machines completely transparently."
  - Documentation
  - Optimized for **speed**
  - **Lazy** evaluation and **automatic internal loop**

```
TTreeReader reader("myTree", file);
TTreeReaderValue<A_t> a(reader, "A");
TTreeReaderValue<B_t> b(reader, "B");
TTreeReaderValue<C_t> c(reader, "C");
while(reader.Next()) {
   if(IsGoodEvent(*a, *b, *c))
      DoStuff(*a, *b, *c);
}
```

→

```
ROOT::RDataFrame d("myTree", file, {"A", "B", "C"});
d.Filter(IsGoodEvent).Foreach(DoStuff);
```