EDM4hep Meeting

EDM4hep.jl Update

Pere Mato / CERN
26 March 2024

https://github.com/perematto/EDM4hep.jl

Motivation for EDM4hep.jl - Reminder

- Generate Julia 'friendly' structures for the EDM4hep data model
  - As friendly as the Python bindings but with C++ performance
- Be able to read event data files in ROOT format (TTree & RNTuple) written by C++ programs from Julia (using the UnROOT.jl package)
  - Development of data analysis functionality
- Later, be able also to write RNTuple files from Julia
  - Either with UnROOT.jl (Julia native) or wrapping ROOT (GSOC project)

Implementing EDM4hep in Julia is a pre-requisite for introducing the Julia language in Simulation and Reconstruction workflows

PODIO Generation

- Written small Julia script to generate Julia structs from YAML file
  - Added a **`ObjectID`** to each object to control its registration state
  - Relations implemented with **`ObjectID`** and **`Relation`** structs with just indices (isbits() = POD)
- Two files: **`genComponents.jl`**, **`genDatatypes.jl`** generated that can be complemented with utility methods

```
"""
struct MCParticle

    Description: The Monte Carlo particle - based on the lcio::MCParticle.
    Author: F.Gaede, DESY
"""
struct MCParticle <: POD
    index::ObjectID{MCParticle}      # ObjectID of itself
   #---Data Members
    PDG::Int32                       # PDG code of the particle
    generatorStatus::Int32           # status of the particle as defined by the ...
    simulatorStatus::Int32           # status of the particle from the simulation ...
    charge::Float32                  # particle charge
    time::Float32                    # creation time of the particle in [ns] wrt. ...
    mass::Float64                    # mass of the particle in [GeV]
    vertex::Vector3d                 # production vertex of the particle in [mm].
    endpoint::Vector3d               # endpoint of the particle in [mm]
    momentum::Vector3f               # particle 3-momentum at the production vertex..
    momentumAtEndpoint::Vector3f     # particle 3-momentum at the endpoint in [GeV]
    spin::Vector3f                   # spin (helicity) vector of the particle.
    colorFlow::Vector2i              # color flow as defined by the generator

    #---OneToManyRelations
    parents::Relation{MCParticle,1}  # The parents of this particle.
    daughters::Relation{MCParticle,2}  # The daughters this particle.
end
```

```
"""
struct SimTrackerHit

    Description: Simulated tracker hit
    Author: F.Gaede, DESY
"""
struct SimTrackerHit <: POD
    index::ObjectID{SimTrackerHit}   # ObjectID of itself
    #---Data Members
    cellID::UInt64                   # ID of the sensor that created this hit
    EDep::Float32                    # energy deposited in the hit [GeV].
    time::Float32                    # proper time of the hit in the lab frame in ...
    pathLength::Float32              # path length of the particle in the sensiti ...
    quality::Int32                   # quality bit flag.
    position::Vector3d               # the hit position in [mm].
    momentum::Vector3f               # the 3-momentum of the particle at the hits ...
    #---OneToOneRelations
    mcparticle_idx::ObjectID{MCParticle}  # MCParticle that caused the hit.
end
```

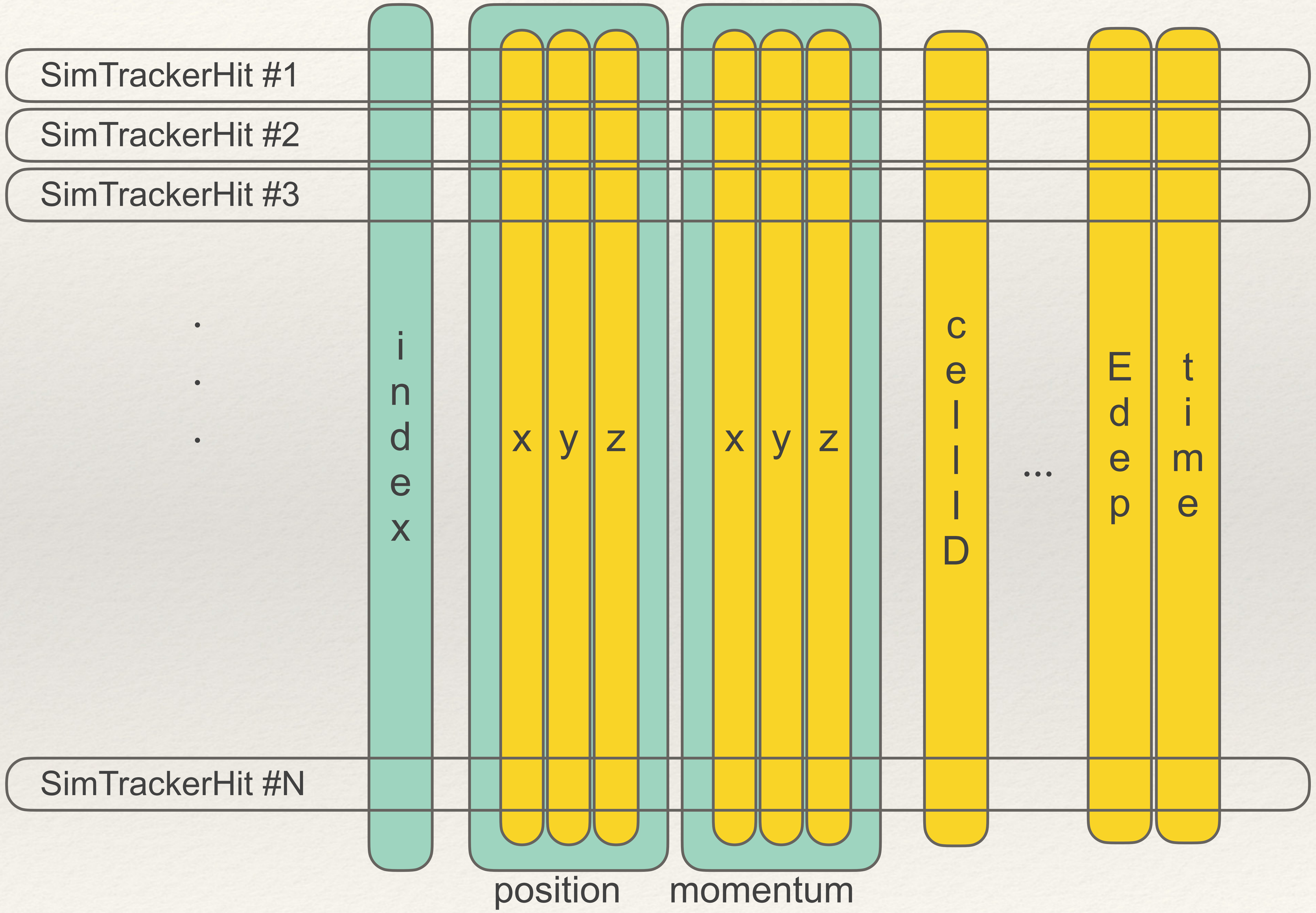
ROOT I/O

- Using **UnROOT.jl** package - a really great package!
- Supports (transparently) TTree and RNTuple formats and several versions of PODIO storage (versions 16.x and 17.x)
    - data files consist exclusively of 'collections-of-datatypes' (e.g. ReconstructedParticles, Vertices, etc.)
- The goal is to obtain a **`StructArray{DataType}`** of each collection for each event
    - The exercise consists in mapping the schema in the ROOT file to the actual Julia datatype (using the Julia introspection or generated code)

Creating SoAs from EDM4hep types

- UnROOT.jl provides the leaves arrays (in a lazy manner) and they are "mapped" to form SoA of a DataType
- Opens the possibility of schema evolution
  - filling empty attributes, type change, re-shaping, etc.

```
using StructArrays

# Create a struct array
hits = StructArray{SimTrackerHit}(Tuple(<TLeaf>...))


# Access elements
println(hits[1])  # Output: SimTrackerHit(....)
```



SoA provides an Ergonomic and Efficient interface

- Storage in memory consists of a set of column arrays
  - very fast access by column
- Materialize, when requested, object instances (usually on the stack) to be able to call user object methods (multiple dispatch)
  - to achieve a user friendly access

```
julia> mcps = <get all MCParticle collection>

julia> typeof(mcps[1])
MCParticle

julia> typeof(mcps.charge)
SubArray{Float32, 1, Vector{Float32},
Tuple{UnitRange{Int64}}, true}

julia> length(mcps.charge)
211

julia> mcps[1:2].momentum
2-element StructArray(::Vector{Float32}, ::Vector{Float32},
::Vector{Float32}) with eltype Vector3f:
 (0.5000167,0.0,50.0)
 (0.5000167,0.0,-50.0)

julia> sum(mcps[1:2].momentum)
(1.0000334,0.0,0.0)
```

Reading from a ROOT (TTree) File

```
using EDM4hep
using EDM4hep.RootIO

cd(@__DIR__)

f = "ttbar_edm4hep_digi.root"

reader = RootIO.Reader(f)
events = RootIO.get(reader, "events")

evt = events[1];

hits = RootIO.get(reader, evt, "InnerTrackerBarrelCollection")
mcps = RootIO.get(reader, evt, "MCParticle")

for hit in hits
    println("Hit $(hit.index) is related to MCParticle $(hit.mcparticle.index) with name $(hit.mcparticle.name)")
end

#---Loop over events-------------------------------------------------------------
for (n,e) in enumerate(events)
    ps =  RootIO.get(reader, e, "MCParticle")
    println("Event #$(n) has $(length(ps)) MCParticles with a charge sum of $(sum(ps.charge))")
end
```

```
Hit #1 is related to MCParticle #65 with name pi+
Hit #2 is related to MCParticle #65 with name pi+
Hit #3 is related to MCParticle #65 with name pi+
Hit #4 is related to MCParticle #65 with name pi+
Hit #5 is related to MCParticle #66 with name pi-
Hit #6 is related to MCParticle #66 with name pi-
Hit #7 is related to MCParticle #66 with name pi-
Hit #8 is related to MCParticle #49 with name pi+
Hit #9 is related to MCParticle #49 with name pi+
Hit #10 is related to MCParticle #49 with name pi+
Hit #11 is related to MCParticle #27 with name K-
Hit #12 is related to MCParticle #27 with name K-
Hit #13 is related to MCParticle #27 with name K-
Hit #14 is related to MCParticle #95 with name e-
Hit #15 is related to MCParticle #95 with name e-
...
```

~ 1500 times faster than Python

What is currently supported?

- EDM4hep files can be local or remote (e.g. root://eospublic.cern.ch/…)
- Single or multiple files
- Sequential and multi-threaded access
- EDM4hep version 1 will be supported after release

	TTree	RNTuple (rc2)
podio v0.16	X	-
podio v0.17	X	X

Multi-threaded Analysis

```
mutable struct MyData <: AbstractAnalysisData
    df::DataFrame
    pevts::Int64
    sevts::Int64
    MyData() = new(DataFrame(...), 0, 0)
end
```

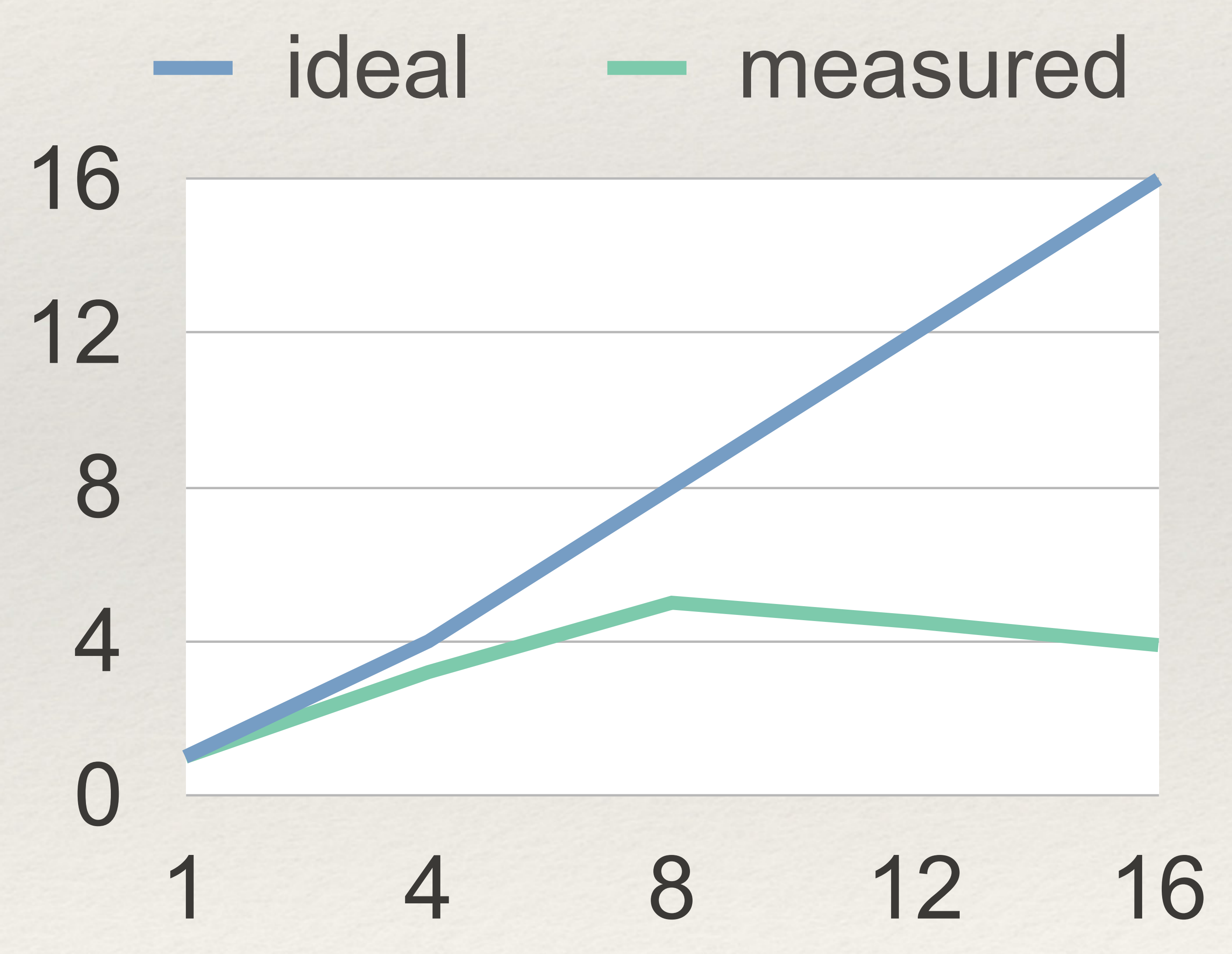
- Developed mini framework to ensure thread safety
  - The user defines a **data structure** and an **analysis function**
- Each thread works on a subset of events using its own copy of the data
- At the end, the results are 'summed' automatically

```
function myanalysis!(data::MyData, reader, events)
  for evt in events
    data.pevts += 1                                 # count process events
    μIDs = RootIO.get(reader, evt, "Muon_objIdx")# get the ids of muons
    length(μIDs) < 2 && continue                    # skip if less than 2

    recps = RootIO.get(reader, evt, "ReconstructedParticles")
    muons = recps[μIDs]                             # use the ids to subset

    sel_muons = filter(x -> pₜ(x) > 10GeV, muons) # select the Pt of muons
    zed_leptonic = resonanceBuilder(91GeV, sel_muons)
    zed_leptonic_recoil = recoilBuilder(240GeV, zed_leptonic)
    if length(zed_leptonic) == 1                    # filter exactly one Z
      Zcand_m        = zed_leptonic[1].mass
      Zcand_recoil_m = zed_leptonic_recoil[1].mass
      Zcand_q        = zed_leptonic[1].charge
      if 80GeV <= Zcand_m <= 100GeV                 # select on mass Z
        push!(data.df, (Zcand_m, Zcand_recoil_m, Zcand_q))
        data.sevts += 1                             # count selected events
      end
    end
  end
  return data
end
```

```
events = RootIO.get(reader, “events")
mydata = MyData()
do_analysis!(mydata, myanalysis!, reader, events;  mt=true)
```

Performance

- Sequential performance is pretty good compared to FCCAnalyses framework (Python+C++) with (higgs/mH-recoil/mumu example)
  - using `lcgapp-centos8-physical.cern.ch`
  - ~21000 events/s compared with ~9500 events/s
- MT scalability is not great
  - peak is reached with 8 cores



Status

- Package EDM4hep.jl is registered and ready for use!
  - Install Julia

```
curl -fsSL https://install.julialang.org | sh
```

  - Install EDM4hep

```
julia -e 'import Pkg; Pkg.add("EDM4hep")'
```

```
julia> using EDM4hep
julia> using EDM4hep.RootIO
julia> file = "root://eospublic.cern.ch//eos/experiment/fcc/ee/generation/DelphesEvents/winter2023/IDEA/
p8_ee_ZZ_ecm240/events_000189367.root"
julia> reader = RootIO.Reader(file)
```

Attribute	Value
File Name(s)	root://eospublic.cern.ch//eos/experiment/fcc/ee/generation/DelphesEvents/winter202....
# of events	100000
IO Format	TTree
PODIO version	0.16.2
ROOT version	6.26.6

```
julia> events = RootIO.get(reader, "events");
julia> evt = events[1];
julia> recps = RootIO.get(reader, evt, "ReconstructedParticles");
julia> recps.energy[1:5]
5-element Vector{Float32}:
```