# podio::(ROOT)DataSource

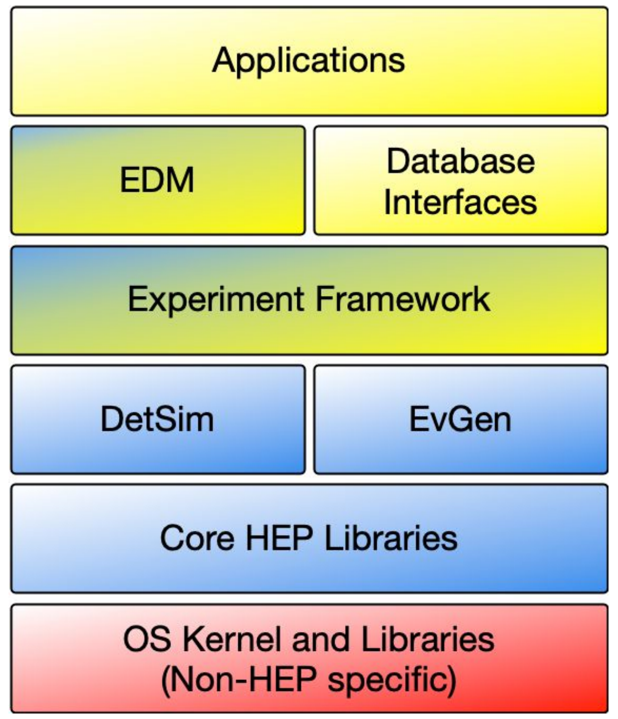
Juraj Smieško (CERN)

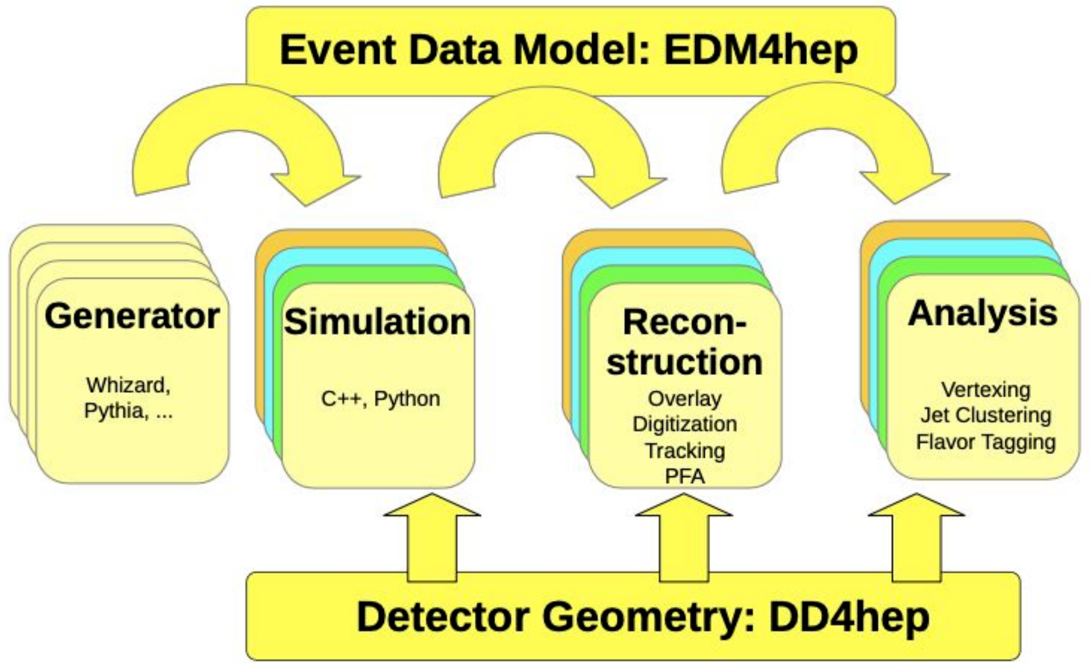
FCC Software Meeting

CERN, 06 May 2024

# Key4hep

- Set of common software packages, tools, and standards for different Detector concepts
- Common for FCC, CLIC/ILC, CEPC, EIC, ...
- Individual participants can mix and match their stack
- Main ingredients:
  - Data processing framework: Gaudi
  - Event data model: EDM4hep
  - Detector description: DD4hep
  - Software distribution: Spack



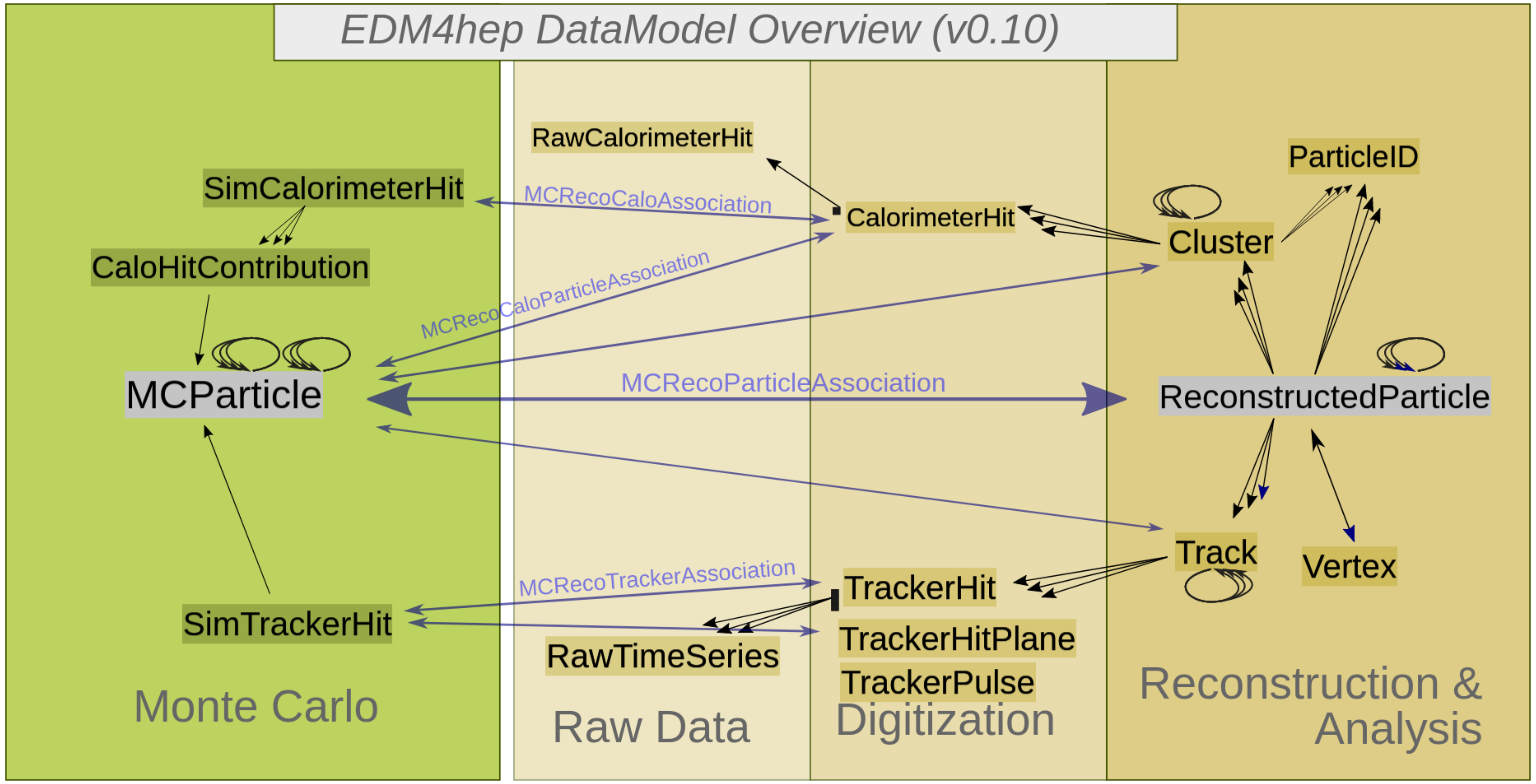




Source: Frank Gaede

# EDM4hep I.

Describes event data with the set of standard objects.

- Specification in a single YAML file
- Generated with the help of Podio



# EDM4hep II.

Example object:

```
#-------------  CalorimeterHit
edm4hep::CalorimeterHit:
  Description: "Calorimeter hit"
  Author: "EDM4hep authors"
  Members:
    - uint64_t cellID                 # detector specific (geometrical) cell id
    - float energy [GeV]              # energy of the hit
    - float energyError [GeV]         # error of the hit energy
    - float time [ns]                 # time of the hit
    - edm4hep::Vector3f position [mm] # position of the hit in world coordinates
    - int32_t type                    # type of hit
```

- Current version: `v0.10.5`
- Objects can be extended / new created
- Bi-weekly discussion: Indico

# EDM4hep 1.0

The EDM4hep will reach version 1.0 soon, breaking changes and fixes are introduced.

Some of the changes/fixes underway:

- Interfaces
- `ReconstructedParticle.type` → `ReconstructedParticle.PDG`
- Reverse the direction of the ParticleID relation(s)
- Vector of weights in EventHeader

```
edm4hep::TrackerHit:
  Description: "Tracker hit interface class"
  Author: "Thomas Madlener, DESY"
  Members:
    - uint64_t cellID                     # ID of the sensor that created this hit
    - int32_t type                        # type of the raw data hit
    - int32_t quality                     # quality bit flag of the hit
    - float time [ns]                     # time of the hit
    - float eDep [GeV]                    # energy deposited on the hit
    - float eDepError [GeV]               # error measured on eDep
    - edm4hep::Vector3d position [mm] # hit position
  Types:
    - edm4hep::TrackerHit3D
    - edm4hep::TrackerHitPlane
```

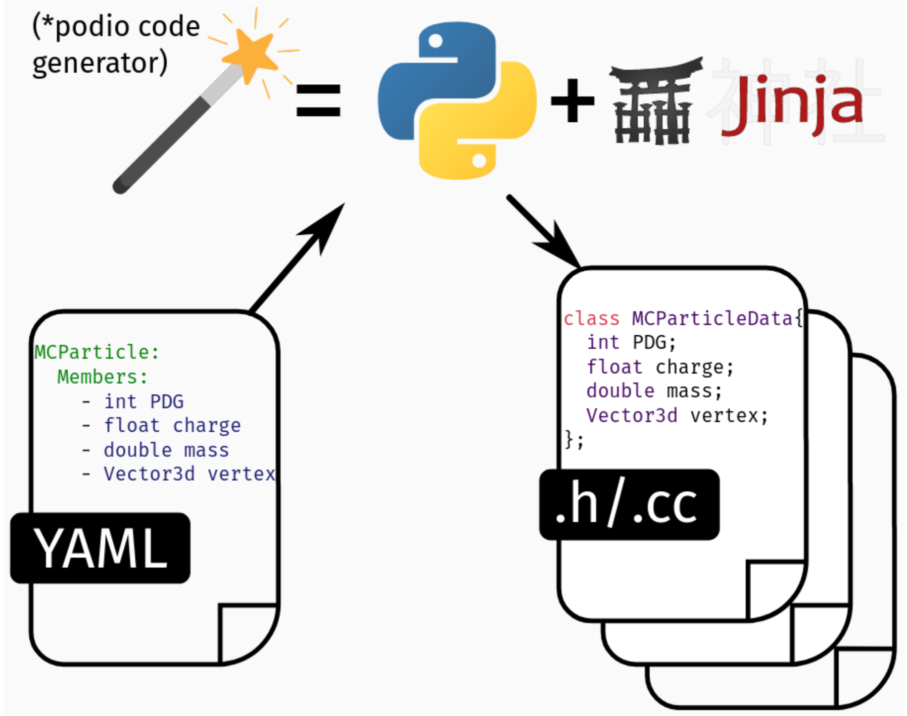
New release of FCCAnalyses 0.9 — preserves state before EDM4hep 1.0 changes

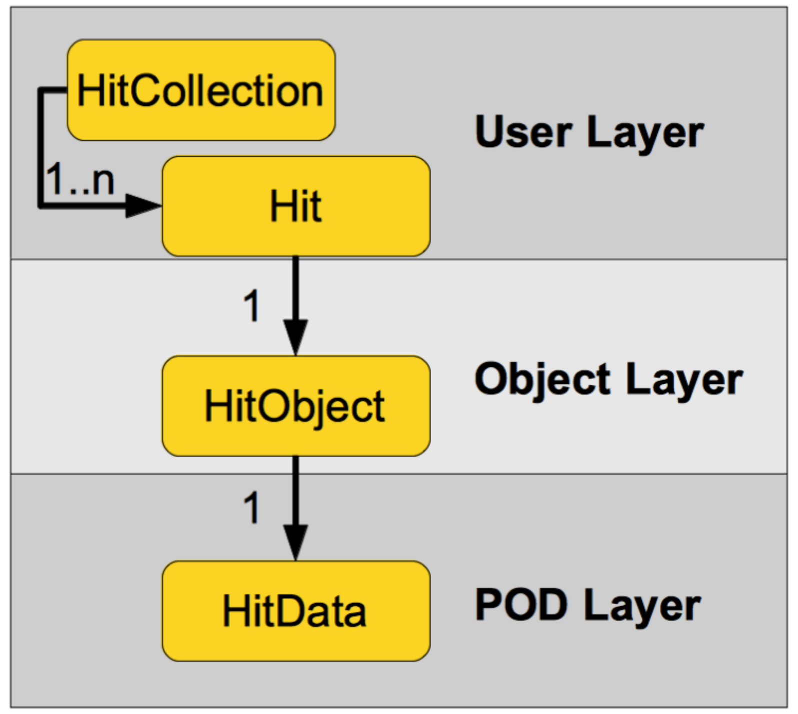
- Will arrive in stable Key4hep stack soon

# Podio

Generates Event Data Model and serves as I/O Layer

- Generates EDM from YAML files
- Employs plain-old-data (POD) data structures
- I/O machinery consists of three layers
  - POD Layer - actual data structures
  - Object Layer - helps resolve the relations
  - User Layer - full fledged EDM objects
- Supports multiple backends:
  - ROOT, SIO, ...
- Current version: `0.99`



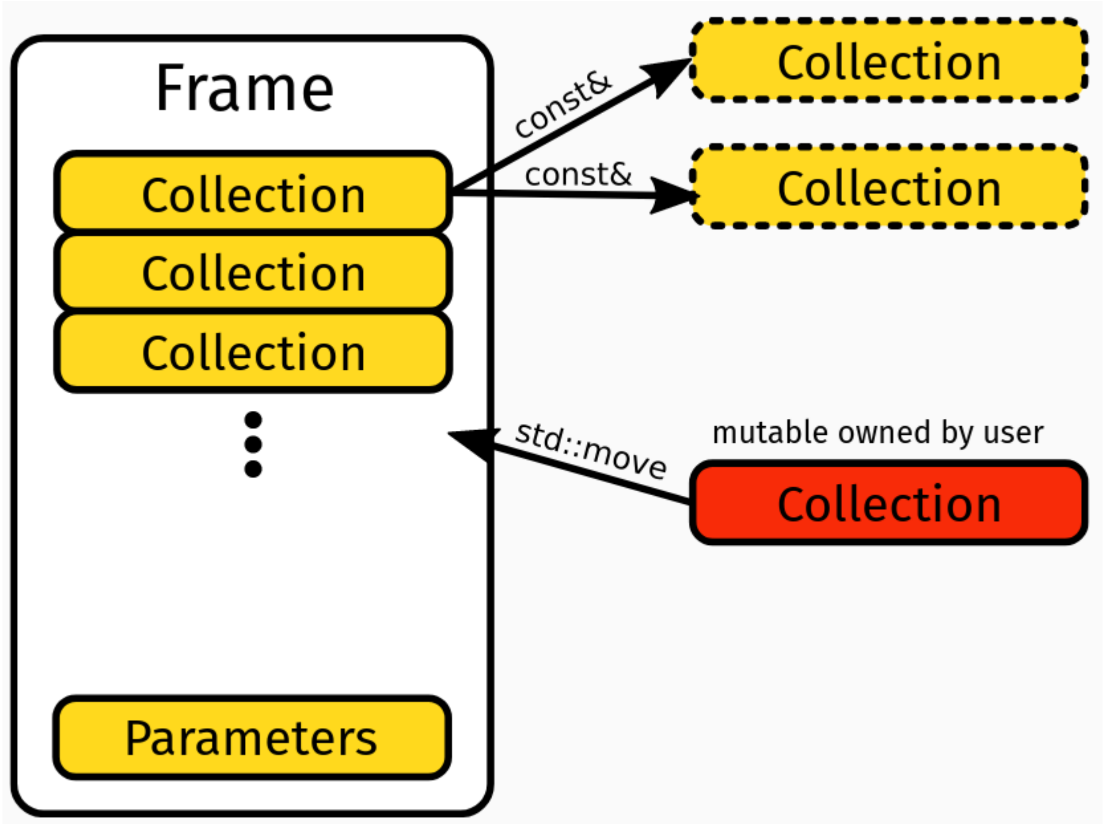



# Podio Reader

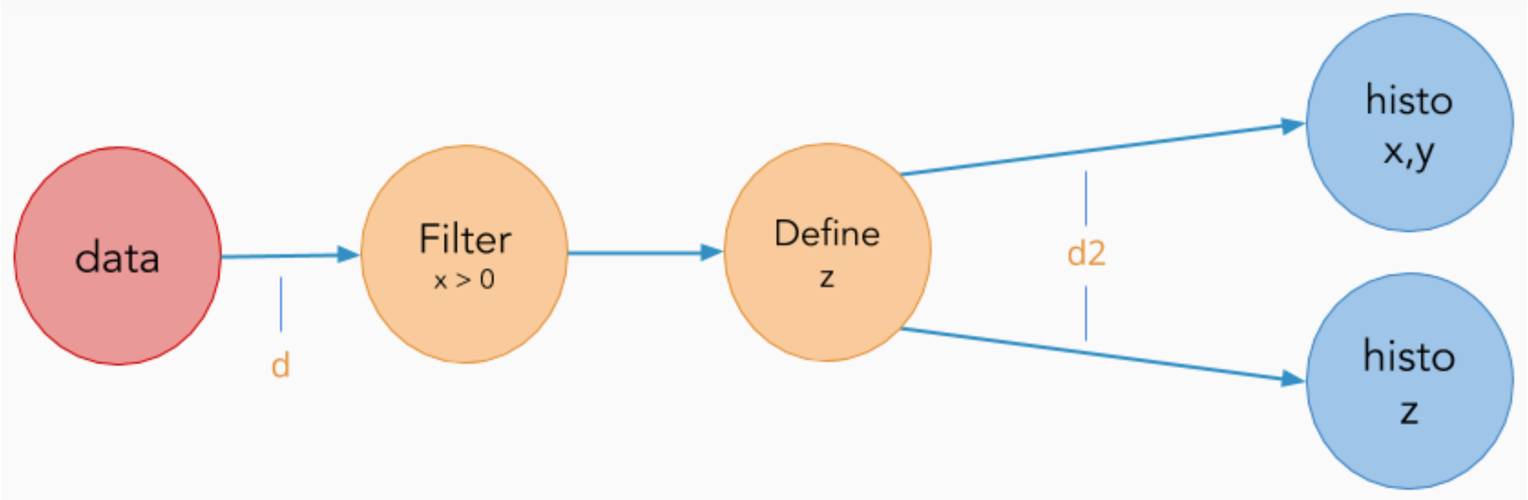
Constructs the EDM4hep objects for the user

Example usage of Podio Reader in Pyhton:

```python
from podio.root_io import Reader
reader = Reader("one or many input files")
for event in reader.get("events"):
  hits = store.get("hits")
  for hit in hits:
    # ...
```



# ROOT RDataFrame




- Describes processing of data as actions on table columns
  - Defines of new columns
  - Filter rules
  - Result definitions (histogram, graph)
- The actions are lazily evaluated
- Multi threading is available out of the box
- Optimized for bulk processing
- Allows integration of existing C++ libraries

# ROOT RDataSource + Podio

*RDataSource defines an API that RDataFrame can use to read arbitrary data formats.*

- RDataSource provides EDM4hep(Podio) collections to the RDataFrame event-by-event
- Collections are constructed by Podio readers
- RDataSource can decide how to organize reading of the events
  - ATM: Not at all optimized
- Multi-threaded implementation
- Might easily support other Podio backends (SIO, ...)
- Schema evolution support

# Reading EDM4hep in FCCAnalyses

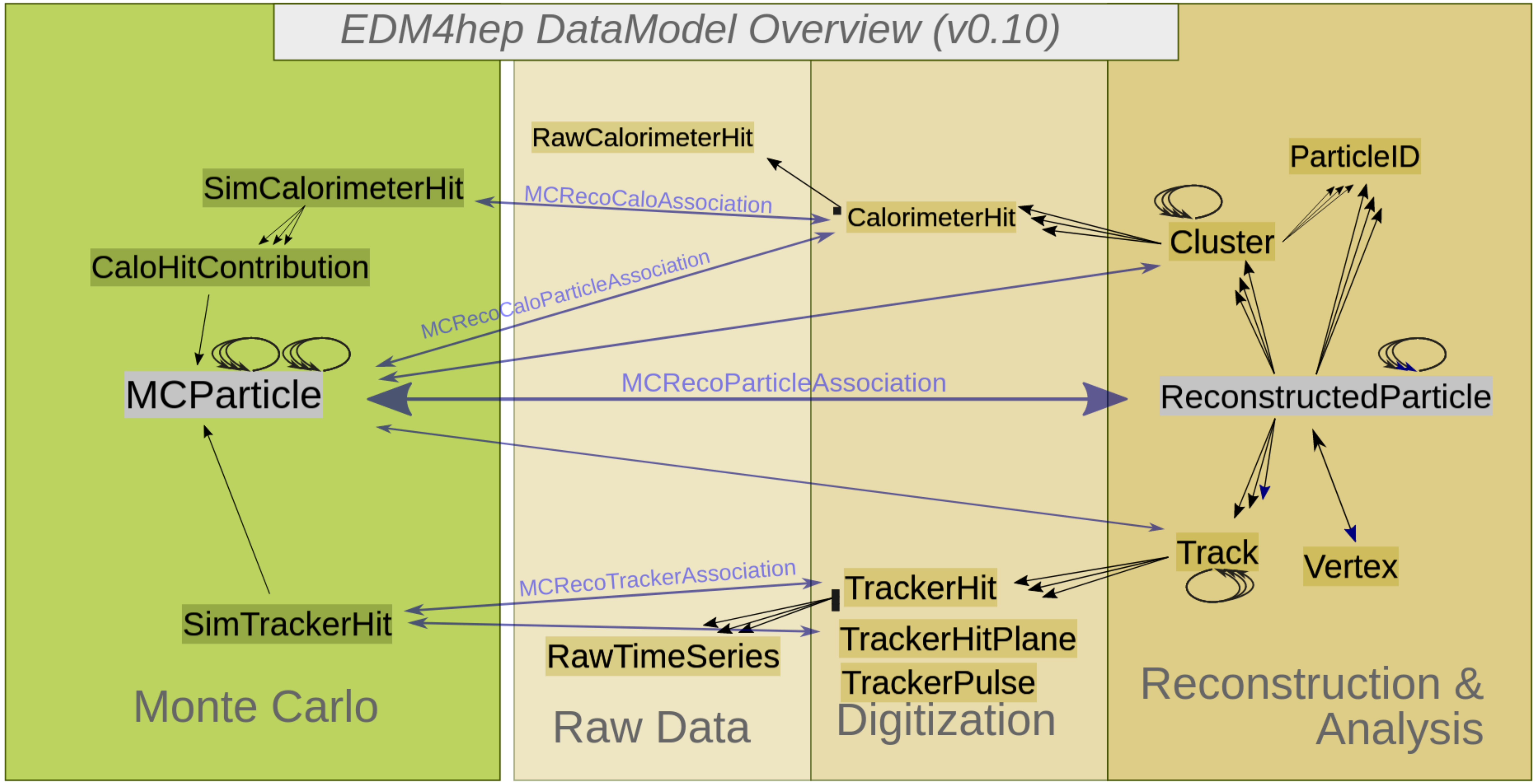
- EDM4hep collection is read in by RDataFrame directly and presented to the user in form:

```
const ROOT::VecOps::RVec<edm4hep::ReconstructedParticleData>& coll
```

  - This is per event
  - No convenient access to relationships
- Example of a simple function:

```
float getMass(const ROOT::VecOps::RVec<edm4hep::ReconstructedParticleData>& in) {
// float getMass(const edm4hep::ReconstructedParticleCollection& in) {
  ROOT::Math::LorentzVector<ROOT::Math::PxPyPzE4D<double>> result;

  for (auto & p: in) {
    ROOT::Math::LorentzVector<ROOT::Math::PxPyPzE4D<double>> tmp;
    tmp.SetPxPyPzE(p.momentum.x, p.momentum.y, p.momentum.z, p.energy);
    // tmp.SetPxPyPzE(p.getMomentum().x, p.getMomentum().y, p.getMomentum().z, p.getEnergy());
    result += tmp;
  }

  return result.M();
}
```

- In the course of the analysis the EDM4hep slowly decays into more trivial objects



EDM4hep DataModel Overview (v0.10)
RawCalorimeterHit
SimCalorimeterHit
MCRecoCaloAssociation
CalorimeterHit
ParticleID
Cluster
CaloHitContribution
MCRecoCaloParticleAssociation
MCParticle
MCRecoParticleAssociation
ReconstructedParticle
Track
Vertex
MCRecoTrackerAssociation
TrackerHit
SimTrackerHit
TrackerHitPlane
RawTimeSeries
TrackerPulse
Monte Carlo
Raw Data
Digitization
Reconstruction &
Analysis

# Relations

- One collection can contain one-to-one or one-to-many relations to other collections, e.g.:
  - `CaloHit` ⇶ `CaloHitContribution`
  - `MCParticle` ⇶ `MCParticle`
- Typically relationships between derived objects (Sim. side separated from Reco. side)
- Example analyzer (FCC Tutorials link):

```
std::vector<int> get_list_of_stable_particles_from_decay( int i, ROOT::VecOps::RVec<edm4hep::MCParticleData> in, ROOT::VecOps::RV
  std::vector<int> res;
  // i = index of a MC particle in the Particle block
  // in = the Particle collection
  // ind = the block with the indices for the daughters, Particle#1.index

  // returns a vector with the indices (in the Particle block) of the stable daughters of the particle i,
  // from the complete decay chain.
  if ( i < 0 || i >= in.size() ) return res;

  int db = in.at(i).daughters_begin ;
  int de = in.at(i).daughters_end;

  if ( db != de ) { // particle is unstable
    for (int id = db; id < de; id++) {
      int idaughter = ind[ id ];
      std::vector<int> rr = get_list_of_stable_particles_from_decay( idaughter, in, ind) ;
      res.insert( res.end(), rr.begin(), rr.end() );
    }
  }
  else {    // particle is stable
     res.push_back( i ) ;
     return res ;
```

# Relations in DataSource

- One collection can contain one-to-one or one-to-many relations to other collections, e.g.:
  - `CaloHit` ⇶ `CaloHitContribution`
  - `MCParticle` ⇶ `MCParticle`
- Typically relationships between derived objects (Sim. side separated from Reco. side)
- Possible rewrite:

```cpp
edm4hep::MCParticleCollection get_stable_particles_from_decay(edm4hep::MCParticle in) {
  edm4hep::MCParticleCollection result;
  result.setSubsetCollection();

  auto daughters = in.getDaughters();
  if (daughters.size() != 0) {  // particle is unstable
    for (const auto& daughter : daughters) {
      auto stable_daughters = get_stable_particles_from_decay(daughter);
      for (const auto& stable_daughter : stable_daughters) {
        result.push_back(stable_daughter);
      }
    }
  } else {  // particle is stable
    result.push_back(in);
  }

  return result;
}
```

# Associations

- One-to-one relationships between two collection types, e.g.:
  - MCParticle ↔ ReconstructedParticle
  - SimTrackerHit ↔ TrackerHit
- Relationships between Simulation and Reconstruction side
- Example analyzer: Association between RecoParticle and MCParticle (link):

```
ROOT::VecOps::RVec<int>
ReconstructedParticle2MC::getRP2MC_index(const ROOT::VecOps::RVec<int>& recind,
                                         const ROOT::VecOps::RVec<int>& mcind,
                                         const ROOT::VecOps::RVec<edm4hep::ReconstructedParticleData>& reco) {
  ROOT::VecOps::RVec<int> result;
  result.resize(reco.size(),-1.);
  for (size_t i=0; i<recind.size();i++) {
    result[recind.at(i)]=mcind.at(i);    // recind.at(i) is the index of a reco'ed particle in the ReconstructedParticl
                                         // mcind.at(i) is the index of its associated MC particle, in the Particle col
  }

  return result;
}
```

# Associations in DataSource

- One-to-one relationships between two collection types, e.g.:
  - `MCParticle` ↔ `ReconstructedParticle`
  - `SimTrackerHit` ↔ `TrackerHit`
- Relationships between Simulation and Reconstruction side
- Possible rewrite:

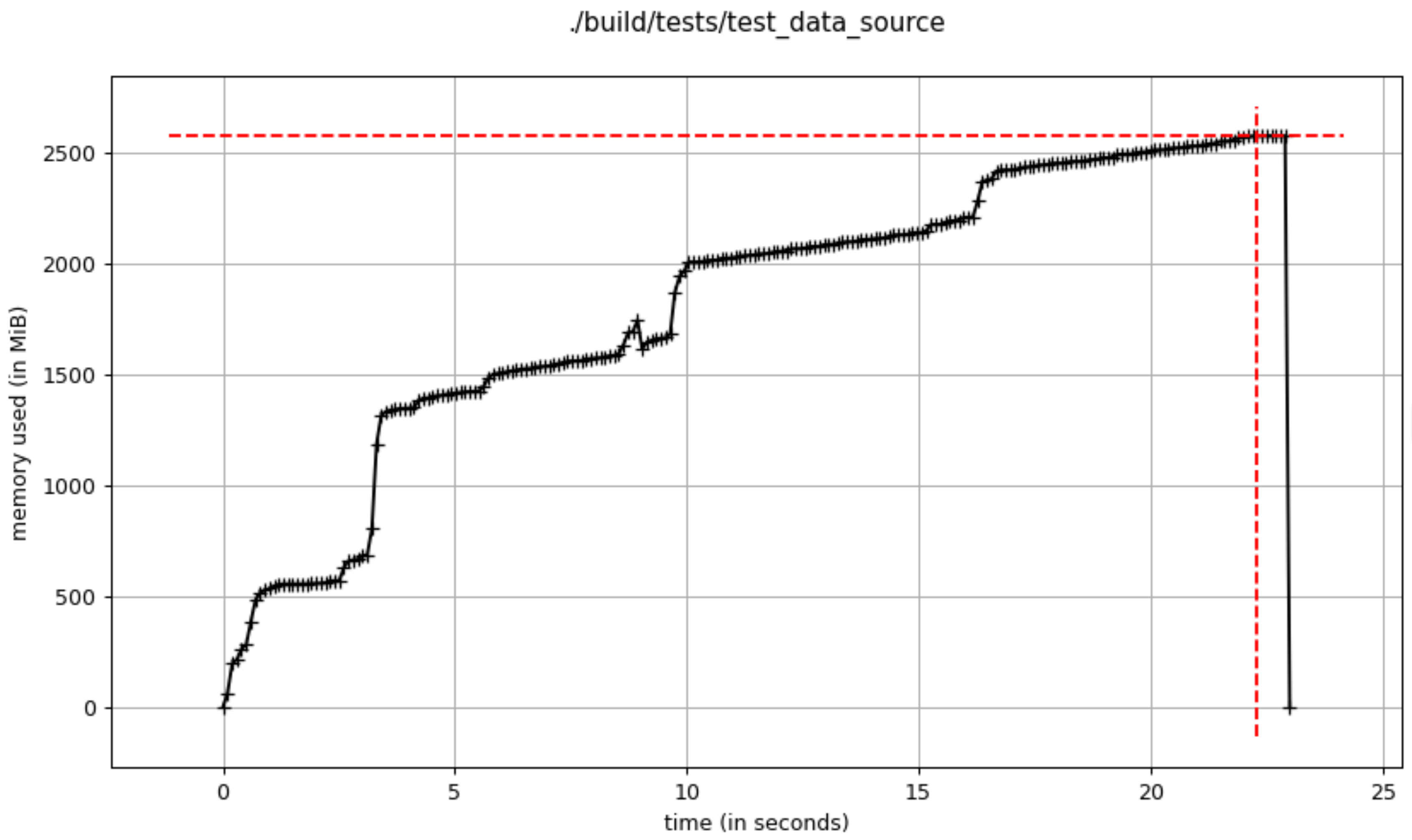
```
edm4hep::MCParticle ReconstructedParticle2MC::get_mcParticle(
    const edm4hep::ReconstructedParticle& recoParticle,
    const edm4hep::MCRecoParticleAssociationCollection& assocCol
  edm4hep::MCParticle no_result;

  for (const auto& assoc: assocColl) {
    if (assoc.getRec() == recoParticle) {
      return assoc.getSim();
    }
  }

  return no_result;
}
```

# Small sample benchmarks

- Three scenarios:
  - Simple C++ analysis
  - C++ analysis with associations
  - Python analysis: `analysis_stage1.py` example
- 300k events
- Local storage
- 4 threads
- DataSource takes approx. 2× memory and 2× execution time
- More tests needed:
  - More complex (real) analysis
  - Running on over the network sample

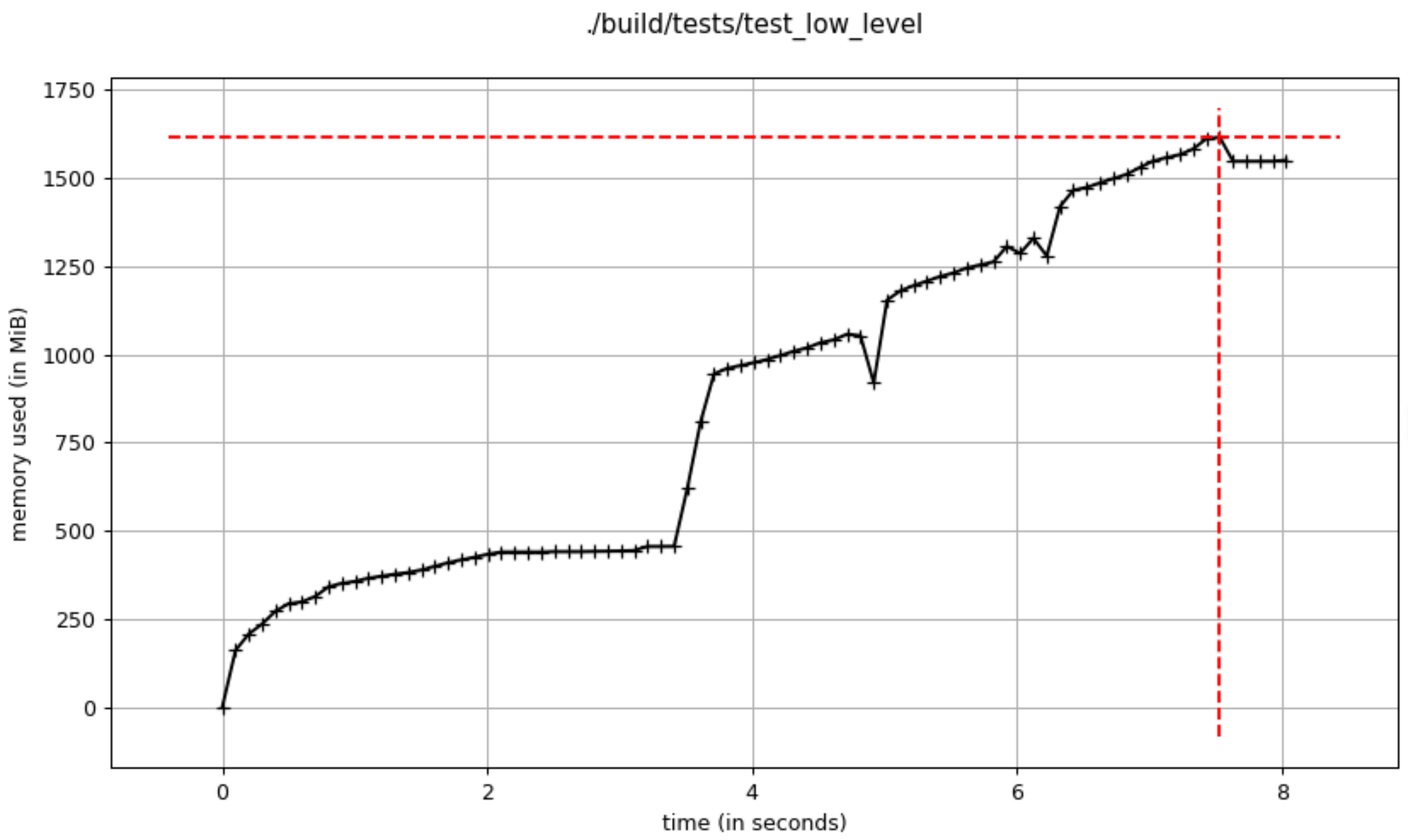
# Simple C++ analysis

DataSource:




- Max RAM: 2.6 GiB
- Run time: 23 s

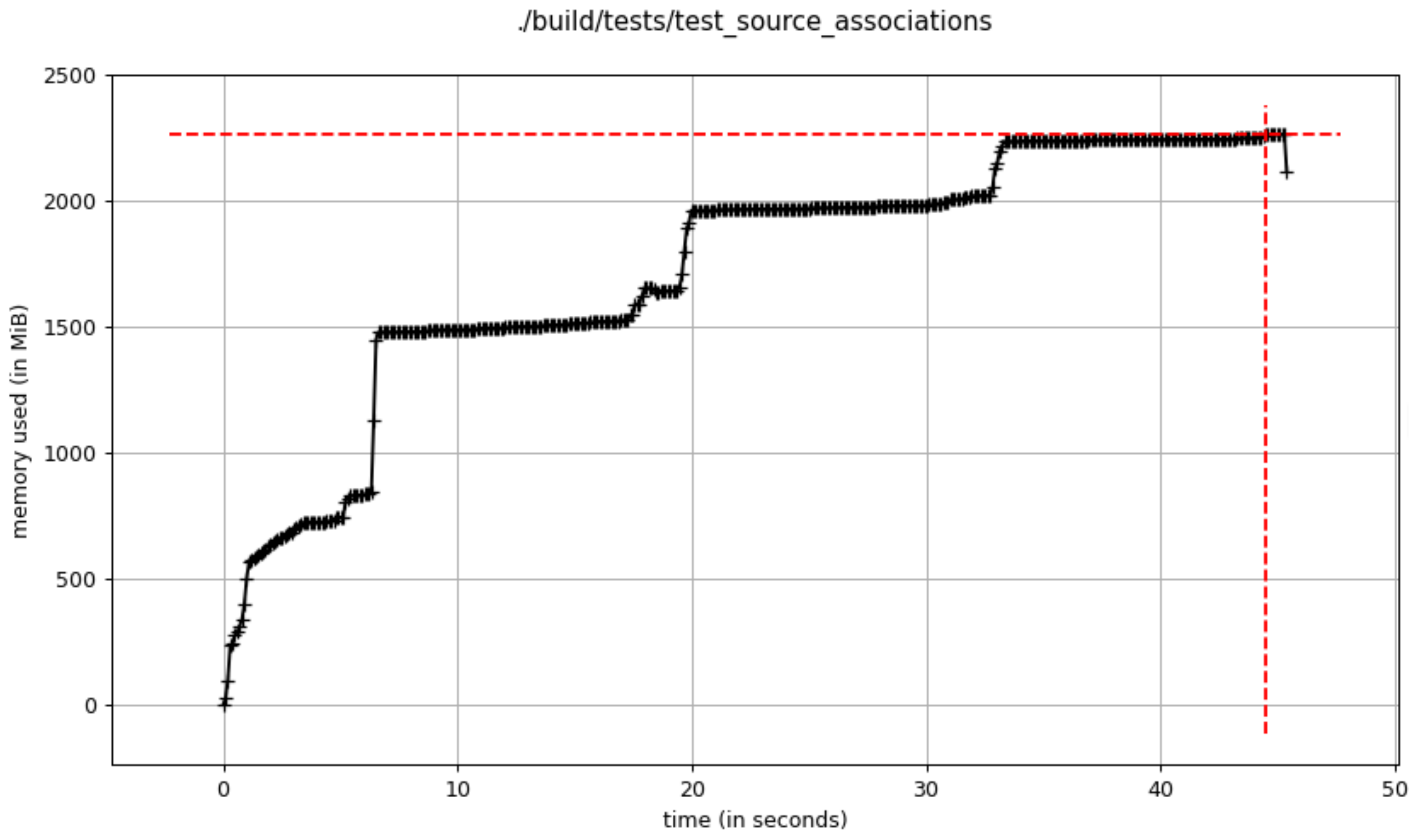
Current implementation:




- Max RAM: 1.6 GiB
- Run time: 8 s

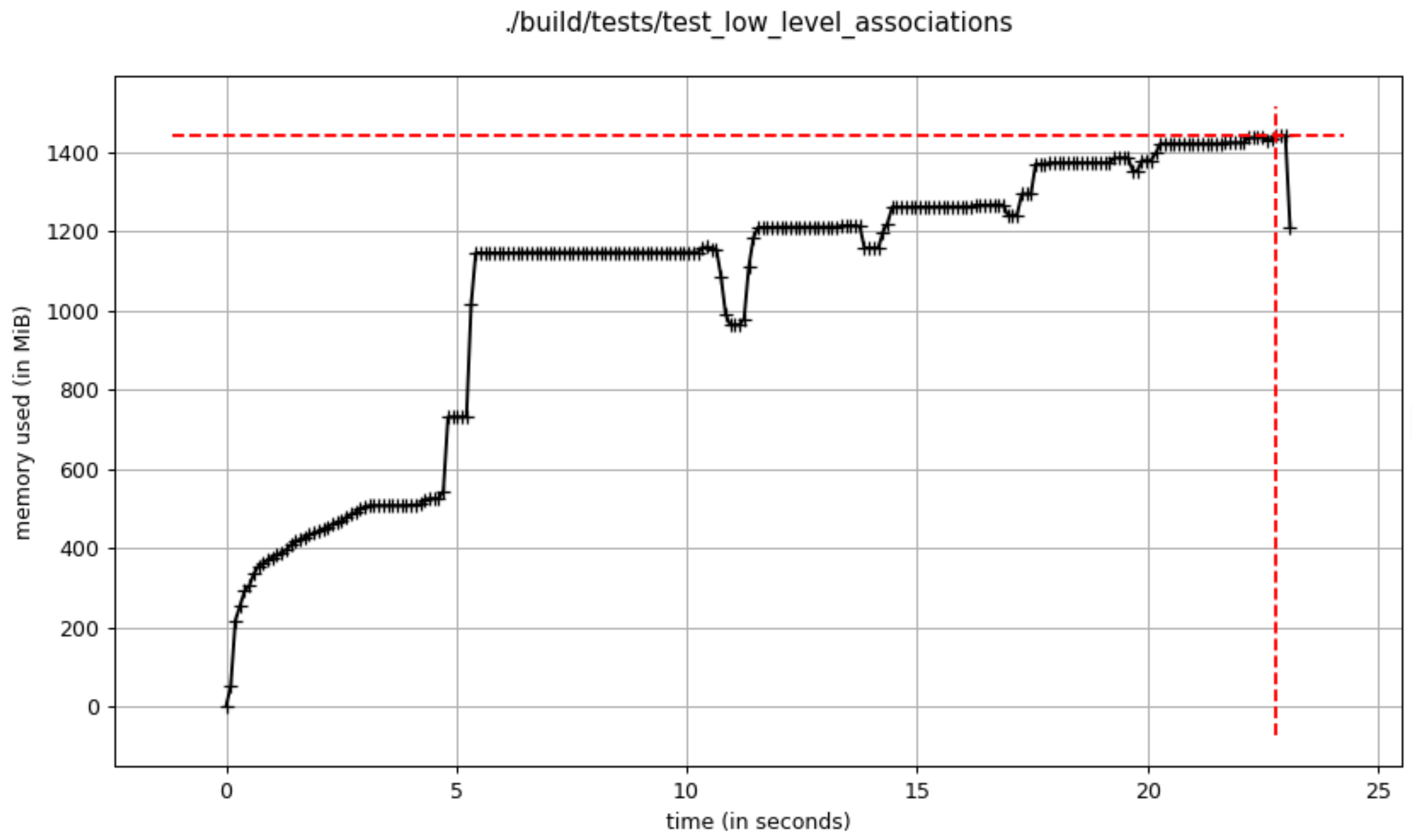
# C++ analysis with Associations

DataSource:




- Max RAM: 2.3 GiB
- Run time: 45 s

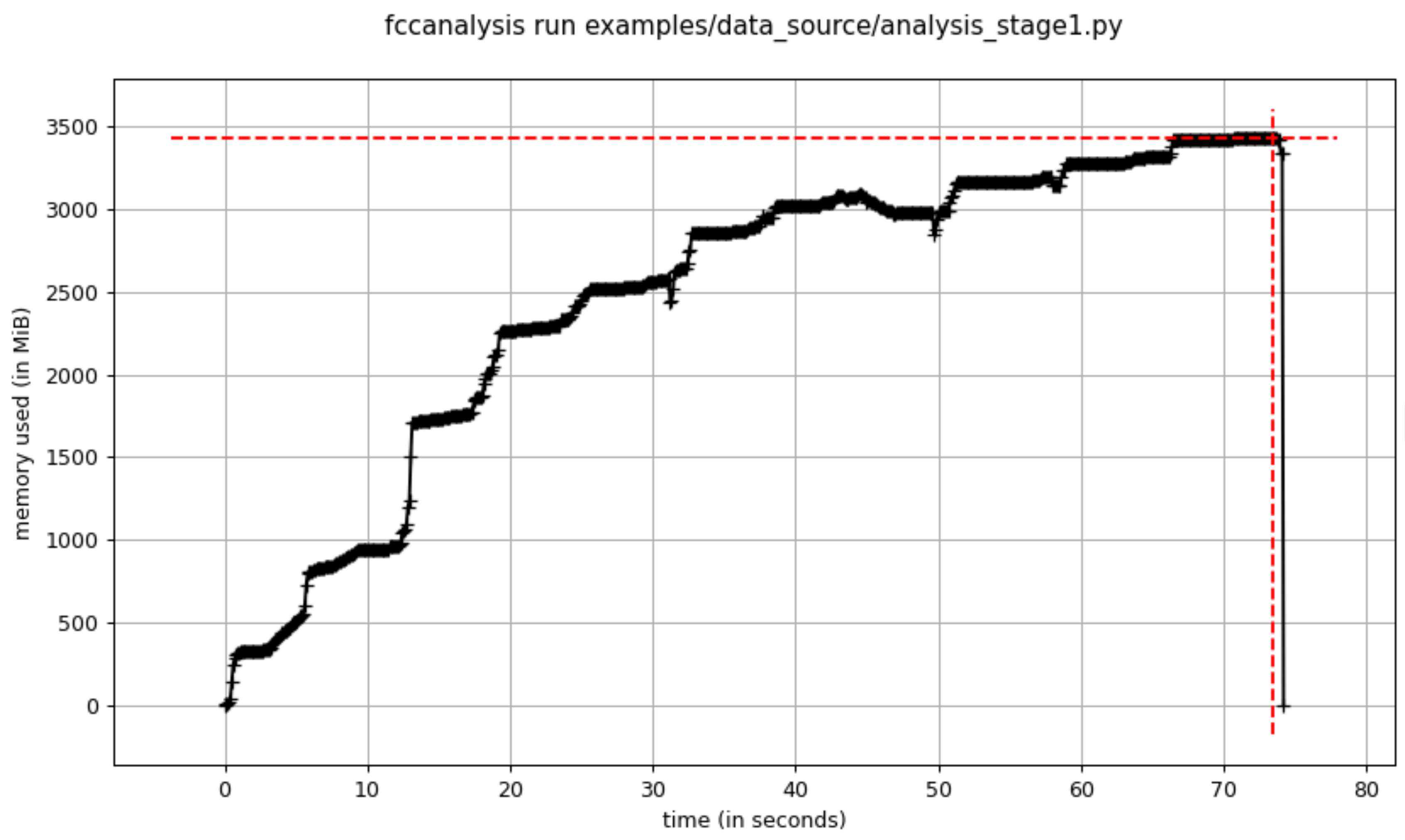
Current implementation:




- Max RAM: 1.45 GiB
- Run time: 23 s

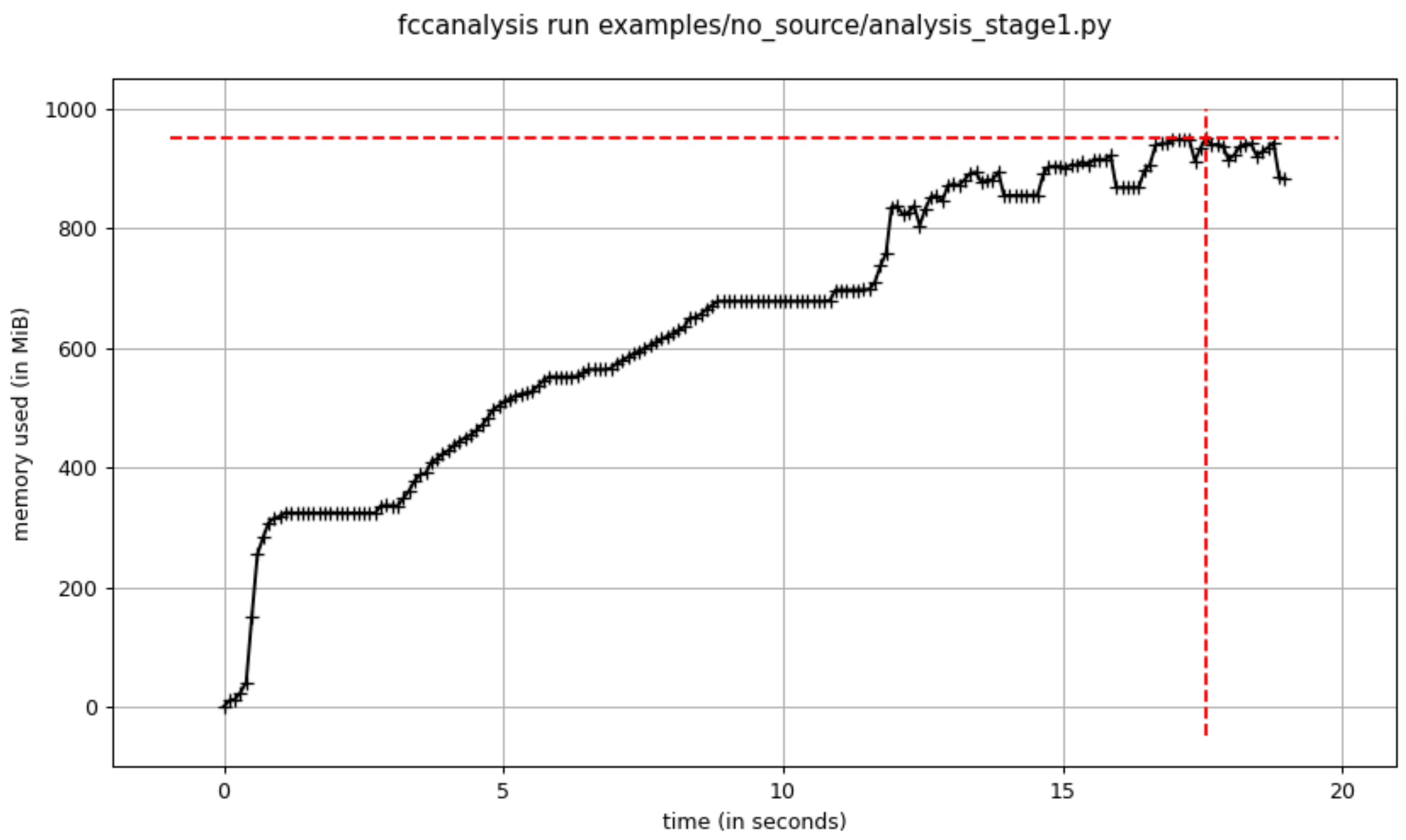
# FCCAnalyses stage 1 (Python)

DataSource:




- Max RAM: 3.4 GiB
- Run time: 75 s

Current implementation:




- Max RAM: 0.95 GiB
- Run time: 19 s

# Documentation

Multiple sources of documentation

- FCC Tutorials: https://hep-fcc.github.io/fcc-tutorials/
  - Focused on providing a tutorial on a specific topic
- Code reference: https://hep-fcc.github.io/FCCAnalyses/doc/latest/
  - Provides details about implementation of individual analyzers
- Manual pages:
  - Info about commands directly in the terminal: `man fccanalysis`
- FCCAnalyses website, FCCSW website

# Conclusions

- Primary focus of EDM4hep is in Reconstruction --- dense event format
- Current index management is tedious and error prone.
- Changes to EDM4hep/Podio require regenerating samples
- With podio::(ROOT)DataSource:
  - Analyzers work with fully fledged EDM4hep objects
  - Schema evolution handled by Podio reader
  - Layer between ROOT file and RDataFrame
    - Costs: 2x RAM and 2x CPU time
- Podio PR: #593, FCCAnalyses PR: #309

# Backup

# Example analysis

The Higgs boson mass and σ(ZH) from the recoil mass with leptonic Z decays (link)

```
#Mandatory: List of processes
processList = {
    'p8_ee_ZZ_ecm240':{'fraction':0.005},#Run the full statistics in one output file named <outputDir>/p8_ee_ZZ_ecm240.root
    'p8_ee_WW_ecm240':{'fraction':0.5, 'chunks':2}, #Run 50% of the statistics in two files named <outputDir>/p8_ee_WW_ecm240/chunk<N>.r
    'p8_ee_ZH_ecm240':{'fraction':0.2, 'output':'p8_ee_ZH_ecm240_out'} #Run 20% of the statistics in one file named <outputDir>/p8_ee_ZH
}

#Mandatory: Production tag when running over EDM4Hep centrally produced events, this points to the yaml files for getting sample statist
prodTag     = "FCCee/spring2021/IDEA/"

#Optional: output directory, default is local running directory
outputDir   = "outputs/FCCee/higgs/mH-recoil/mumu/stage1"

#Optional: analysisName, default is ""
#analysisName = "My Analysis"

#Optional: ncpus, default is 4
#nCPUS       = 8

#Optional running on HTCondor, default is False
#runBatch    = False

#Optional batch queue name when running on HTCondor, default is workday
#batchQueue = "longlunch"

#Optional computing account when running on HTCondor, default is group_u_FCC.local_gen
#compGroup = "group_u_FCC.local_gen"

#Optional test file
testFile ="root://eospublic.cern.ch//eos/experiment/fcc/ee/generation/DelphesEvents/spring2021/IDEA/p8_ee_ZH_ecm240/events_101027117.roo

#Mandatory: RDFanalysis class where the use defines the operations on the TTree
class RDFanalysis():
```