Software Ecosystem for DRD6

Brieuc Francois (CERN)

DRD6 Collaboration Meeting

April 11th, 2024

FUTURE CIRCULAR COLLIDER

CERN

Content

- Detector R&D and software
- Key4hep Overview
- Key4hep for DRD6
  - What is already available to serve DRD6 needs
- Organization

Detector R&D and software

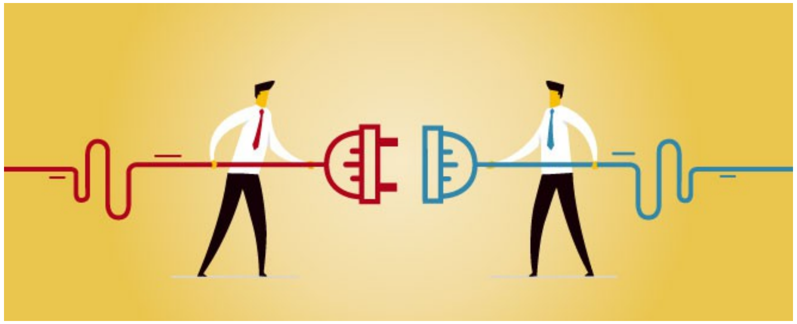
- Detector R&D activities rely on many different software (SW) packages
  - Ranging from commercial Finite Element Analysis tools (usually Windows) to custom made DAQ systems
    - And including detector geometry description, digitization, reconstruction, analyses, distributed computing, …
- It is difficult to reach the "one solution fits all needs" → SW **ecosystem**
  - But we should try get as close as we can
- Why?
  - Starting software from scratch brings important overhead → inefficient usage of manpower if every Work Package (WP) develops their own solutions
    - And R&D/test-beam teams are usually small...
  - Code sharing across DRD6 WP's (and across DRD's)
    - Practical only if we agree in advance on technology choices (consistent framework)
  - Re-using code (with minimal modification) over e.g. different test-beam campaigns of a WP
    - Needs a continuously maintained framework

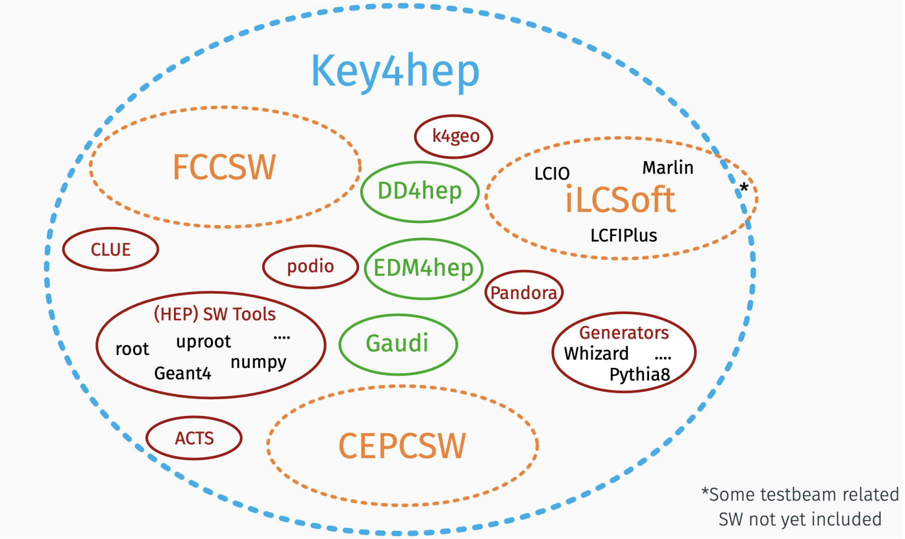
DRD Data Persistency

- DRD's will **produce** a lot of **valuable data** (e.g. test-beams)
  - These data should be accessible and analyzable by people not directly involved in their production
- DRD's will span over decades
  - The **data should remain readable over long periods of time** (target should be “forever”)
    - Release/versioning scheme, backward compatibility
  - But the technology choice has to be made 'today'
    - Does not mean it can/will not evolve
- A good candidate for DRD6 **software ecosystem** should be “**modern**”, used by a **large community** and with good chances to be **maintained over the long run**
  - Key4hep would be a natural choice to develop (most) DRD6 software
  - Win-win situation
    - Key4hep already meets a lot of DRD6 needs (profit from existing component)
    - Seamlessly port DRD6 developments (e.g. from test beams) to the more general future collider Full Sim studies (already using Key4hep)

Key4hep Overview

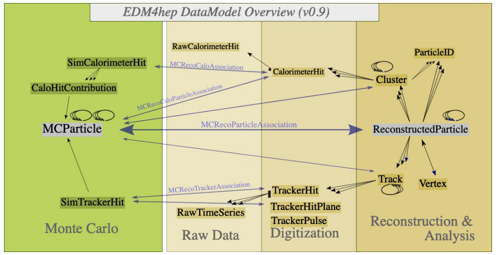
Key4hep Philosophy

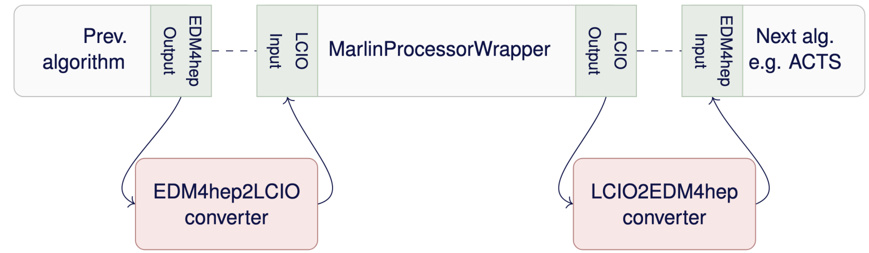




- Key4hep is a **software framework** serving (and developed by) the **future collider community**
- **Key4hep** guiding principles
  - **Interoperability:** what is developed by some should be useable by others (with minimal modifications)
  - **Versatility:** covers a large spectrum of needs (serves diverse facilities and detectors)
  - **Flexibility:** still under active development (nothing is frozen), targets "the future" → has to adapt to evolving needs, detector configurations, etc

Key4hep Building Blocks



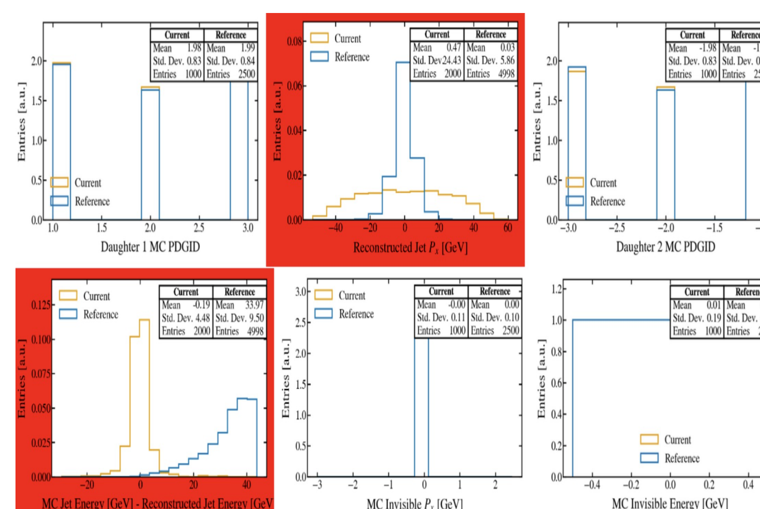


- Key4hep building blocks: state of the art software with active user community
  - Algorithm orchestration framework: **Gaudi** (LHCb, ATLAS)
  - Data format for algorithm input/output: **edm4hep**
    - ROOT based, inspired by lcio (used in CALICE) and FCC-edm, built with **PODIO** (more later)
  - Detector geometry description: **DD4hep** (LHCb, CMS, …)
  - Package manager building the software stack: **Spack** (1.3k+ contributors)
- Interoperability enabled by the compliance to the above (aka being 'Key4hep compliant')
- When possible/practical, prefer interfacing existing solutions over starting from scratch
  - Inspired by LHC solutions and integrates e.g. iLCSoft packages (used in CALICE)
    - Avoid re-inventing the wheel
    - Can be used with edm4hep data (converter), as Gaudi algorithms (wrappers)

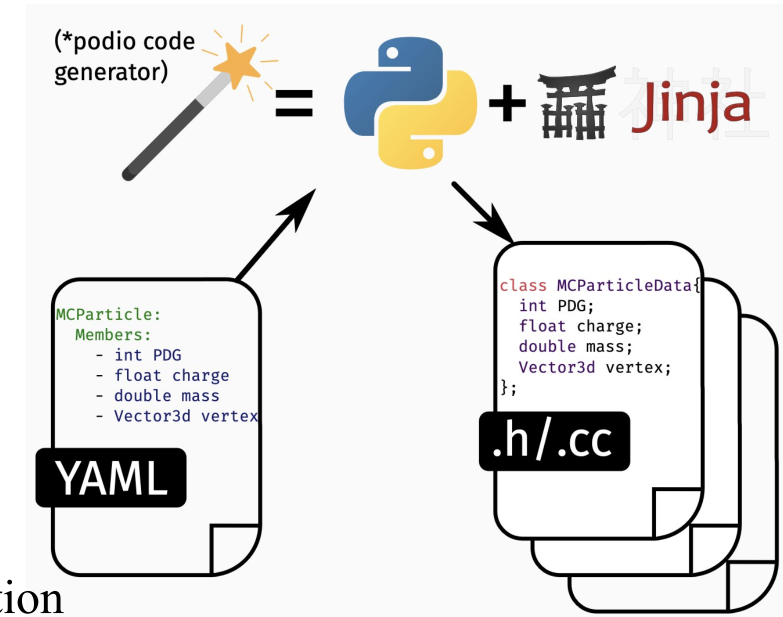
Key4hep Releases

```
source /cvmfs/sw.hsf.org/key4hep/setup.sh
```

- With cvmfs mounted, sourcing one script gives you access to the whole Key4hep stack
  - Plethora of consistently built **HEP software packages** (ROOT, Geant4, KKMC, Whizard, PyTorch, …) and **future collider user codes** e.g. FCC/ILC calos reconstruction chains
- Two types of release
  - **Stable releases**: built ~twice a year, guaranteed stability, availability on the long run
  - **Nightly releases**: built every day, stability not guaranteed, includes greatest and latest versions of all packages, short-lived
    - For development purposes
- Supported Operating System: AlmaLinux9, Ubuntu 22
  - Docker images available
- A lot of testing functionalities are in place
  - Building
  - Running unit tests and/or complete algorithm chains
  - Physics based validation



- EDM4hep is the data model of Key4hep
  - Input/Output of all components
    - A must to exploit synergies
    - Full freedom on what is used internally
- EDM4hep data model generated by podio
  - C++ classes and Python bindings automatically generated from a simple YAML data description
  - User friendly interface, including relations between collection
  - Schema evolution: can still read 'old data' with newer versions of EDM4hep (from v1.0 onwards)



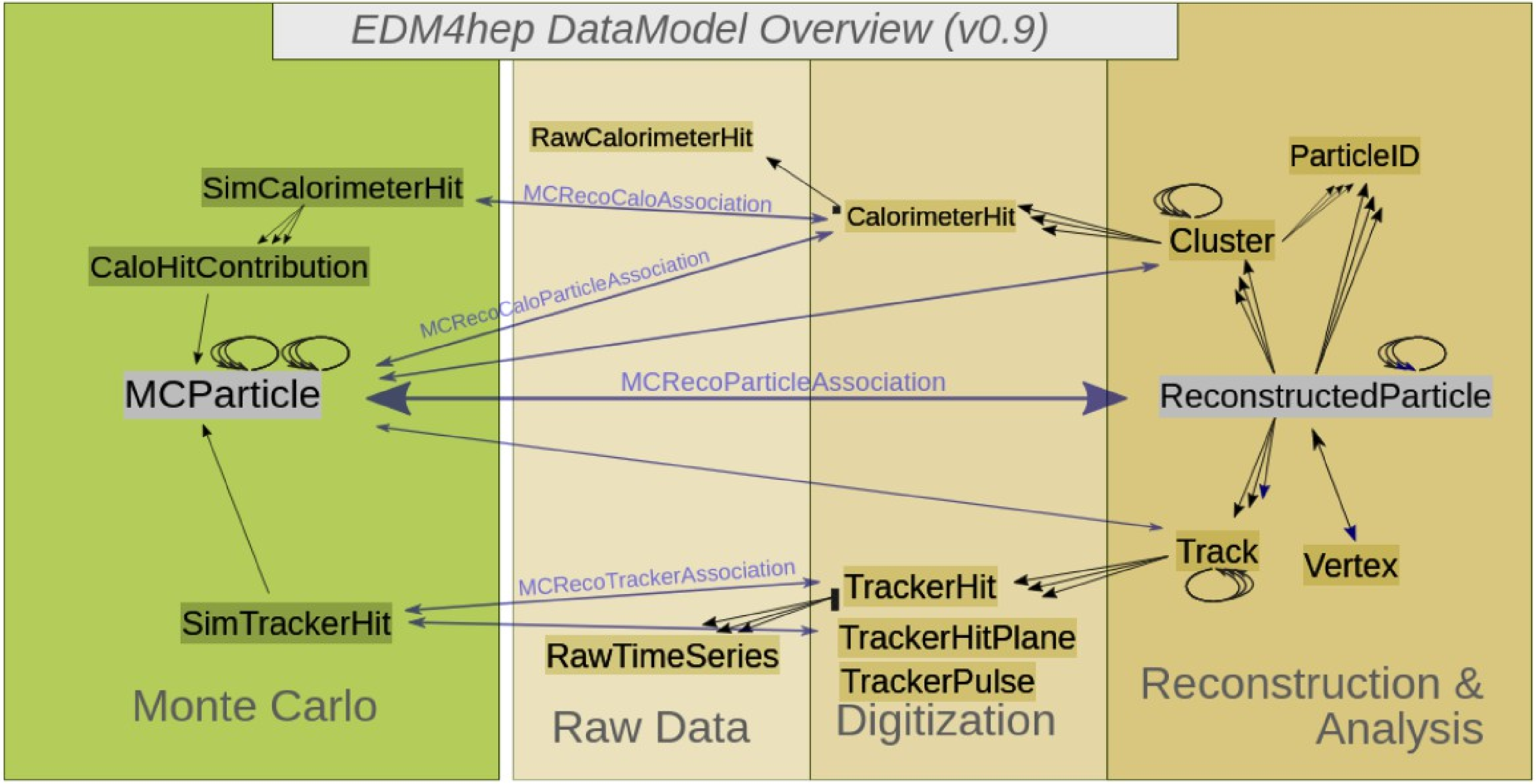

```cpp
auto recos = edm4hep::ReconstructedParticleCollection();

// ... fill, e.g. via
auto p = recos.create();
// or via
auto p2 = edm4hep::ReconstructedParticle();
recos.push_back(p2);

// Loop over a collection
for (auto reco : recos) {
  auto vtx = reco.getStartVertex();
  // do something with the vertex

  // loop over related tracks
  for (auto track : reco.getTracks()) {
    // do something with this track
  }
}
```

```python
recos = edm4hep.ReconstructedParticleCollection()

# ... fill, e.g. via
p = recos.create()
# or via
p2 = edm4hep.ReconstructedParticle()
recos.push_back(p2)

# Loop over a collection
for reco in recos:
  vtx = reco.getStartVertex()
  # do something with the vertex

  # loop over related tracks
  for track in reco.getTracks():
    # do something with the tracks
```

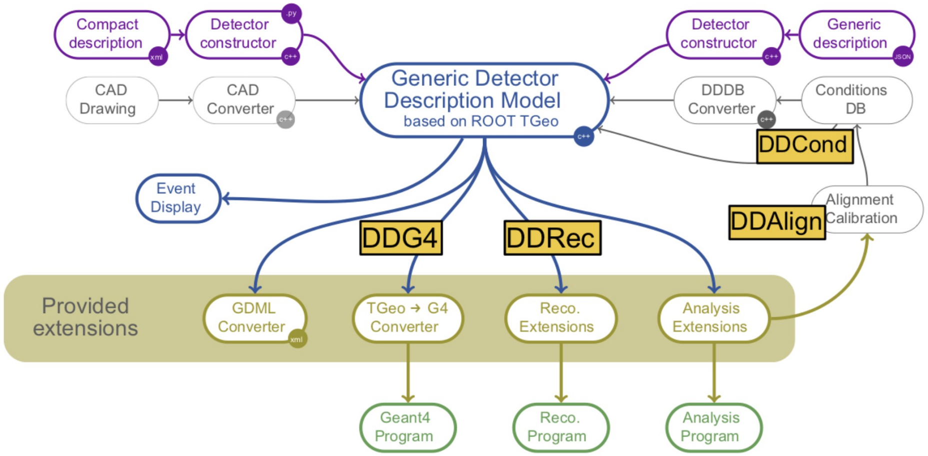
T. Madlener

EDM4hep

- Schematic view of the EDM4hep data model (some changes coming with version > 1.0)
  - Easy to extend to complete the data model if needed
    - First user defined extension for development → upstream to main edm4hep once finalized



DD4hep



- **DD4hep**: generic **detector geometry implementation framework** supporting the full life cycle of the experiment
  - Conceptualization, optimization, construction and operations
- Whole detector description from a single source of information
  - Geometry, materials, readout, alignment, calibration, …
  - Accessible from simulation, visualization, reconstruction and analysis
- Now a **community standard:** CMS, LHCb, EIC, ILC, CEPC, FCC, …
- Convenient factorization enables the **plug and play approach**
  - C++ for generic geometry structure construction (where complexity hides)
  - Very simple XML configuration for detector specific implementations (dimensions, materials, readout granularity, etc)

Key4hep for DRD6

Implementing Detector Geometries

- Implementing the **detector geometry description in DD4hep**
  - DD4hep detector examples: from simple shape based to complex geometries
  - **Many future collider detector geometries available** in k4geo, useful to take inspiration
    - Full sub-detectors (Si-W, Noble Liquid, Dual-Readout, TileCal, …) and test-beam prototypes (CALICE)
  - Many experts happy to provide support!
- Some studies require more than a calorimeter
  - How is my sub-detector performing in a full detector concept?
  - How to optimize my sub-detector based on high level quantities?
- In DD4hep, a **sub-detector can be plugged in an existing full detector** concept with minimal work (varies depending on how the det. concept was implemented)
  - E.g. ALLEGRO ECAL plugged in CLD to start ParticleFlow developments
  - Many available in k4geo: ALLEGRO, CLIC, CLD, IDEA, ILD, SiD, …
- DD4hep provides tools to visualize the detector, handle detector condition data (DDCond), alignment (DDAlign), …
  - Still under active development, can integrate potential uncovered DRD6 need

Simulation

- Recommended way to interact with Geant4: **ddsim** (part of DD4hep)
  - Supports **many generator output formats**: stdhep, hepmc, hepevt
  - Several particle guns available (including Geant4 GPS)
  - Passes Gen level particles to Geant4, taking into account e.g. charged particle with macroscopic displacement that have to be curved due to magnetic fields (not accessible to generators)
  - **Highly configurable**: PhysicsList, SensitiveActions, rangeCut, …
    - Command line interface or (better) configured with a Python steering file
    - *ddsim -h* and *ddsim –dumpSteeringFile* for more details
  - Produces two edm4hep collections per calorimeter readout
    - *SimCalorimeterHit* with **energy per readout cell**, linked to *CaloHitContribution* being **single energy deposits inside the cell** (only the latter has time information)
  - Get linking between CaloHitContribution and MCParticle out of the box

```
#------------- SimCalorimeterHit
edm4hep::SimCalorimeterHit:
  Description: "Simulated calorimeter hit"
  Author: "EDM4hep authors"
  Members:
    - uint64_t cellID       // ID of the sensor that created this hit
    - float energy [GeV]                   // energy of the hit
    - edm4hep::Vector3f position [mm]      // position of the hit in world coordinat
  OneToManyRelations:
    - edm4hep::CaloHitContribution contributions  // Monte Carlo step contributions
```

```
#------------- CaloHitContribution
edm4hep::CaloHitContribution:
  Description: "Monte Carlo contribution to SimCalorimeterHit"
  Author: "EDM4hep authors"
  Members:
    - int32_t   PDG                       // PDG code of the shower particle that caused this contribution
    - float energy [G]                   // energy of the this contribution
    - float time [ns]                    // time of this contribution
    - edm4hep::Vector3f stepPosition [mm] // position of this energy deposition (step)
  OneToOneRelations:
    - edm4hep::MCParticle particle     // primary MCParticle that caused the shower responsible for this contribution to the hit
```

Digitization

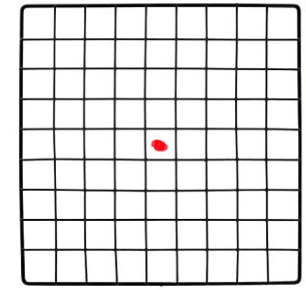
- Software digitizers are Gaudi algorithms receiving *edm4hep::SimCalorimeterHit (or edm4hep::RawCalorimeterHit/TimeSeries)* and outputting *edm4hep::CalorimeterHit*
  - A lot of similarities across calorimeter technologies → can share common generic algorithms
- Several (simple) digitization algorithms already available in Key4hep
  - ALLEGRO: CreateCaloCells
    - Cell calibration (sampling fraction), noise, zero suppression (planning a more detailed version)
    - Used both for Noble Liquid ECAL and the TileCal HCAL (quite generic)
  - ILCSoft digitization: DDCaloDigi (lcio data format based)
    - Used e.g. for CLD ECAL and HCAL with edm4hep format through wrappers/converters
  - Detailed Dual-readout digitization: DigiSiPM based on SimSiPM (full waveform)

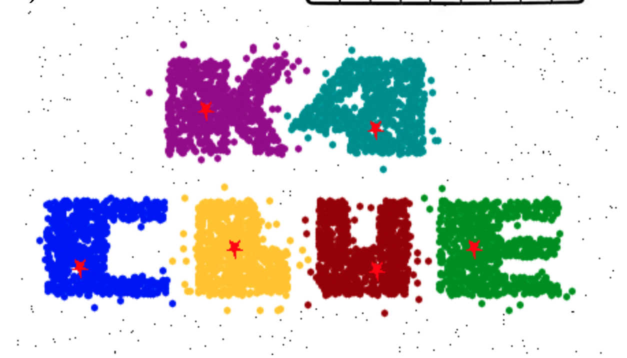
```
#------------- RawCalorimeterHit
edm4hep::RawCalorimeterHit:
  Description: "Raw calorimeter hit"
  Author: "EDM4hep authors"
  Members:
    - uint64_t cellID   // detector specific (geometrical) cell id
    - int32_t amplitude             // amplitude of the hit in ADC counts
    - int32_t timeStamp             // time stamp for the hit
```

```
#------------- 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
```

Reconstruction Algorithms

- **Three calo clustering algorithms** available in Key4hep
  - CreateCaloClustersSlidingWindow
    - Simple sliding window with fixed size (ALLEGRO E/HCAL)
  - CaloTopoCluster
    - Find seeds and iteratively collects cells in several steps of S/N thresholds (ALLEGRO E/HCAL)
  - k4CLUE (originates from CMS)
    - Fast, energy-density based algorithm. Already applied on LAr, CLD and CLIC ECALs 2311.03089
    - Only 2D clustering for now (per longitudinal layer)
    - k4Clue3D on its way
- **Particle Flow**: PandoraPFA available through Wrappers/Converters
  - Used in CLD, and CLD with LAr ECAL
  - k4PandoraPFA under development






```
#------ Cluster
 edm4hep::Cluster:
   Description: "Calorimeter Hit Cluster"
   Author: "EDM4hep authors"
   Members:
     - int32_t              type          // flagword that defines
     - float             energy [GeV]     // energy of the cluster
     - float             energyError [GeV]  // error on the energy
     - edm4hep::Vector3f   position [mm]     // position of the clust
     - std::array<float,6>  positionError  // covariance matrix of the
     - float             iTheta         // intrinsic direction of cl
     - float             phi            // intrinsic direction of cl
     - edm4hep::Vector3f   directionError [mm**2]  // covariance matrix
   VectorMembers:
     - float    shapeParameters      // shape parameters. This should be
     - float    subdetectorEnergies  // energy observed in a particular s
   OneToManyRelations:
     - edm4hep::Cluster        clusters     // clusters that have been c
     - edm4hep::CalorimeterHit hits         // hits that have been combi
     - edm4hep::ParticleID     particleIDs  // particle IDs (sorted by t
```

Online Software & Data Preservation

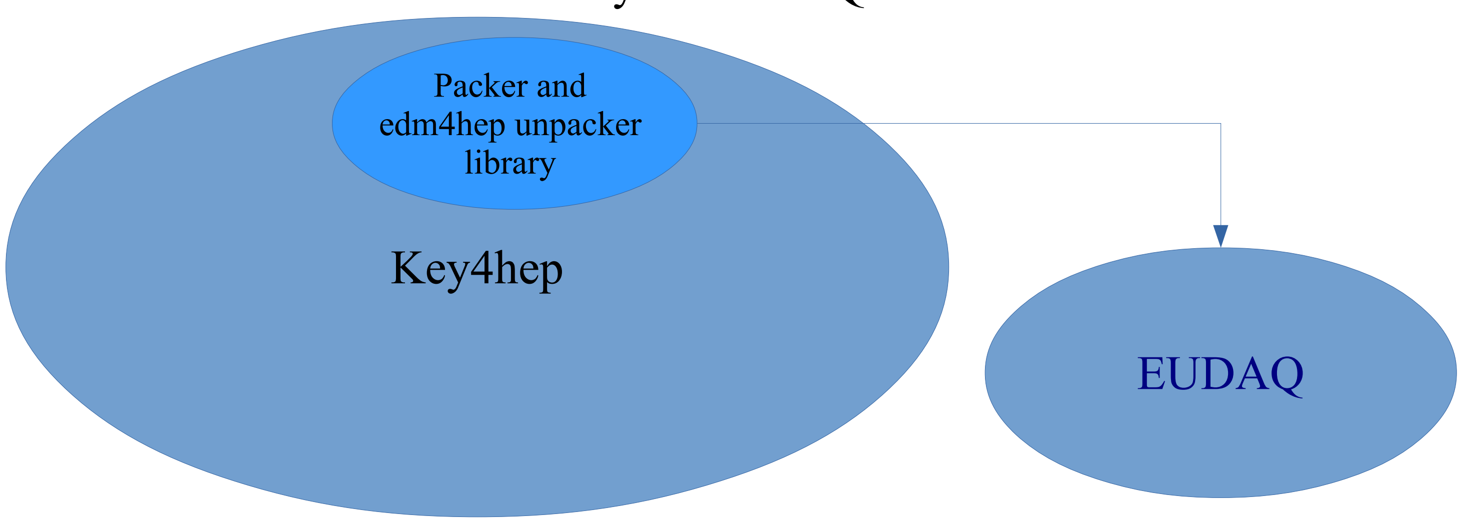
- People developing online software solutions may not need (nor want) the full Key4hep stack
  - "Low level" code with little dependencies, lose requirements in terms of e.g Operating System, should be possible to run it without internet connection (no cvmfs)
  - More details in Gerald's and Andreas' talks
- Encouraging DRD(6) collaborators to use a common framework for online software development would still be highly beneficial (e.g. EUDAQ)
  - Not re-inventing the wheel, share code and expertise, …
- **Data preservation?**
  - After/during data taking, **translate EUDAQ produced data into edm4hep format**
  - **Schema evolution guarantees to be able to read these data on the long run**

```
#---------- RawTimeSeries
edm4hep::RawTimeSeries:
  Description: "Raw data of a detector readout"
  Author: "EDM4hep authors"
  Members:
    - uint64_t cellID  // detector specific cell id
    - int32_t quality             // quality flag for the hit
    - float time [ns]             // time of the hit
    - float charge [fC]           // integrated charge of the hit
    - float interval [ns]         // interval of each sampling
  VectorMembers:
    - int32_t adcCounts         // raw data (32-bit) word at i
```


```
#------------  RawCalorimeterHit
edm4hep::RawCalorimeterHit:
  Description: "Raw calorimeter hit"
  Author: "EDM4hep authors"
  Members:
    - uint64_t cellID   // detector specific (geometrical) cell id
    - int32_t amplitude             // amplitude of the hit in ADC counts
    - int32_t timeStamp             // time stamp for the hit
```

Online/Offline Software Compatibility

- How to guarantee synchronization between data frame format definition (EUDAQ) and the unpacker (Key4hep)?
  - Include EUDAQ in Key4hep (releases ensure compatibility and reproducibility)
    - Does not mean EUDAQ users need Key4hep
  - Or factor the **packer/unpacker** out of EUDAQ, as a **library living in Key4hep**
    - Library depends on edm4hep but is free from EUDAQ dependencies (packer only manipulates low-level objects)
    - Interface this 'external' library to EUDAQ



Further Topics

Further topics that could not be covered in details

- Porting test-beam data to Geant4 validation: benefits to the whole HEP community and more
  - See Lorenzo's talk
- Machine learning
  - PyTorch available in Key4hep
    - Training can potentially be done outside of Key4hep
  - Models from any other machine learning framework (e.g. TensorFlow) can be evaluated inside Key4hep through the ONNX interface
- Cluster correction
  - Several Key4hep Gaudi algorithms already exists: upstream/downstream energy correction, MVA based calibration (Noble Liquid ECAL)
- Flat Tree Producer + Analyses: having common DRD6 tools is also important here
  - FCCAnalyses available in Key4hep: RDataFrame based analysis tools of edm4hep events
  - A “caloNtupleizer” is already there and can be extended or used as example
    - Produces a plain root tree easy to manipulate for analysis
  - Plain C++ version ongoing
  - Performance plotter to be written

```
ROOT::VecOps::RVec<std::vector<float>>
getCaloCluster_energyInLayers (const ROOT::VecOps::RVec<edm4hep::ClusterData>& in,
                               const ROOT::VecOps::RVec<edm4hep::CalorimeterHitData>& cells,
                               const int nLayers) {
```

```
ROOT::VecOps::RVec<float> getSimCaloHit_phi (const ROOT::VecOps::RVec<edm4hep::SimCalorimeterHitData>& in){
```

Organization

Code Organization

- Key4hep has a highly factorized structure
  - Many “task oriented” GitHub repositories
  - Minimizes frictions between independent developments
- The DRD6 needs being quite specific, it may be interesting to investigate different approaches
  - E.g. having one repository hosting all the components needed for a given test-beam
- While the code should be '**Key4hep compliant**' maybe not everything must be **included in Key4hep** (i.e. made available with the stack environment)
  - Included in Key4hep ↔ can harm the stack if not properly maintained
  - Possible approach
    - Having “common/generic core components” needed for DRD6 activities inside Key4hep, leave specific applications being 'only' Key4hep compliant
    - Still need to maintain “user” code, but won't harm Key4hep if not

Getting Everyone Onboard

- Following “common software policies” and writing generic code with documentation is clearly **beneficial at the collaboration level but** may be seen as adding **overhead** on **individuals**
  - Community effort: how do we get everyone onboard?
- Lower the 'barriers to entry' as mush as we can
  - Hide the complexity for the end users
  - **Github repository template** including dependencies/testing/etc
    - Ready to be used to start a new project: k4-project-template
    - May want a DRD6 specific version (e.g. adding the DD4hep dependency, …)
- The added value of the centrally available tools must overcome the potential overhead of using the common framework
  - Will need some 'pioneering' work
- If transversal working group activities is on a best effort basis, it has to be somehow rewarded

Useful Resource

- Useful Resource
  - Full Sim with Key4hep tutorial
  - FCC Full Sim webpage
  - Key4hep tutorials
  - Bi-weekly FCC Full Sim working meeting, announced on the **FCC-PED-SoftwareAndComputing-Full-Simulation** e-group
  - Key4hep/edm4hep working meetings, announced on the **key4hep-sw** e-group

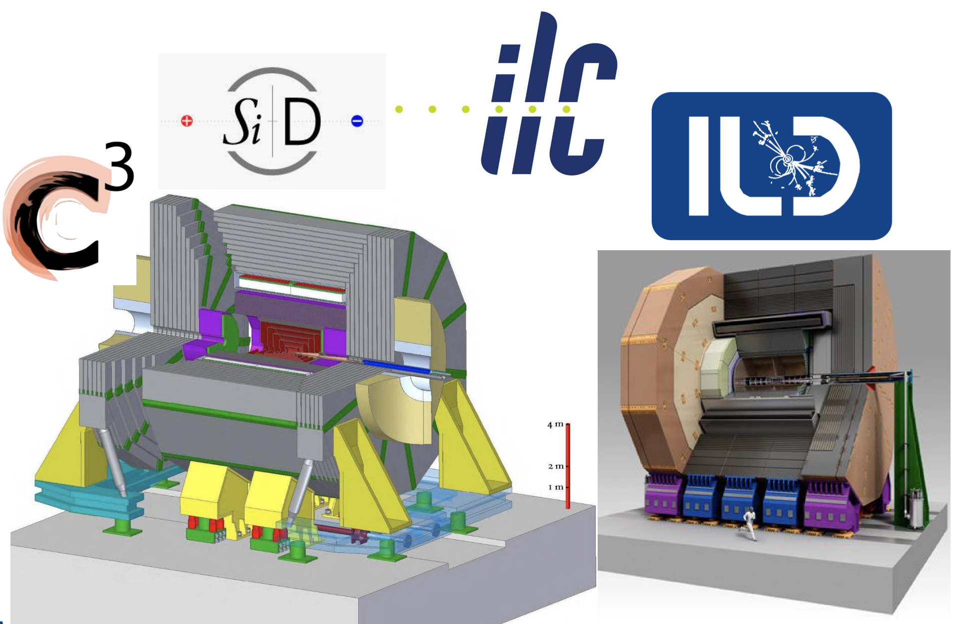
Summary

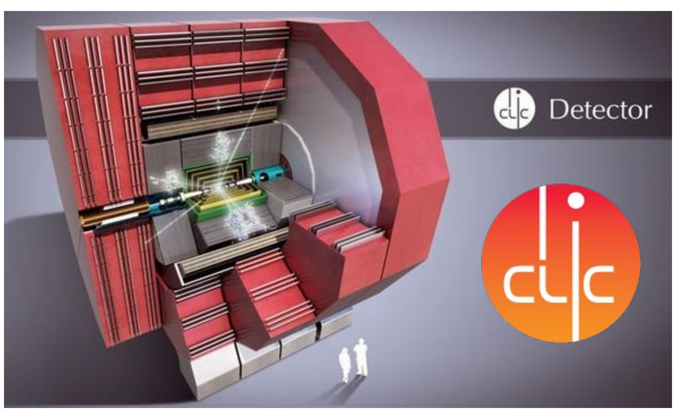
- DRD6 has important software needs
- Using a common software ecosystem will allow us to leverage synergies
  - Across DRD's, across DRD6 WP's and across WP phases
- Data persistency must be a central consideration (valuable datasets will be produced)
- Key4hep is a very good candidate to be the common software base for (most) DRD6 activities
  - Wide (and growing) adoption by the Future Collider Community (but built with LHC experience)
  - Already meets most DRD6 needs (except for online software, likely not integrated in Key4hep, but for which we should still have common standards)
  - Under active development: can be adapted/complemented if needed
- The Key4hep team warmly welcomes new contributors
  - Good opportunity for the DRD6 Transversal Software Working Group!
- Next important step: agree on the set of software tools that we want to set as standards

Thanks to the Key4hep team for the useful feedback and discussions!

Additional material

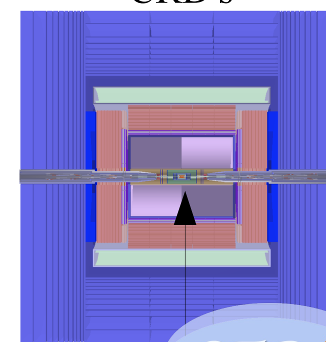
The Detector Zoo





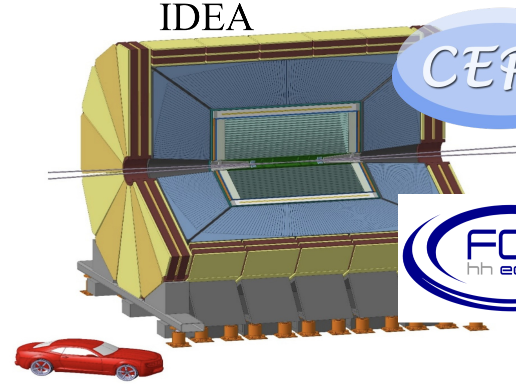
ALLEGRO

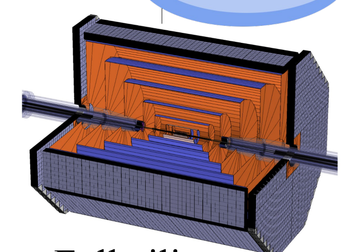
CRD's



ILD like

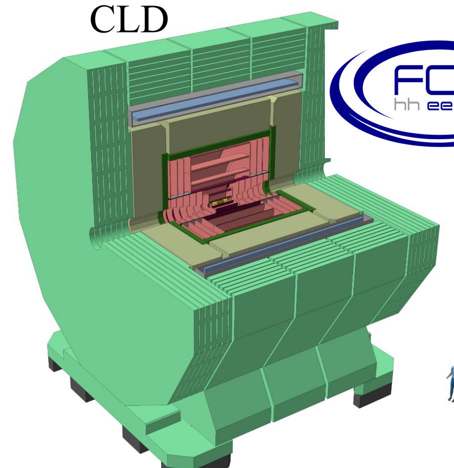
IDEA

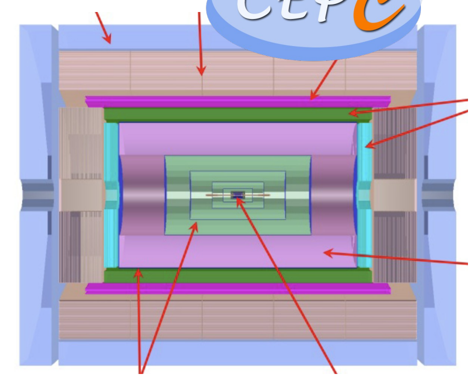




Full-silicon

CLD





4th CEPC concept

Dual Readout Digitization

- Going through the full waveform with SimSiPM

```
#---------- RawTimeSeries
 edm4hep::RawTimeSeries:
   Description: "Raw data of a detector readout"
   Author: "EDM4hep authors"
   Members:
     - uint64_t cellID  // detector specific cell id
     - int32_t quality              // quality flag for the hit
     - float time [ns]              // time of the hit
     - float charge [fC]            // integrated charge of the
     - float interval [ns]          // interval of each sampling
   VectorMembers:
     - int32_t adcCounts         // raw data (32-bit) word at i
```

```
#----------  TimeSeries
 edm4hep::TimeSeries:
   Description: "Calibrated Detector Data"
   Author: "EDM4hep authors"
   Members:
     - uint64_t cellID            // cell id
     - float time [ns]            // begin time
     - float interval [ns]        // interval of each sampling
   VectorMembers:
     - float amplitude            // calibrated detector data
```

```
#-------------  RawCalorimeterHit
edm4hep::RawCalorimeterHit:
  Description: "Raw calorimeter hit"
  Author: "EDM4hep authors"
  Members:
    - uint64_t cellID   // detector specific (geometrical) cell id
    - int32_t amplitude             // amplitude of the hit in ADC
    - int32_t timeStamp             // time stamp for the hit
```

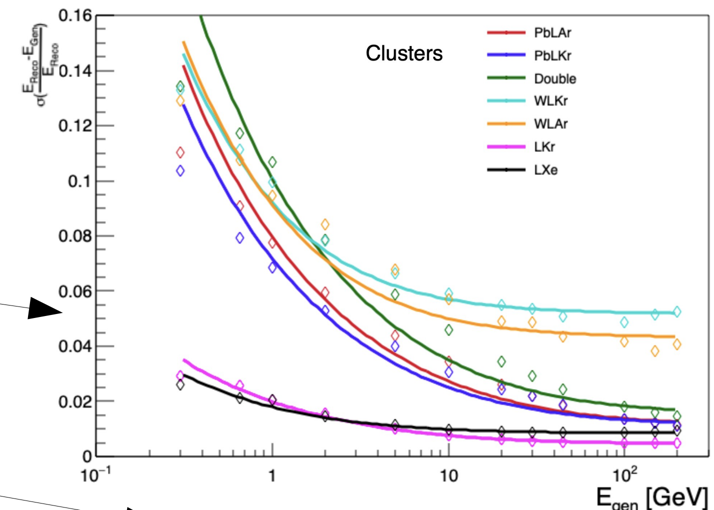
```
#------------  CalorimeterHit
edm4hep::CalorimeterHit:
  Description: "Calorimeter hit"
  Author: "EDM4hep authors"
  Members:
    - uint64_t cellID  // detector specific (geometrical)
    - float energy [GeV]            // energy of the hi
    - float energyError [GeV]       // error of the hit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

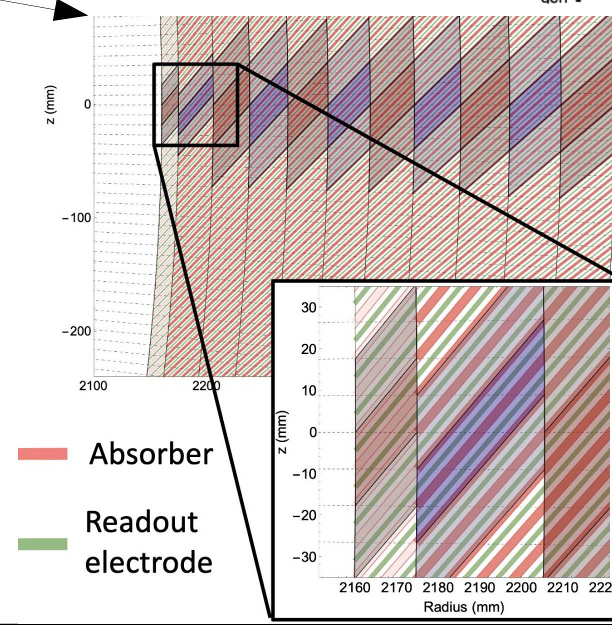
Detector description (I)

- **Factorized** Detector Building (DD4HEP)
  - C++ detector factory: handles the **generic geometry structure**
    - Cryostat cylinder, inclined plates, …
  - XML file with **specific detector parameters**
    - Inner/Outer radius, materials, inclination, …
  - Allows you to study different scenarios with minimal work
- Detector segmentation based on DD4HEP Readouts
- Readout cells differ in general from physical cells
  - E.g. having high sampling frequency improves energy resolution → phi granularity higher than what physics requires
  - Flexibility choice: do the time consuming Geant4 simulation with atomic granularity, then apply (possibly several) cell recombinations with RedoSegmentation
- Caveats
  - The same cell recombination scheme is applied to the whole calorimeter
    - One would like to have smaller cells for e.g. the strip layer
  - The cell position assignment would benefit from a refactoring

FCCDetectors (DD4HEP)

k4SimGeant4





Detector description (II)

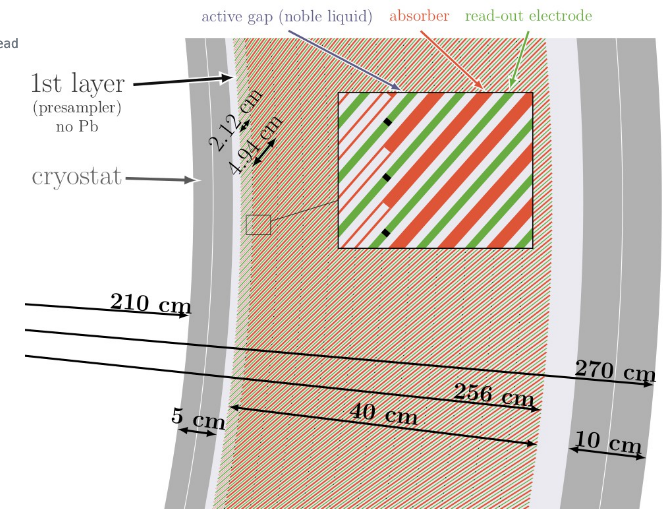
FCCDetectors (DD4HEP)

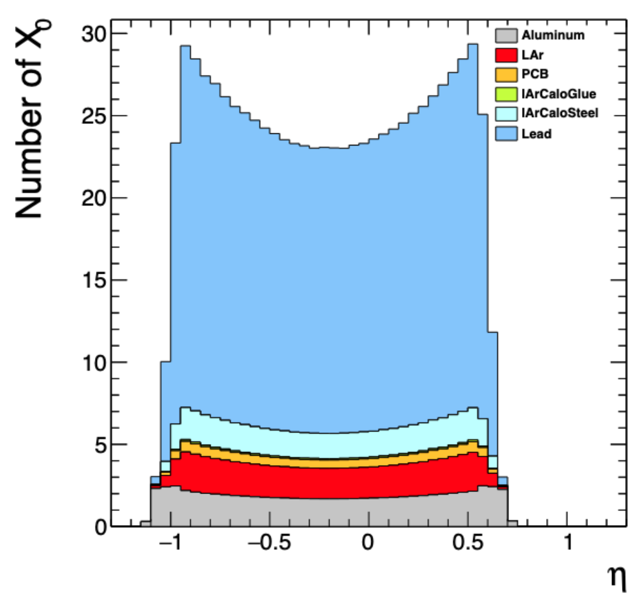
k4SimGeant4

- Accurate FCC-ee detector description
  - 1536 absorber plates: 100 µm Steel sheet + 100 µm glue + 1.8 mm Lead
  - 2 x 1.2 mm sensitive LAr gap (widening towards high radius) + 1.2 mm PCB
  - Typical readout cells size
    - $\theta \times \Phi \times r \sim 2$ (0.5 strip) $\times 1.8 \times 3$ cm^3
  - Cryostat + space reserved for services (filled with LAr)
  - 40 cm depth sensitive area, ~23 X_0 including the cryostat
- So far, **only the** **Barrel ECAL** has been implemented for the FCC-ee geometry
  - Currently working on Endcap LAr ECAL implementation

materials.xml

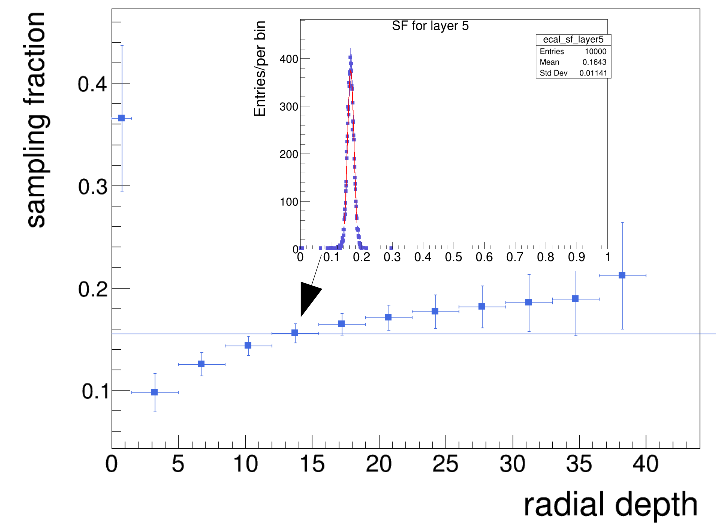
```xml
<!-- prepreg glue (C5H8O4Si) between steel and lead
<!-- from ATL-LARG-PUB-2009-001 -->
<material name="lArCaloGlue">
  <D value="1.69" unit="g/cm3" />
  <composite n="5" ref="C"/>
  <composite n="8" ref="H" />
  <composite n="4" ref="O" />
  <composite n="1" ref="Si" />
</material>

<!-- stainless steel as in ATLAS EM calo -->
<!-- from ATL-LARG-PUB-2009-001 -->
<material name="lArCaloSteel">
  <D value="7.84" unit="g/cm3" />
  <fraction n="0.7175" ref="Fe"/>
  <fraction n="0.19"   ref="Cr" />
  <fraction n="0.0925" ref="Ni" />
</material>
```

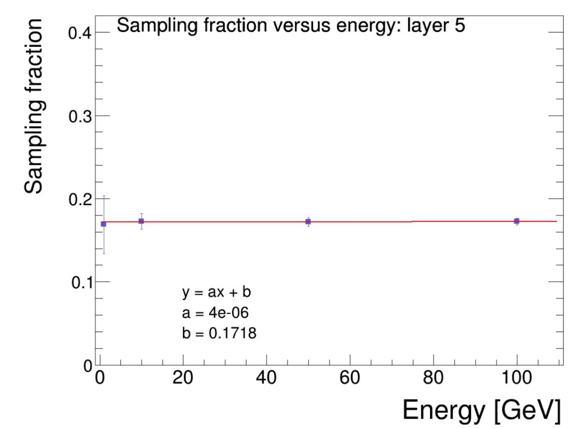




Sampling Fraction

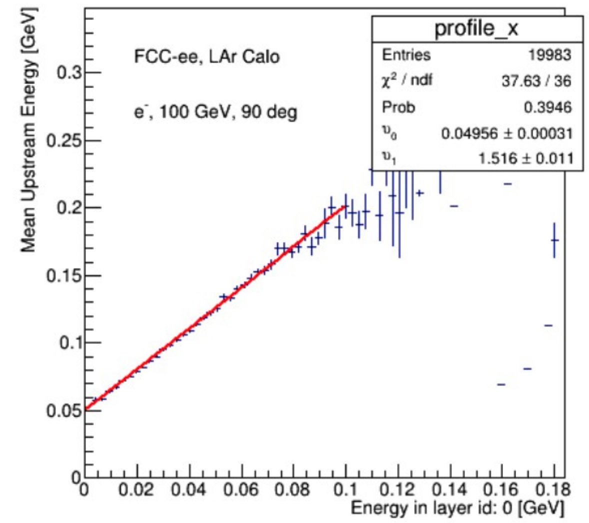
- In a Sampling Calorimeter, only a fraction of the particle energy is measured
  - One scales each cell energy to account for energy deposited in absorber and PCB
- Modified detector config with the absorbers set as sensitive (XML)
  - SamplingFractionInLayers stores the energy ratio (active/passive) per event and per longitudinal layer k4SimGeant4
  - User Code SF = mean of Gaussian fit of the active/passive energy ratio
  - Propagate results to CalibrateInLayersTool k4RecCalorimeter
  - Fully automatized procedure (with control plots)
    - Everything defined in a Gaudi config can be passed as command line argument
    - Or you can use sed for more permanent usage
- In a Noble Liquid calorimeter, the sampling fraction has almost **no dependence on the incident particle energy**
  - No need to apply this procedure to many energy points

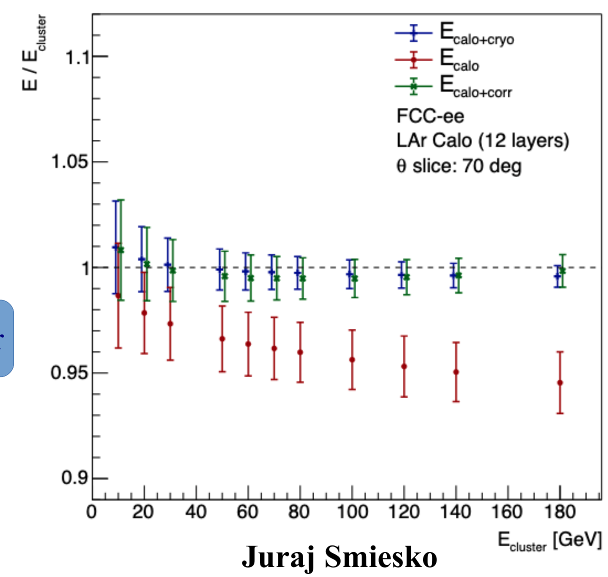




Upstream/Downstream energy correction

- Unmeasured **energy deposited in upstream material**: calorimeter supporting structure/cryostat, magnet, services, …
- Always try to minimize calorimeter radial extent + stochastic nature of shower depth → **energy deposited after the calorimeter**
- **Strong correlation** between energy in first(last) sensitive layer and energy deposited upstream(downstream) → **one can correct for that!**
  - EnergyInCaloLayers → stores energy in various dead materials and in all the active layers (modified XML) k4SimGeant4
  - Centrally available scripts perform the fits
  - CorrectCaloClusters → applies the correction based on cluster total energy and energy from first/last layer k4RecCalorimeter
- Again, fully automatized procedure with intermediate diagnostic plot production

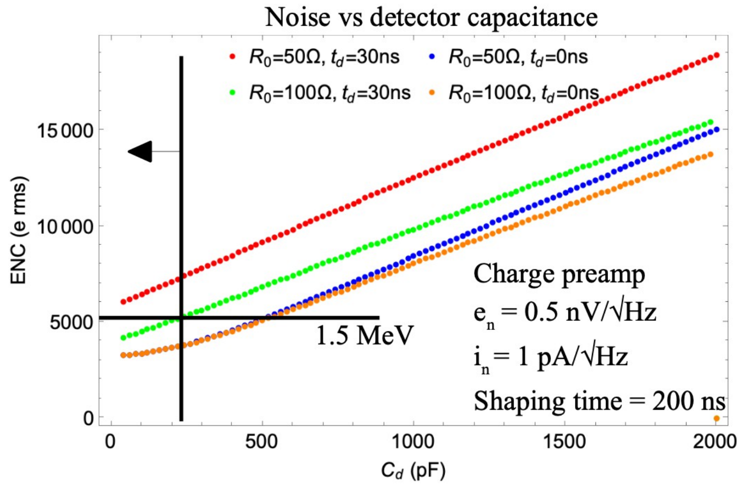


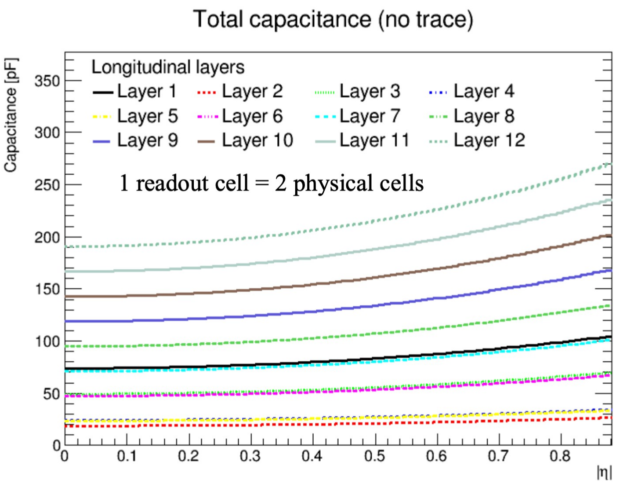


Juraj Smiesko

Noise

- Noise depends on many factors
  - Detector capacitance, signal extraction scheme, front-end electronics, etc...
  - **Estimated outside of the main software** framework: Finite Element Method tools (Ansys) + analytical implementation (Mathematica)
  - Stored in a rootfile, per longitudinal layer and as a function of polar angle
- Introduced in the simulation by NoiseCaloCellsFromFileTool k4RecCalorimeter
  - Random number from Gaussian whose width is taken from the rootfile (layer/Θ dependent)
  - Added only after the final readout segmentation step (cell geometry dependance)
- Very tricky to fully automatize 😐





Practical workflow example

- Little user specific code needed to orchestrate these tools and automatize the sequence

LXPlus HTCondor submission script

- Less events per job towards higher energy → 10 energy points (1 – 200 GeV) with 10 k events takes < 2 hours
- Writes all the subsequent shell scripts

k4Gen | k4SimGeant4 | k4RecCalorimeter

RunCaloSim.py

Shell script →

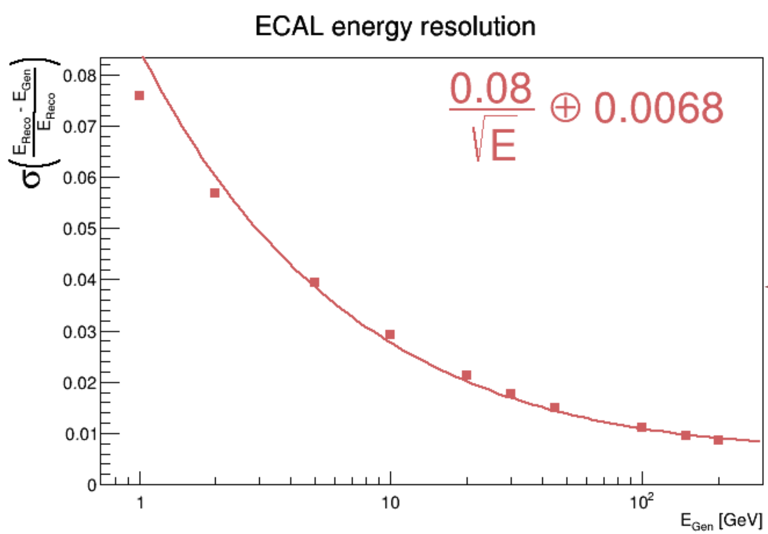
FCCAnalyses CaloNtupleizer

- Outputs plain ROOT n-tuples
- Very easy to analyze with simple macros
- Parametrizable object content
- Can store multiple objects of the same type (e.g. Sliding Window and Topo Clusters)
- Can access DD4HEP geometry information (decode the cellID)

Shell script ↓

Simple analysis macro

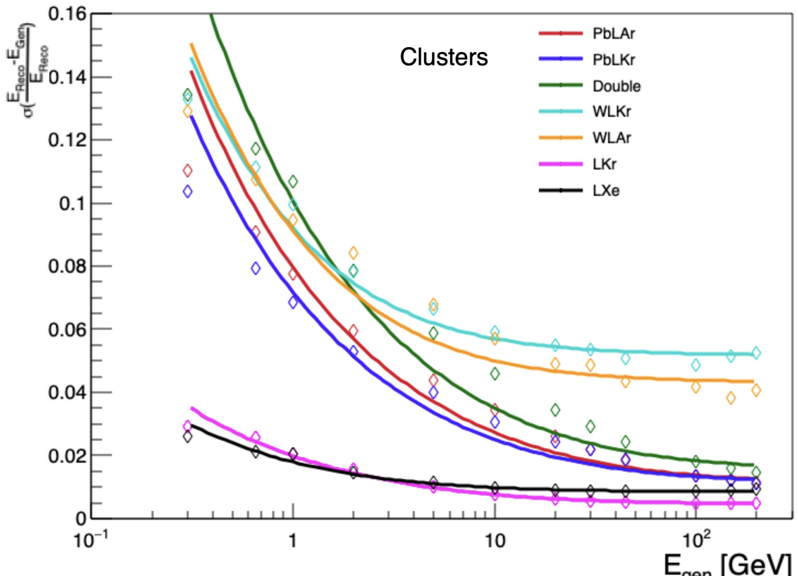
← Outputs



Performance results

- Example of performance results produced recently with FCC-ee LAr ECAL

Energy resolution for different absorber and Noble Liquid material

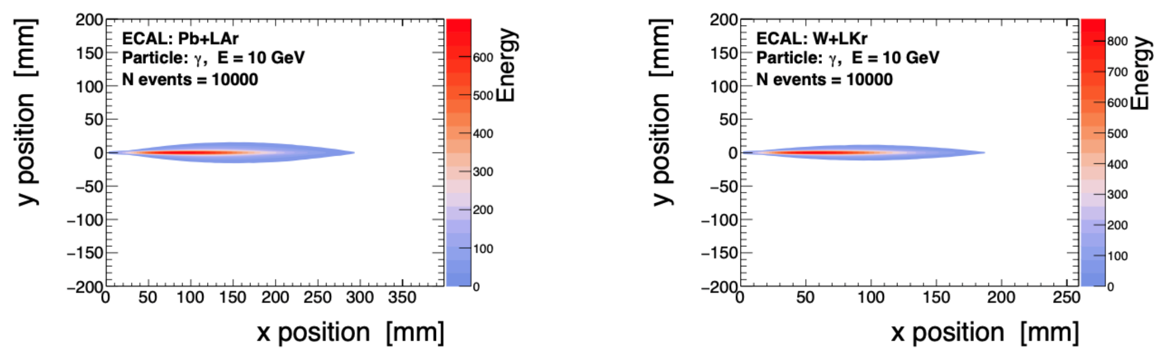


Mark Waterlaat

τ final state categorization confusion matrix

Recon → / Gen ↓	$\pi^{\pm}\,\nu$	$\pi^{\pm}\,\pi^{0}\,\nu$	$\pi^{\pm}\,2\pi^{0}\nu$	$\pi^{\pm}\,3\pi^{0}\nu$	$\pi^{\pm}\,4\pi^{0}\nu$
$\pi^{\pm}\,\nu$	**0.9560**	0.0425	0.0010	0.0003	0.0002
$\pi^{\pm}\,\pi^{0}\,\nu$	0.0374	**0.9020**	0.0586	0.0016	0.0002
$\pi^{\pm}\,2\pi^{0}\nu$	0.0090	0.1277	**0.7802**	0.0808	0.0022
$\pi^{\pm}\,3\pi^{0}\nu$	0.0036	0.0372	0.2679	**0.5972**	0.0910

Moliere Radius comparison between Pb + LAr and W + LKr



$R_M = 41$ mm $\qquad R_M = 27$ mm

Katinka Wandall-Christensen and Mogens Dam

- Stay tuned, more to come!
  - Not so far from being able to do a first Full Sim physics analysis (e.g. Axion → γγ once we have the ECAL endcap)