Migration of dual-readout simulation to key4hep

Sanghyun Ko

21 Jul 2021

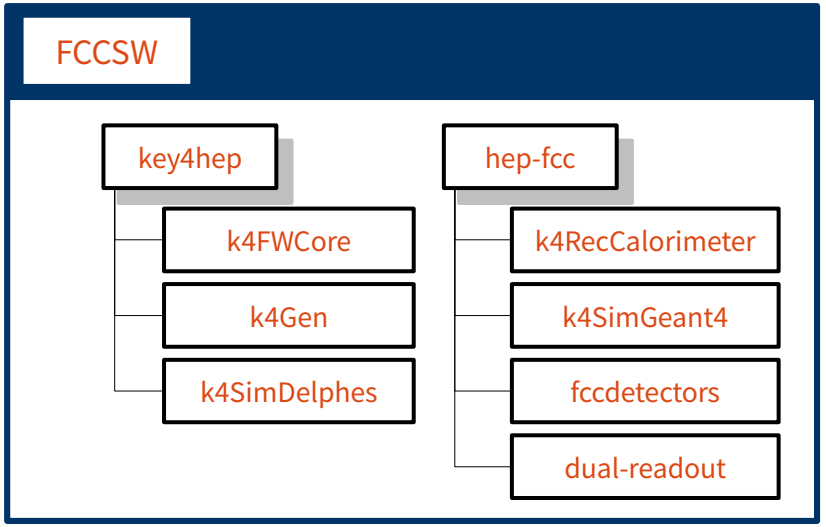
Introduction to Key4hep

Brief history of Key4hep

- Common software framework for all future HEP experiments proposed at the Future Collider Software Workshop at Bologna June 2019 [link], followed by mini-workshop at Hong Kong Jan 2020 [link]
- Consensus among future collider communities including ILC, CLIC, CEPC and FCC
- Encompass typical needs of HEP experiments, provide common turnkey stack covering different domains

Key4hep as compositions of FCCSW

- Core framework of FCCSW has been transferred to key4hep
  - FWCore → k4FWCore
  - Generation → k4Gen
  - Sim → k4SimGeant4
  - …
- Now FCCSW repository holds scripts and config files



Introduction to Key4hep

Brief history of Key4hep

- Common software framework for all future HEP experiments proposed at the Future Collider Software Workshop at Bologna June 2019 [link], followed by mini-workshop at Hong Kong Jan 2020 [link]
- Consensus among future collider communities including ILC, CLIC, CEPC and FCC
- Encompass typical needs of HEP experiments, provide common turnkey stack covering different domains

Key4hep

- Core fra
  - FW
  - Ge
  - Sim
  - …
- Now FC

FCCSW [link]

Common software for all FCC experiments. FCCSW is a collection of components intended to be run in the Gaudi framework. Because of the modular nature of the software, this repository contains little actual code, which can instead be found in the following places:

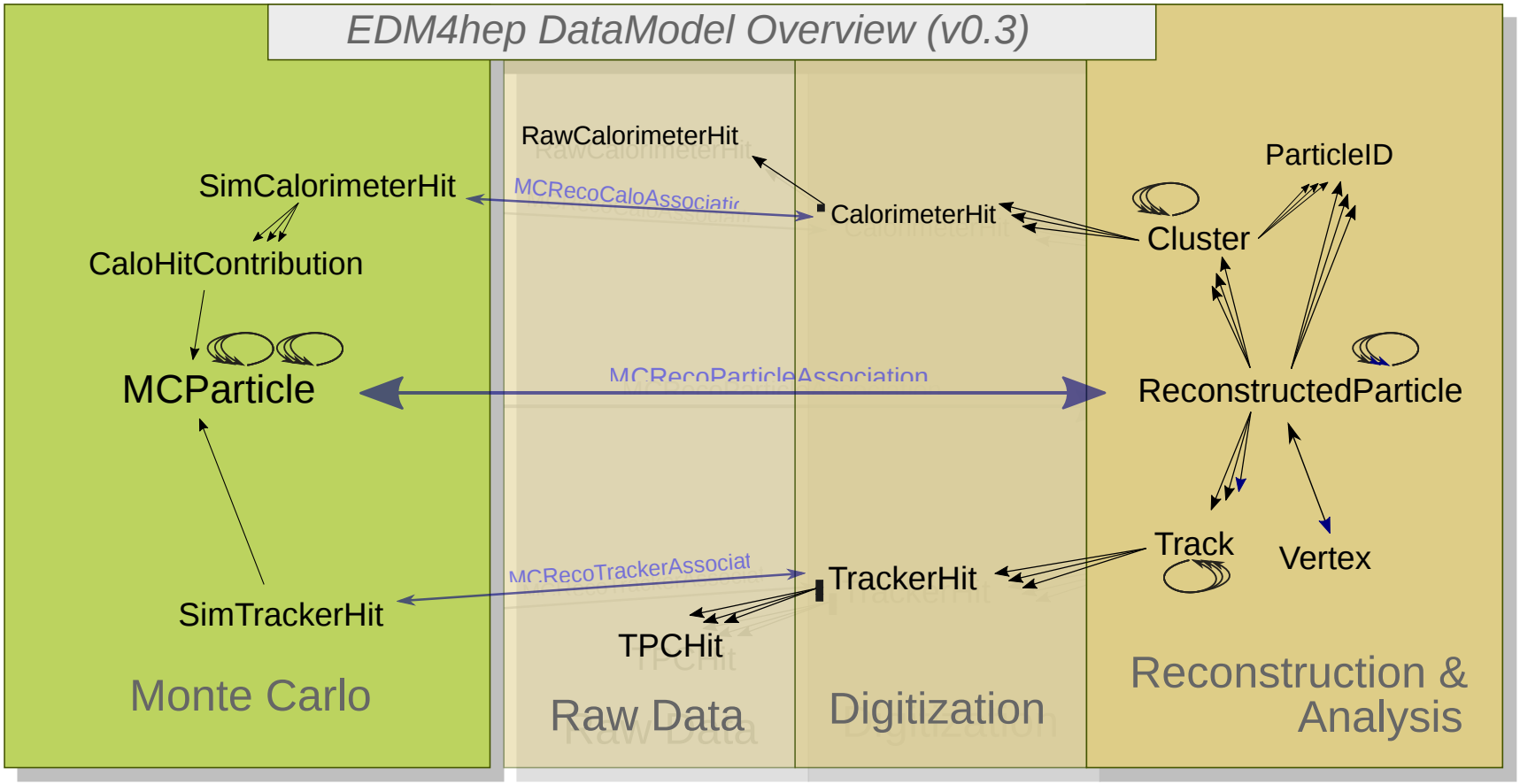
- `key4hep/k4FWCore` : Basic I/O components
- `key4hep/k4Gen` : Generators and Particle Guns
- `key4hep/k4SimDelphes` : Delphes Fast Sim
- `key4hep/k4SimGeant4` : Geant4 Full Sim
- `hep-fcc/fccdetectors` : DD4hep models of FCC detector geometries for Full Sim
- `hep-fcc/k4RecCalorimeter` : Calorimeter Reconstruction Code
- `hep-fcc/dual-readout` : DD4hep model of the DREAM dual readout calorimeter

FCCSW is a stakeholder of the Key4hep software collaboration.

EDM4hep & Podio

Common event data model for key4hep stakeholders

- EDM4hep [github] is the common EDM which can be shared by multiple future collider communities
- Support various use-cases motivated from different experiments
- Data model based on the Podio [github], a code-generator which supports the creation and handling data models



EDM4hep & Podio

Podio – Generator for EDM4hep

- Generate thread-safe code starting from a high-level description written in yaml format
- Produces c++ code at CMake step via Python
- Additional user-class ‘SparseVector’ was linked to EDM4hep for handling waveforms at the moment

```yaml
#------------ SimCalorimeterHit
edm4hep::SimCalorimeterHit:
  Description: "Simulated calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID       //ID of the sensor that created this hit
    - float energy                    //energy of the hit in [GeV].
    - edm4hep::Vector3f position      //position of the hit in world coordinates.
  OneToManyRelations:
    - edm4hep::CaloHitContribution contributions  //Monte Carlo step contribution
```

```cpp
class SimCalorimeterHit {

  friend SimCalorimeterHitCollection;
  friend SimCalorimeterHitCollectionIterator;
  friend ConstSimCalorimeterHit;

public:

  /// default constructor
  SimCalorimeterHit();
  SimCalorimeterHit(unsigned long long cellID, float energy, edm4hep:

  /// constructor from existing SimCalorimeterHitObj
  SimCalorimeterHit(SimCalorimeterHitObj* obj);

  /// copy constructor
  SimCalorimeterHit(const SimCalorimeterHit& other);

  /// copy-assignment operator
  SimCalorimeterHit& operator=(const SimCalorimeterHit& other);
public:

  /// Access the ID of the sensor that created this hit
  const unsigned long long& getCellID() const;

  /// Access the energy of the hit in [GeV].
  const float& getEnergy() const;

  /// Access the position of the hit in world coordinates.
  const edm4hep::Vector3f& getPosition() const;



  /// Set the ID of the sensor that created this hit
  void setCellID(unsigned long long value);

  /// Set the energy of the hit in [GeV].
  void setEnergy(float value);

  /// Set the position of the hit in world coordinates.
  void setPosition(edm4hep::Vector3f value);
  /// Get reference to position of the hit in world coordinates.
  edm4hep::Vector3f& position();
```

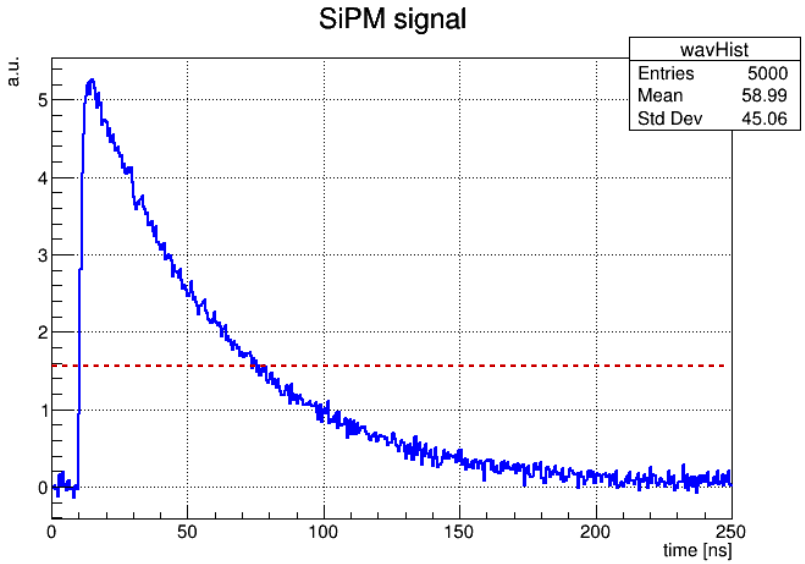
Saving waveforms

User-class edm4hep::SparseVector

- EDM4hep does not have designated data model to save waveforms
- Expect O(100ps) sampling time, while the length of integration gate can be up to O(100ns)
- Majority of bins can be empty after applying DCR threshold

  → use ‘sparse’ vector instead of a plain vector to suppress storage
- User-class is linked to EDM4hep

  → able to use EDM4hep classes and user-class simultaneously under Podio framework




```
edm4hep::SparseVector:
  Description: "User class to store sparse vector"
  Author : "Sanghyun Ko"
  Members:
    - float sampling // size of a bin
    - edm4hep::ObjectID assocObj // associated object ID
  VectorMembers:
    - float centers  // center value of the bin
    - float contents // content of the vector within [ cen
```

FYI: on-going discussion on user data in EDM4hep #117

```
add_library(edm4dr SHARED ${sources} ${headers})
target_link_libraries(edm4dr
  PUBLIC
  EDM4HEP::edm4hep
  podio::podio
)
target_include_directories(edm4dr
  PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>
  $<INSTALL_INTERFACE:${CMAKE_INSTALL_INCLUDEDIR}>
)
set_target_properties(edm4dr PROPERTIES PUBLIC_HEADER "${headers}")
```

Implementation w/ EDM4hep

```
#------------ SimCalorimeterHit
edm4hep::SimCalorimeterHit:
  Description: "Simulated calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID       //ID of the sensor that created this hit
    - float energy                    //energy of the hit in [GeV].
    - edm4hep::Vector3f position      //position of the hit in world coordinates.
  OneToManyRelations:
    - edm4hep::CaloHitContribution contributions  //Monte Carlo step contribution - parallel to particle


#------------  RawCalorimeterHit
edm4hep::RawCalorimeterHit:
  Description: "Raw calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID   //detector specific (geometrical) cell id.
    - int amplitude               //amplitude of the hit in ADC counts.
    - int timeStamp               //time stamp for the hit.

#------------  CalorimeterHit
edm4hep::CalorimeterHit:
  Description: "Calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID  //detector specific (geometrical) cell id.
    - float energy               //energy of the hit in [GeV].
    - float energyError          //error of the hit energy in [GeV].
    - float time                 //time of the hit in [ns].
    - edm4hep::Vector3f position //position of the hit in world coordinates.
    - int type                   //type of hit. Mapping of integer types to names via collection parameters
```

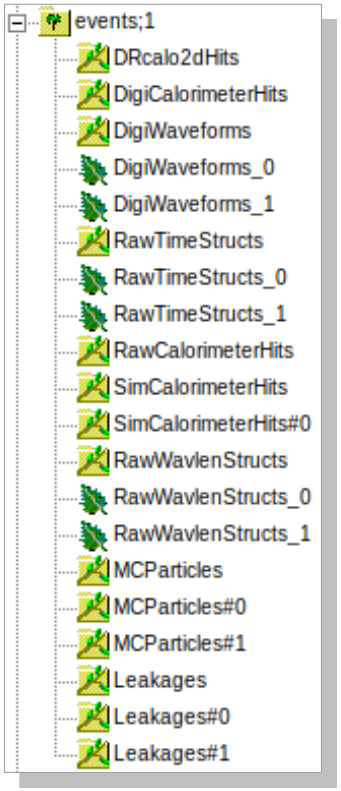
MC truth Edeps “SimCalorimeterHits”

GEANT4 SiPM hits “RawCalorimeterHits”

Digitized SiPM hits “DigiCalorimeterHits”

Calibrated Calorimeter hits “DRcalo2dHits”

See example at [analysis.cpp]



Implementation w/ EDM4hep

```
#------------ SimCalorimeterHit
edm4hep::SimCalorimeterHit:
  Description: "Simulated calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID       //ID of the sensor that created this hit
    - float energy                    //energy of the hit in [GeV].
    - edm4hep::Vector3f position      //position of the hit in world coordinates.
  OneToManyRelations:
    - edm4hep::CaloHitContribution contributions  //Monte Carlo step contribution - parallel to particle


#------------  RawCalorimeterHit
edm4hep::RawCalorimeterHit:
  Description: "Raw calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID   //detector specific (geometrical) cell id.
    - int amplitude               //amplitude of the hit in ADC counts.
    - int timeStamp               //time stamp for the hit.

#------------  CalorimeterHit
edm4hep::CalorimeterHit:
  Description: "Calorimeter hit
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID
    - float energy
    - float energyError
    - float time
    - edm4hep::Vector3f positio
    - int type
```

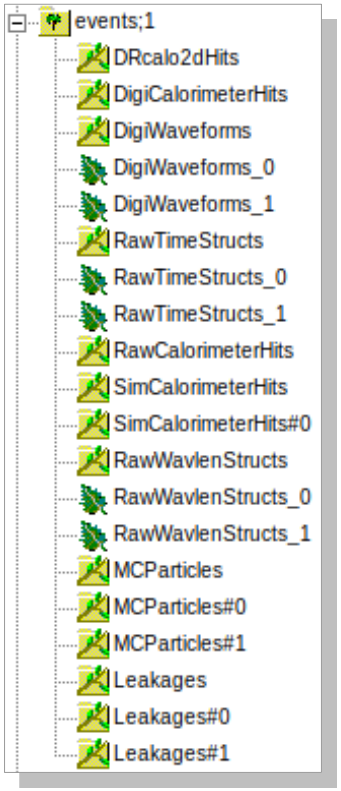
MC truth Edeps “SimCalorimeterHits”

GEANT4 SiPM hits “RawCalorimeterHits”

Digitized SiPM hits “DigiCalorimeterHits”

Calibrated Calorimeter hits “DRcalo2dHits”

See example at [analysis.cpp]




```
datatypes :
  edm4hep::SparseVector:
    Description: "User class to store sparse vector"
    Author : "Sanghyun Ko"
    Members:
      - float sampling // size of a bin
      - edm4hep::ObjectID assocObj // associated object ID
    VectorMembers:
      - float centers  // center value of the bin
      - float contents // content of the vector within [ center-sampling/2., center+sampling/2. )
```

User class to store timing (waveforms)

~ O(Mb) / evt

dominated by waveforms, can be dropped by ’drop DigiWaveforms’

History of DRC framework

Development history of DRC simulation

- Started from standalone GEANT4 simulation of 4th concept geometry i.e. WGR16
- Essential refactoring & implementing ROOT datamodel

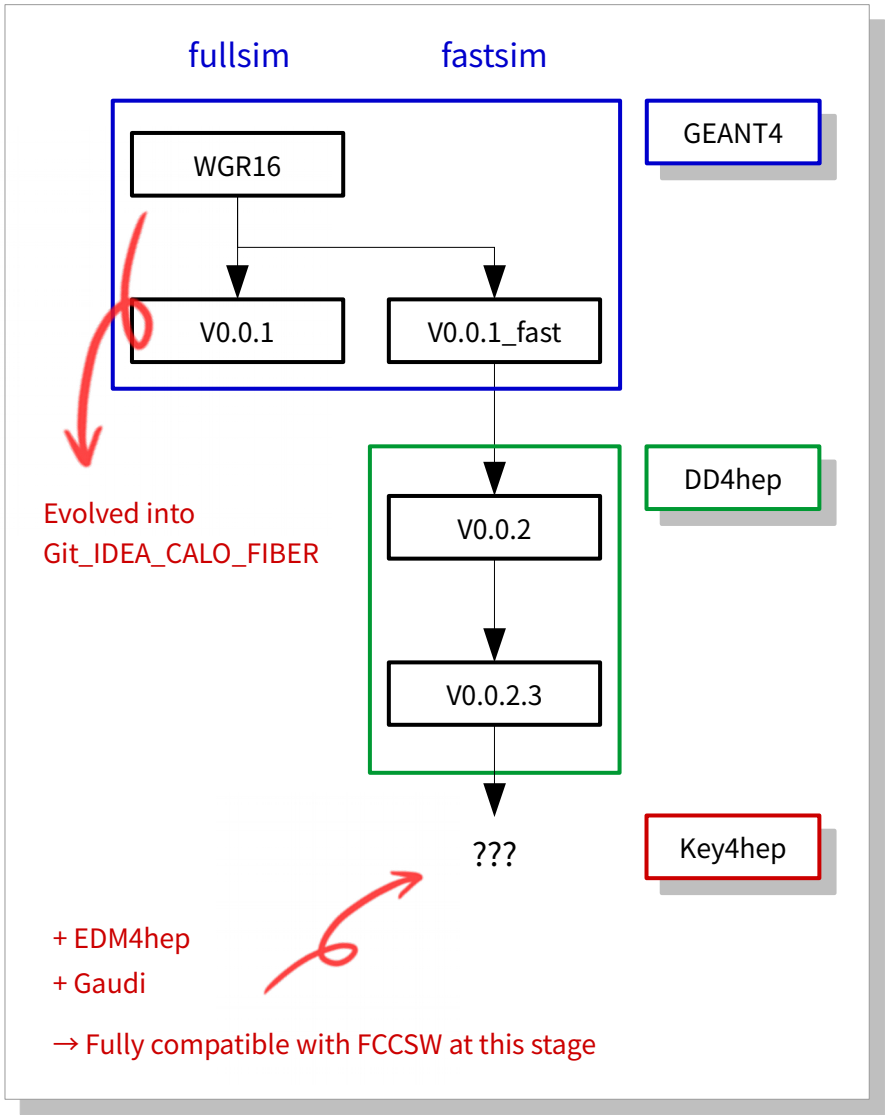
  → v0.0.1 (full simulation)
- Development of G4VFastSimulationModel for optical fibers

  → v0.0.1_fast (fast simulation) – GEANT4 R&D meeting [link]
- Migration to DD4hep

  → v0.0.2 – FCCSW meeting [link]
- ⋯ leads to current version (v0.0.2.3) after several minor fixes

  e.g. support IDEA geometry, calibration constants, …

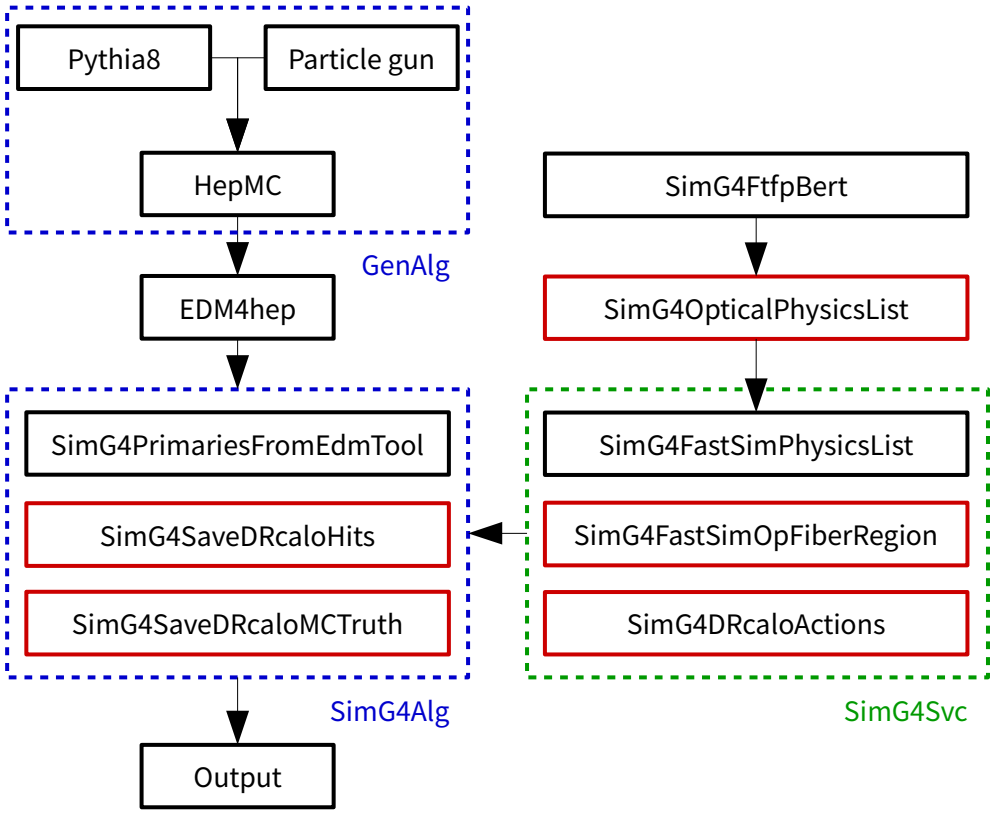
- Last remaining components for migration were EDM4hep (data model) and Gaudi (framework core)



Simulation flow

Typical simulation workflow (runDRsim.py)

- A Gaudi workflow consists of 2 types of components:
  - Service & Algorithm
- Gaudi Service takes responsibility of handling setup of the run, e.g.
  - Geometry
  - G4RunManager
- Gaudi Algorithm handles flow of the event, e.g.
  - Generating & saving events
  - Triggering & event selection
- Baseline framework of GEN & SIM step is based on k4Gen [link] and k4SimGeant4 [link], respectively.




```
ApplicationMgr(
  TopAlg = [gen, hepmc2edm, geantsim, podiooutput],
  EvtSel = 'NONE',
  EvtMax = 10,
  # order is important, as GeoSvc is needed by SimG4Svc
  ExtSvc = [dataservice, geoservice, geantservice]
)
```

Simulation flow

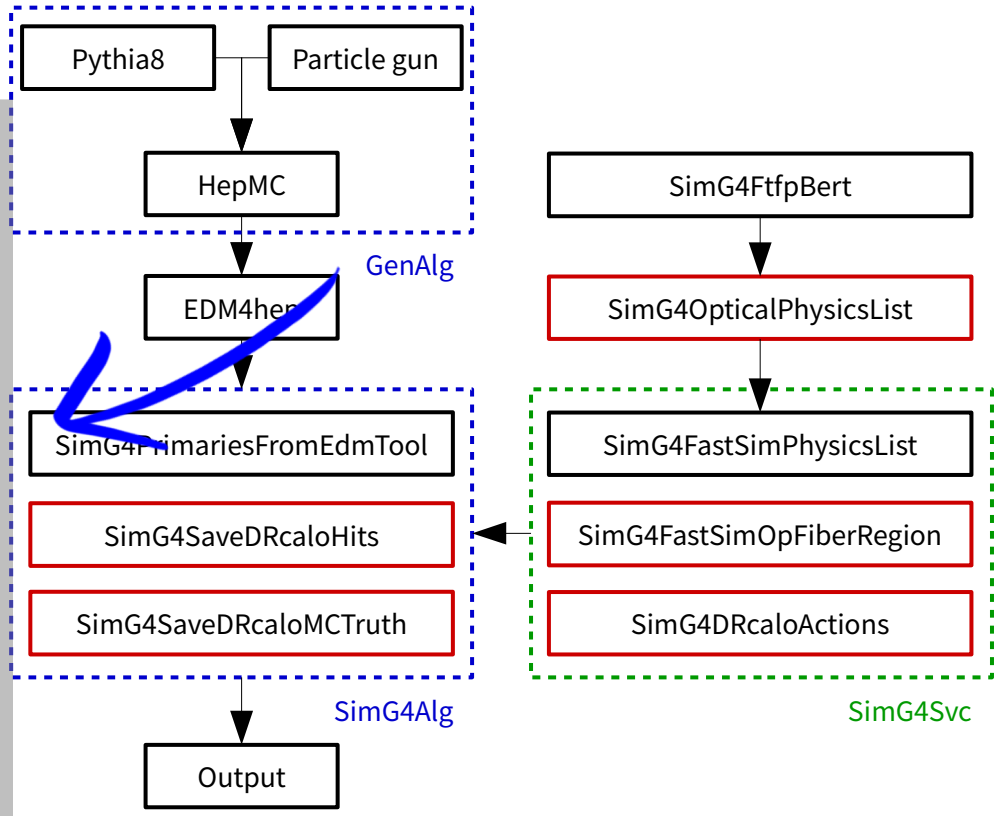
Typical simulation workflow (runDRsim.py)

```
from Configurables import GenAlg, MomentumRangeParticleGun
pgun = MomentumRangeParticleGun("PGun",
  PdgCodes=[11], # electron
  MomentumMin = 20.*units.GeV, # GeV
  MomentumMax = 20.*units.GeV, # GeV
  ThetaMin = 1.5335, # rad
  ThetaMax = 1.5335, # rad
  PhiMin = 0.01745, # rad
  PhiMax = 0.01745 # rad
)

from Configurables import FlatSmearVertex
smearTool = FlatSmearVertex("VertexSmearingTool",
  yVertexMin = -36.42, # mm
  yVertexMax = -26.42, # mm
  zVertexMin = -52.135, # mm
  zVertexMax = -42.135, # mm
  beamDirection = 0 # 1, 0, -1
)

from Configurables import HepMCToEDMConverter
hepmc2edm = HepMCToEDMConverter("Converter")

gen = GenAlg("ParticleGun", SignalProvider=pgun, VertexSmearingTool=sme
```




```
ApplicationMgr(
  TopAlg = [gen, hepmc2edm, geantsim, podioutput],
  EvtSel = 'NONE',
  EvtMax = 10,
  # order is important, as GeoSvc is needed by SimG4Svc
  ExtSvc = [dataservice, geoservice, geantservice]
)
```

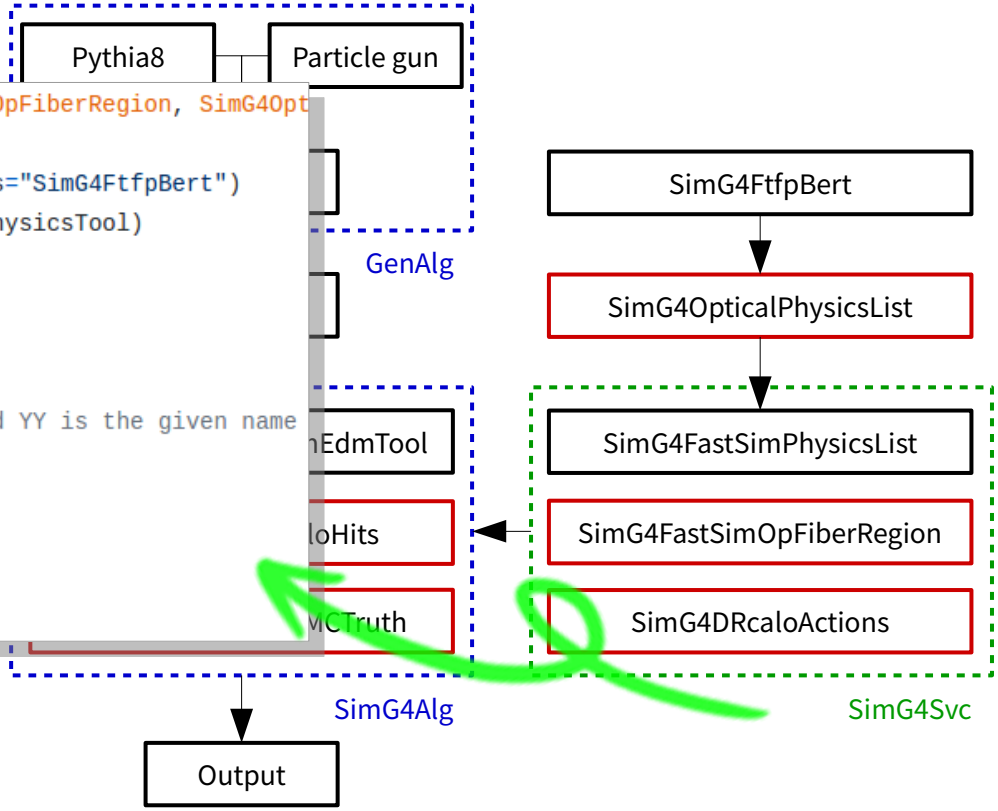
Simulation flow

Typical simulation workflow (runDRsim.py)

```
from Configurables import SimG4Svc, SimG4FastSimPhysicsList, SimG4FastSimOpFiberRegion, SimG4Opt
regionTool = SimG4FastSimOpFiberRegion("fastfiber")
opticalPhysicsTool = SimG4OpticalPhysicsList("opticalPhysics", fullphysics="SimG4FtfpBert")
physicslistTool = SimG4FastSimPhysicsList("Physics", fullphysics=opticalPhysicsTool)

from Configurables import SimG4DRcaloActions
actionTool = SimG4DRcaloActions("SimG4DRcaloActions")

# Name of the tool in GAUDI is "XX/YY" where XX is the tool class name and YY is the given name
geantservice = SimG4Svc("SimG4Svc",
  physicslist = physicslistTool,
  regions = ["SimG4FastSimOpFiberRegion/fastfiber"],
  actions = actionTool
)
```



- Gaudi Algorithm handles flow of the event, e.g.
  - Generating & saving events
  - Triggering & event selection

- Baseline framework of GEN & SIM step is based on k4Gen [link] and k4SimGeant4 [link], respectively.

```
ApplicationMgr(
  TopAlg = [gen, hepmc2edm, geantsim, podiooutput],
  EvtSel = 'NONE',
  EvtMax = 10,
  # order is important, as GeoSvc is needed by SimG4Svc
  ExtSvc = [dataservice, geoservice, geantservice]
)
```

Simulation flow

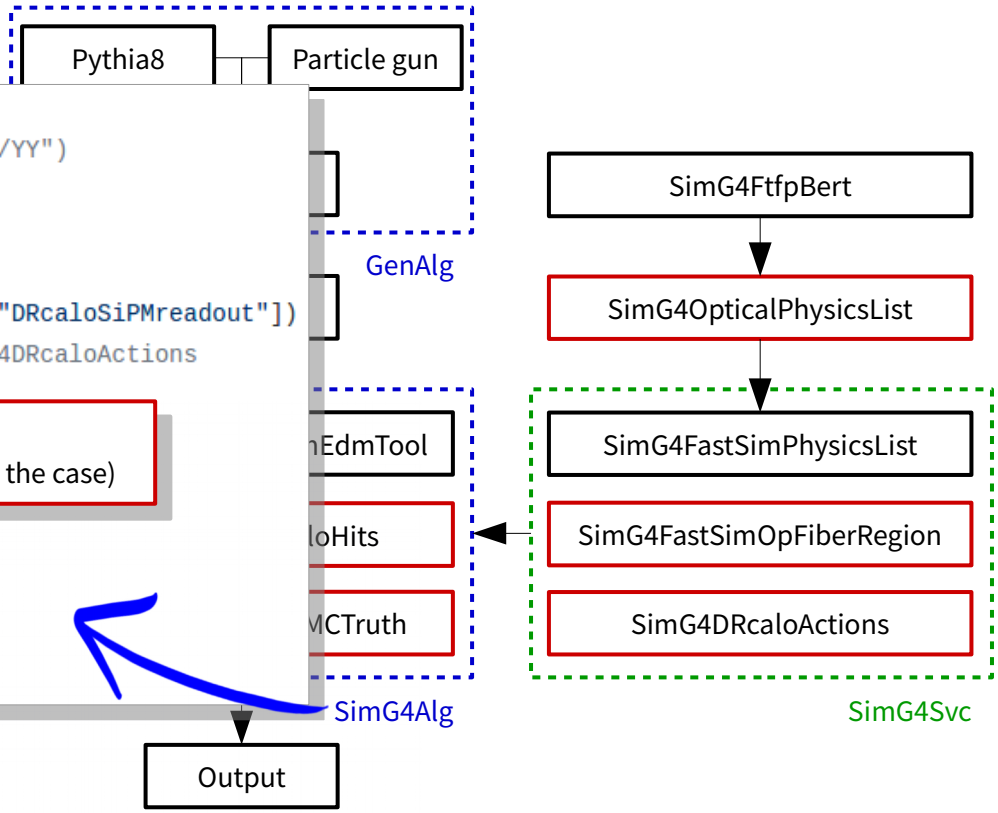
Typical simulation workflow (runDRsim.py)

```
from Configurables import SimG4Alg, SimG4PrimariesFromEdmTool
# next, create the G4 algorithm, giving the list of names of tools ("XX/YY")
edmConverter = SimG4PrimariesFromEdmTool("EdmConverter")

from Configurables import SimG4SaveDRcaloHits, SimG4SaveDRcaloMCTruth
saveDRcaloTool = SimG4SaveDRcaloHits("saveDRcaloTool", readoutNames = ["DRcaloSiPMreadout"])
saveMCTruthTool = SimG4SaveDRcaloMCTruth("saveMCTruthTool") # need SimG4DRcaloActions

geantsim = SimG4Alg("SimG4Alg",
  outputs = [
    "SimG4SaveDRcaloHits/saveDRcaloTool",
    "SimG4SaveDRcaloMCTruth/saveMCTruthTool"
  ],
  eventProvider = edmConverter
)
```

Holds G4UserActions
(UserSteppingAction in the case)



- Generating & saving events
- Triggering & event selection

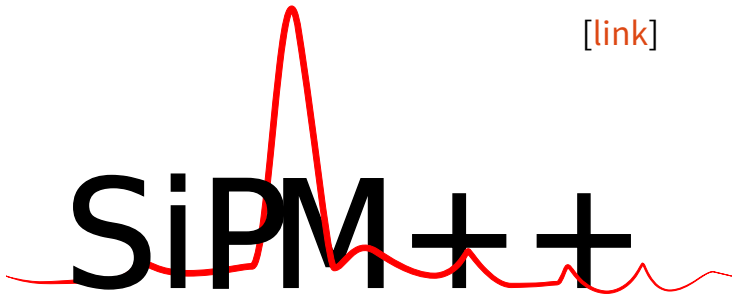
```
ApplicationMgr(
  TopAlg = [gen, hepmc2edm, geantsim, podiooutput],
  EvtSel = 'NONE',
  EvtMax = 10,
  # order is important, as GeoSvc is needed by SimG4Svc
  ExtSvc = [dataservice, geoservice, geantservice]
)
```

- Baseline framework of GEN & SIM step is based on k4Gen [link] and k4SimGeant4 [link], respectively.

Digitization

Digitization configuration (runDigi.py)

[link]




```
from Gaudi.Configuration import *
from Configurables import ApplicationMgr

from Configurables import k4DataSvc
dataservice = k4DataSvc("EventDataSvc", input="sim.root")

from Configurables import PodioInput
podioinput = PodioInput("PodioInput", collections = ["RawTimeStructs", "RawCalorimeterHits", "SimCalorimeterHits", "RawWavlenStructs",

from Configurables import DigiSiPM
digi = DigiSiPM("DigiSiPM", OutputLevel=DEBUG)

from Configurables import PodioOutput
podiooutput = PodioOutput("PodioOutput", filename = "digi.root", OutputLevel = DEBUG)
podiooutput.outputCommands = ["keep *"]

ApplicationMgr(
    TopAlg = [
        podioinput,
        digi,
        podiooutput
    ],
    EvtSel = 'NONE',
    EvtMax = 10,
    ExtSvc = [dataservice]
)
```

Choose with collection to keep or drop

Define sequence

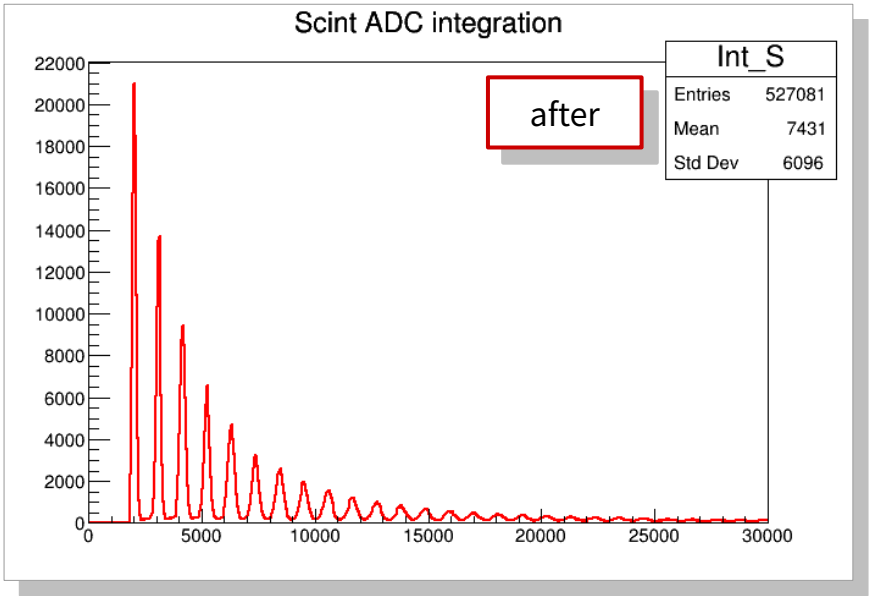
Configurable SiPM parameters

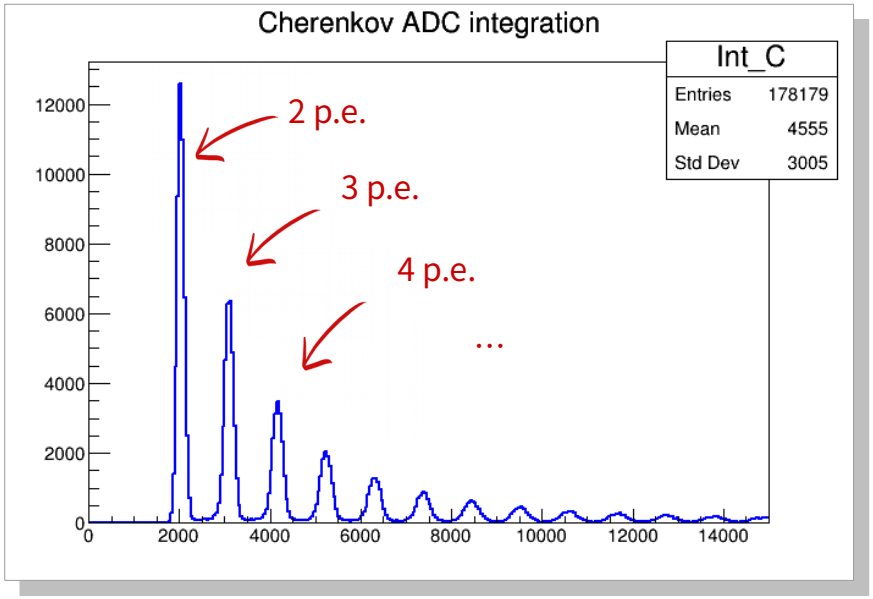
```
from Configurables import DigiSiPM
digi = DigiSiPM("DigiSiPM",
  # Hamamatsu S13615-1025
  signalLength = 500.,
  SiPMsize = 1.,
  DCR = 100e3,
  Xtalk = 0.03,
  sampling = 0.1,
  recovery = 20.,
  cellpitch = 25.,
  afterpulse = 0.03,
  falltimeFast = 50.,
  risetime = 1.,
  SNR = 30.,
  gateLength = 240.,
  OutputLevel = DEBUG
)
```

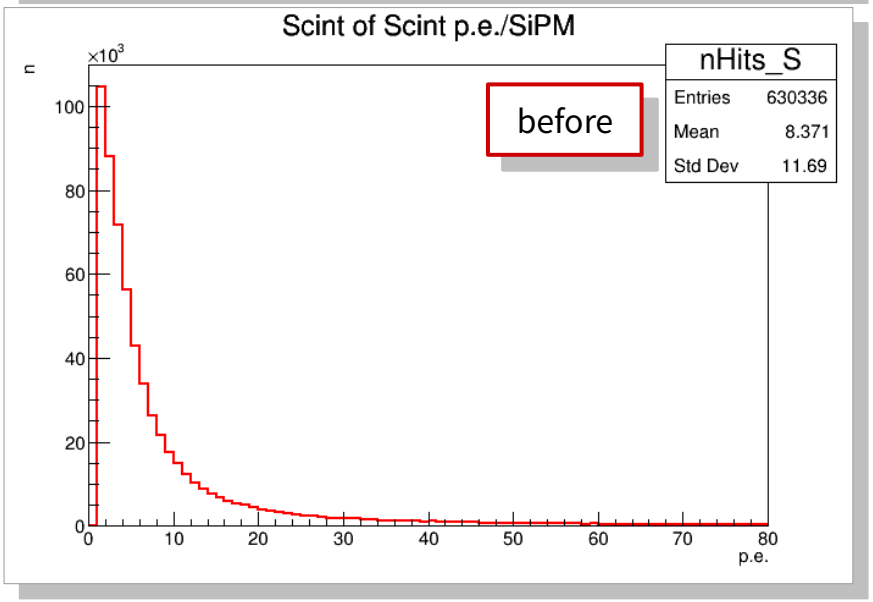
simply run with ‘k4run runDigi.py’

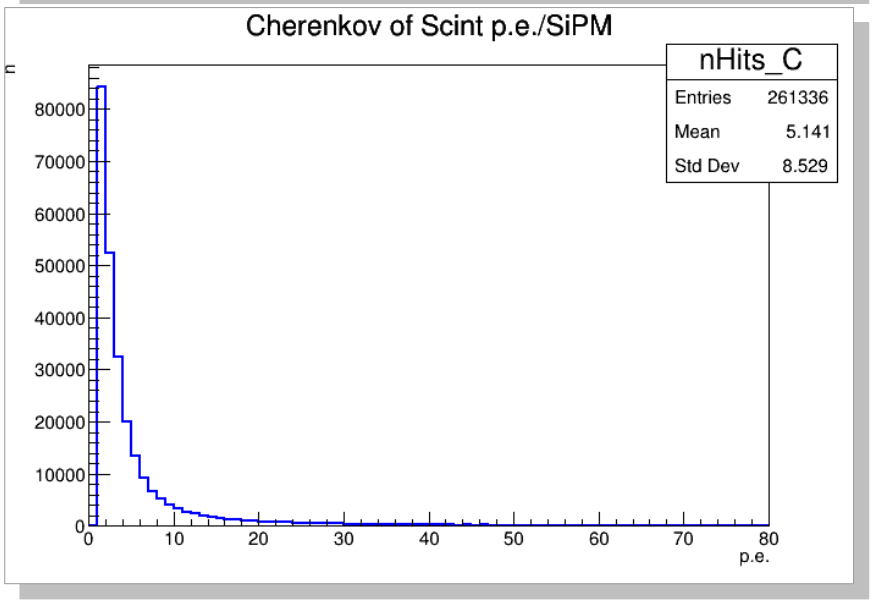
Integration

Signal integration (and # p.e. before digitization) for 20 GeV e-







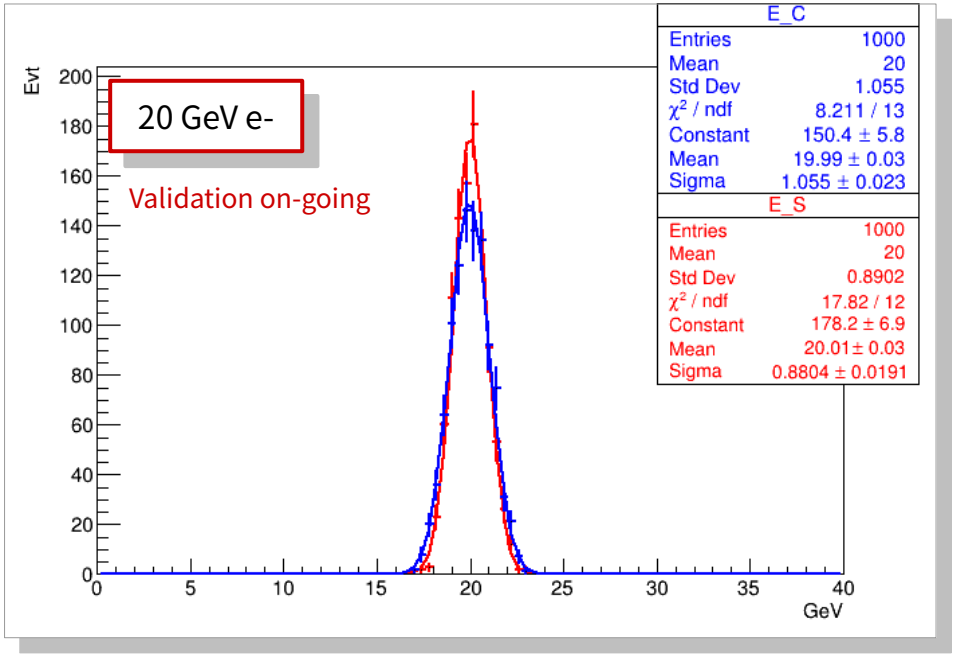


Reconstruction (calibration)

Reconstruction configuration (runDRcalib.py)

```
22  from Configurables import GeoSvc
23  geoservice = GeoSvc(
24      "GeoSvc",
25      detectors = [
26          'file:share/compact/DRcalo.xml'
27      ]
28  )
29
30  from Configurables import DRcalib2D
31  calib2d = DRcalib2D("DRcalib2D", OutputLevel=DEBUG)
32
33  from Configurables import PodioOutput
34  podiooutput = PodioOutput("PodioOutput", filename = "reco.root", OutputLevel = DEBUG)
35  podiooutput.outputCommands = ["keep *"]
36
37  ApplicationMgr(
38      TopAlg = [
39          podioinput,
40          calib2d,
41          podiooutput
42      ],
43      EvtSel = 'NONE',
44      EvtMax = 10,
45      ExtSvc = [dataservice, geoservice]
46  )
```

Append “GeoSvc” module when needs geometry



Applications & Remarks

Potential applications

- The framework is now fully migrated to key4hep (FCCSW) at this stage

  → possible to overlay any existing FCCSW algorithms to DRC

  - Applying magnetic field
  - Merging (grouping) calorimeter hits
  - …

```
G4TransportationManager* transpManager = G4TransportationManager::GetTransportationManager();
G4FieldManager* fieldManager = transpManager->GetFieldManager();
G4PropagatorInField* propagator = transpManager->GetPropagatorInField();

// The field manager keeps an observing pointer to the field, ownership stays with this tool. (Cleaned up in dtor)
m_field =
    new sim::ConstantField(m_fieldComponentX, m_fieldComponentY, m_fieldComponentZ, m_fieldRadMax, m_fieldZMax);
fieldManager->SetDetectorField(m_field);

G4ChordFinder* chordFinder =
new G4ChordFinder( m_field,  m_minStep, stepper(m_integratorStepper, m_field));
fieldManager->SetChordFinder(chordFinder);

propagator->SetLargestAcceptableStep(m_maxStep);

if (m_deltaChord > 0) fieldManager->GetChordFinder()->SetDeltaChord(m_deltaChord);
if (m_deltaOneStep > 0) fieldManager->SetDeltaOneStep(m_deltaOneStep);
if (m_minEps > 0) fieldManager->SetMinimumEpsilonStep(m_minEps);
if (m_maxEps > 0) fieldManager->SetMaximumEpsilonStep(m_maxEps);
```

Pre-exist in k4SimGeant4 [link]

Applications & Remarks

Potential applications

- The framework is now fully migrated to key4hep (FCCSW) at this stage

  → possible to overlay any existing FCCSW algorithms to DRC

  - Applying magnetic field
  - Merging (grouping) calorimeter hits
  - …

Remarks

- Validation on-going
- Minor difference of definitions in using edm4hep::SimCalorimeterHit & edm4hep::RawCalorimeterHit
- User-class edm4hep::SparseVector to save waveforms & timing structure
- Usage of SimSiPM as an external library → needs to be part of key4hep-spack to deploy the simulation on CVMFS

Backups

Backups

```
#------------- SimCalorimeterHit
edm4hep::SimCalorimeterHit:
  Description: "Simulated calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID       //ID of the sensor that created this hit
    - float energy                    //energy of the hit in [GeV].
    - edm4hep::Vector3f position      //position of the hit in world coordinates.
  OneToManyRelations:
    - edm4hep::CaloHitContribution contributions  //Monte Carlo step contribution - parallel to


#-------------  RawCalorimeterHit
edm4hep::RawCalorimeterHit:
  Description: "Raw calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID   //detector specific (geometrical) cell id.
    - int amplitude               //amplitude of the hit in ADC counts.
    - int timeStamp               //time stamp for the hit.
```

```
edm4hep::SparseVector:
  Description: "User class to store sparse vector"
  Author : "Sanghyun Ko"
  Members:
    - float sampling // size of a bin
    - edm4hep::ObjectID assocObj // associated object ID
  VectorMembers:
    - float centers  // center value of the bin
    - float contents // content of the vector within [ center-sampling/2., center+sampling/2. )
```

SiPM & ADC spec

Hamamatsu S13615-1025

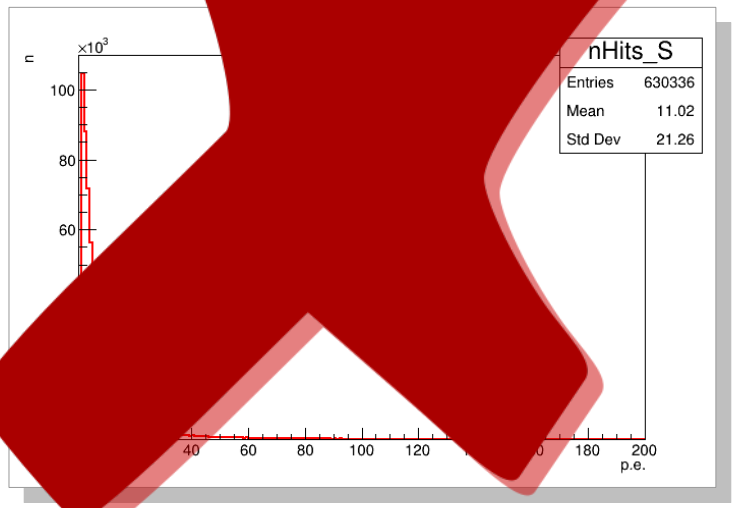
- PDE applied in GEANT4
- Referenced from INFN Insubria group [link]

	S13615-1025
Dark count rate	100 kHz
X-talk prob	0.03
Recovery time	20 ns
Cell pitch	25 μm
Afterpulse prob	0.03
Signal decay time	50 ns
Signal rise time	1 ns
Signal-to-Noise ratio	30 dB
Sampling rate	0.1 ns

ADC & integration parameters

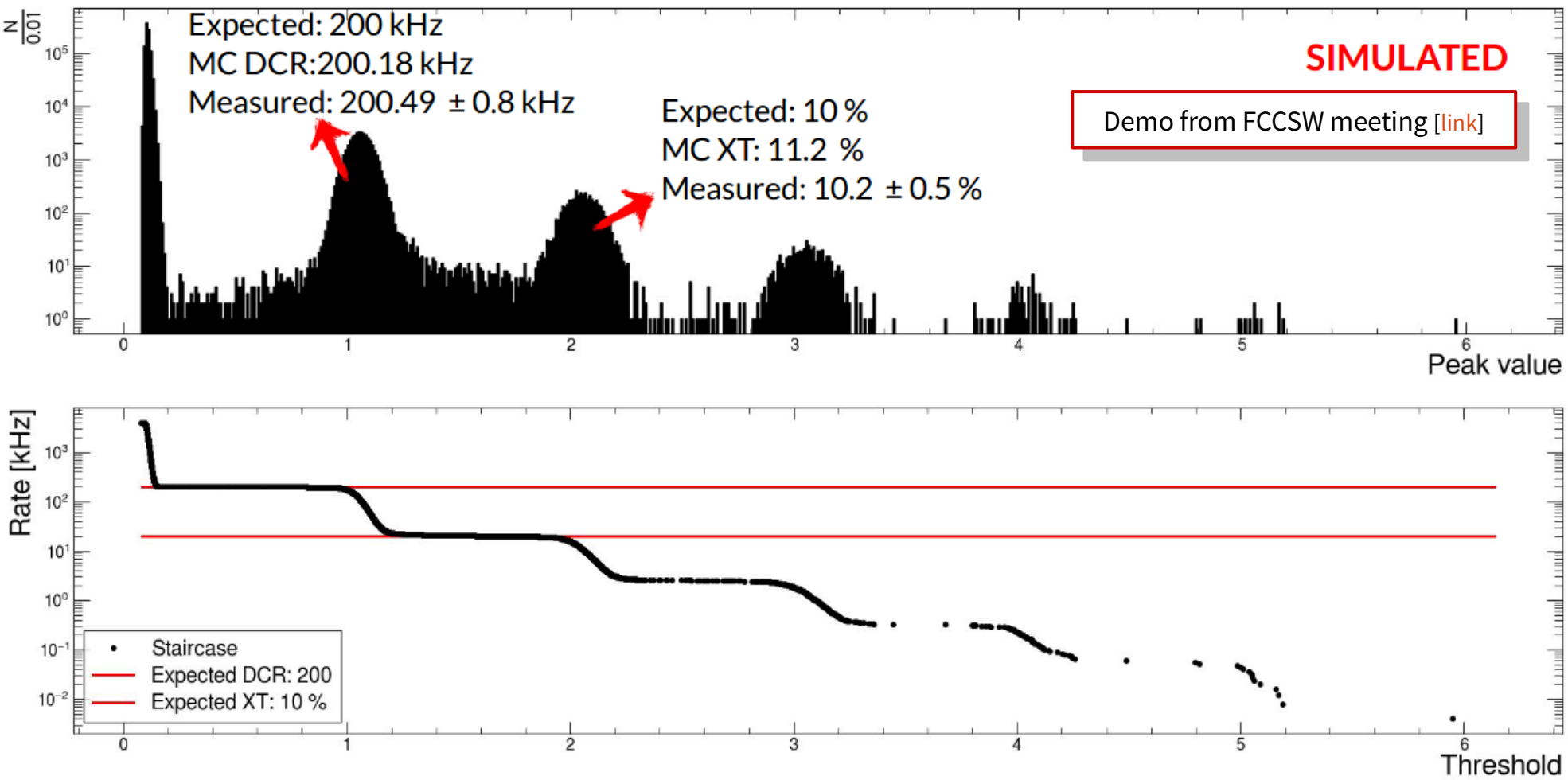
- Expected bits & gain estimated from the light yield
  - s.p.e./SiPM $\lesssim$ 100 for 20 GeV e-
  - Avoid s… at ~ 1000 s.p.e.
  - Need 10… for sig… .p.e.

→ Require cov…



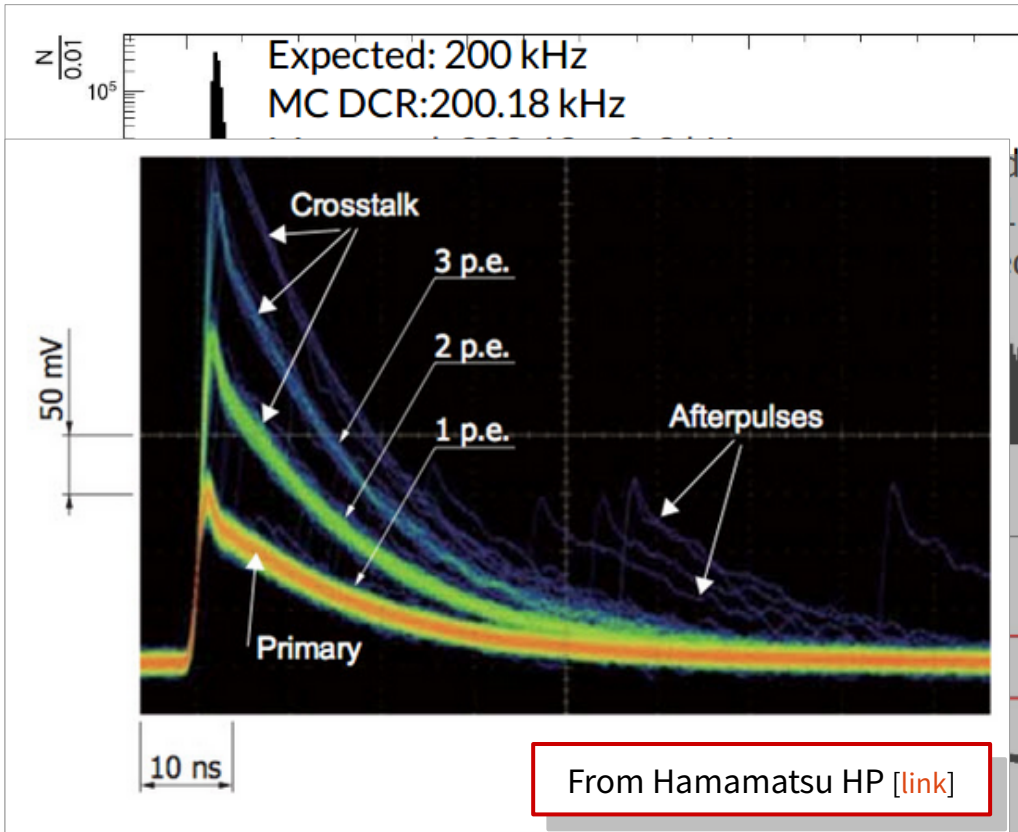
ADC param		Integration param	
Bits	16	Gate start	10 ns
Range	[-2^15, 2^15]	Gate length	240 ns
Gain	20 dB	threshold	30 (1.5 p.e.)

SiPM & ADC spec



Bits	16	Gate start	10 ns
Range	[-2^15, 2^15]	Gate length	240 ns
Gain	20 dB	threshold	30 (1.5 p.e.)

SiPM & ADC spec



From Hamamatsu HP [link]

Figure 12. An oscilloscope trace with the display set to persistence mode.

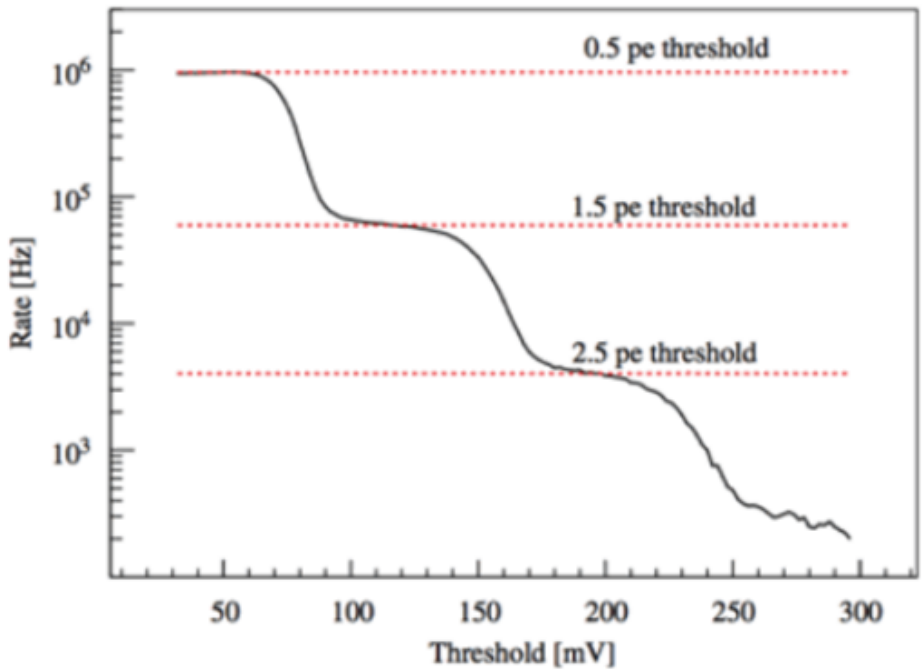


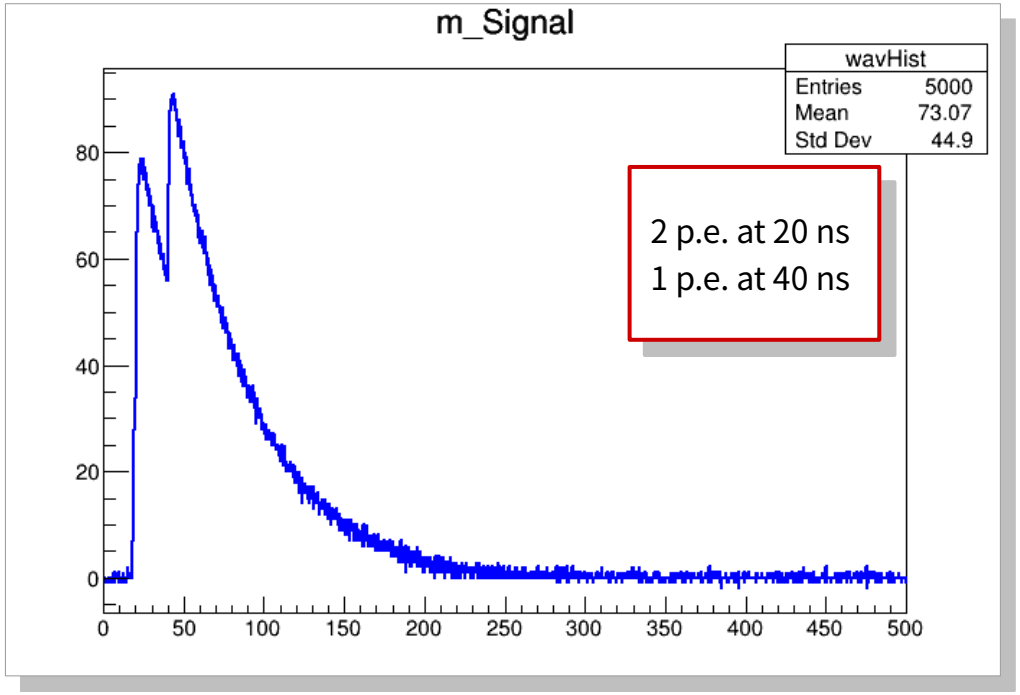
Figure 14. Plot of experimental data producing characteristic "staircase."

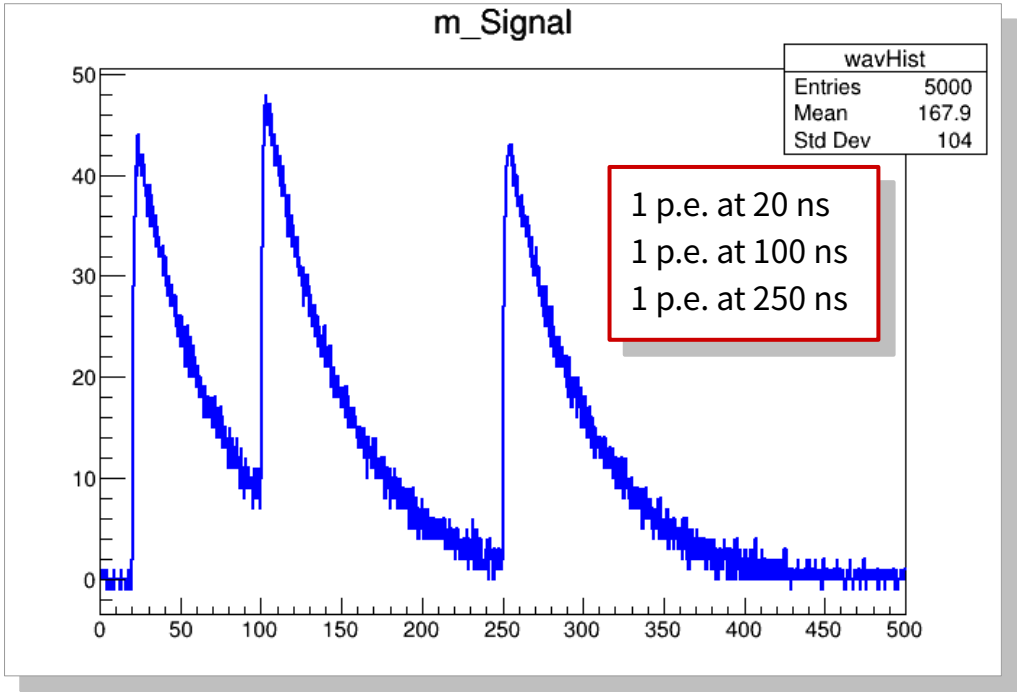
Bits	16	Gate start	10 ns
Range	[-2^15, 2^15]	Gate length	240 ns
Gain	20 dB	threshold	30 (1.5 p.e.)

Toy SiPM signal

SiPM signal demonstration with toy input

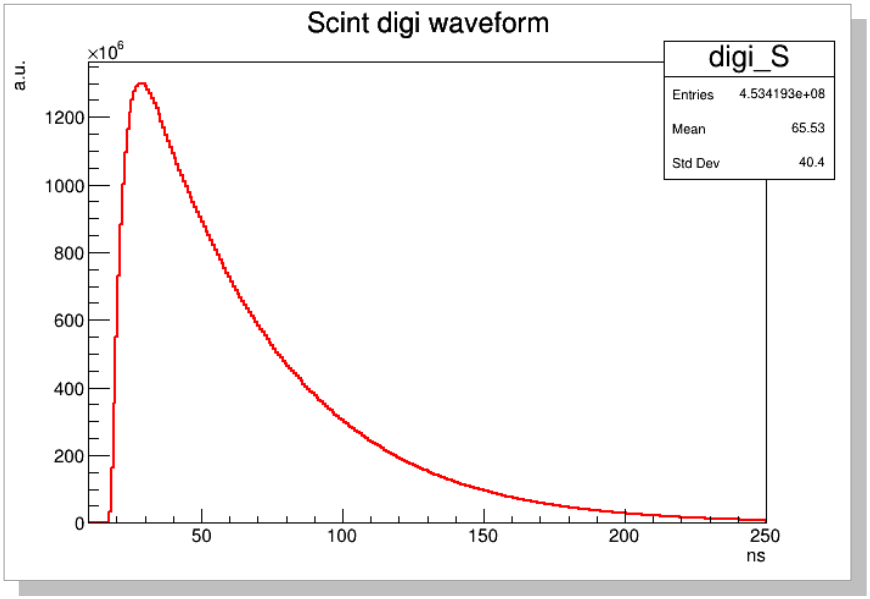
- Demonstrated SiPM signal with only 2 – 3 p.e. for visual purpose
  - Hamamatsu S13615-1025
  - 16 bits ADC with 20x gain

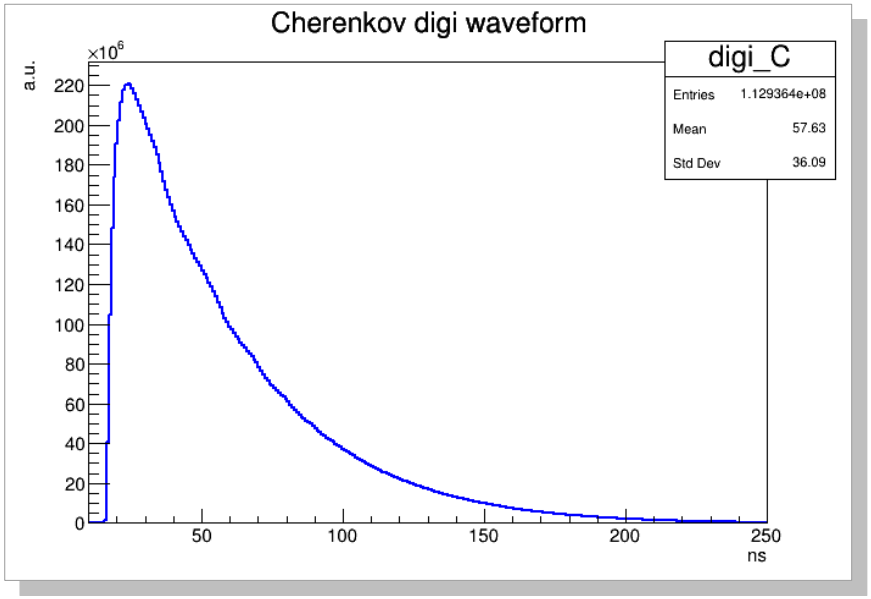


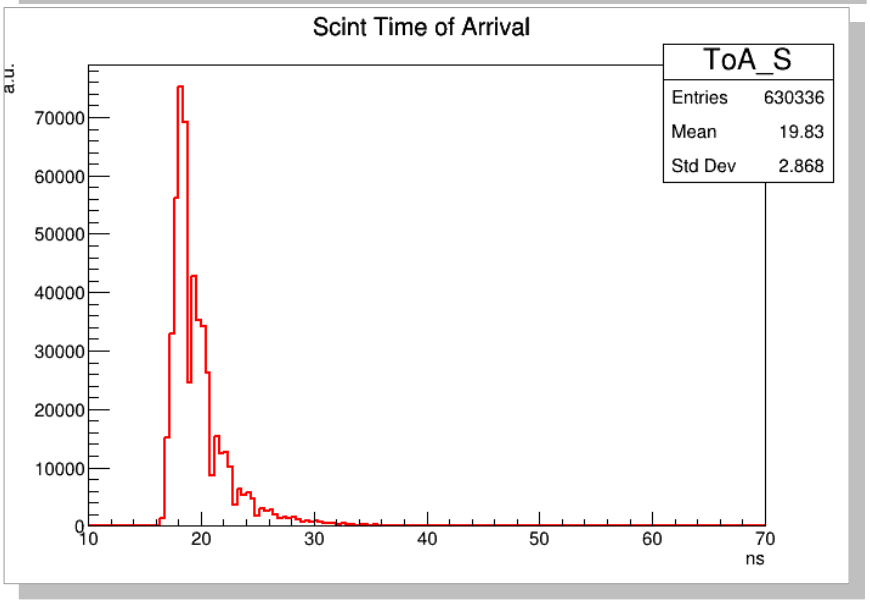


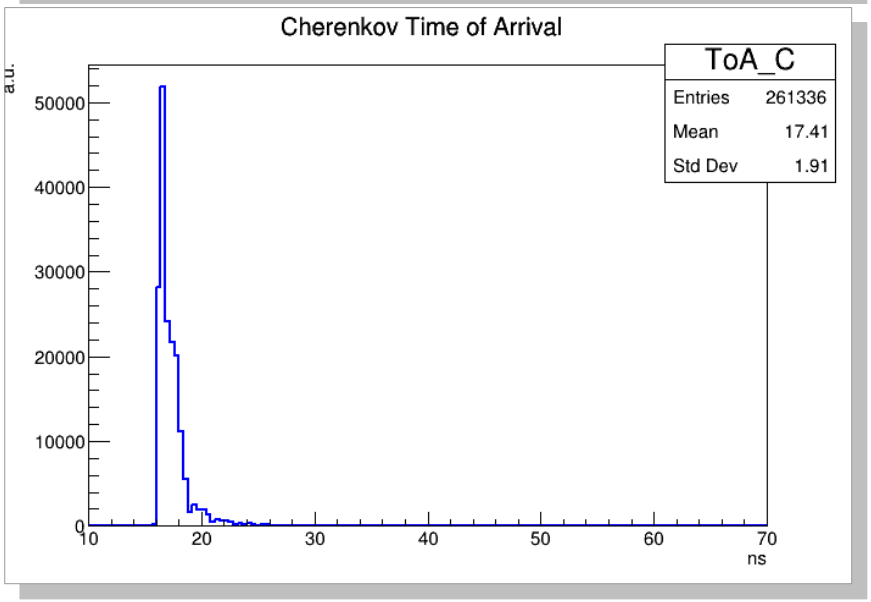
Waveforms & ToA

Digitized waveforms & Time of Arrival for 20 GeV e-









Speeding up optical photon tracking

Optical photons tracking in dual-readout calorimeter

- Full simulation of optical photon tracking explodes CPU cost: O(h)/evt

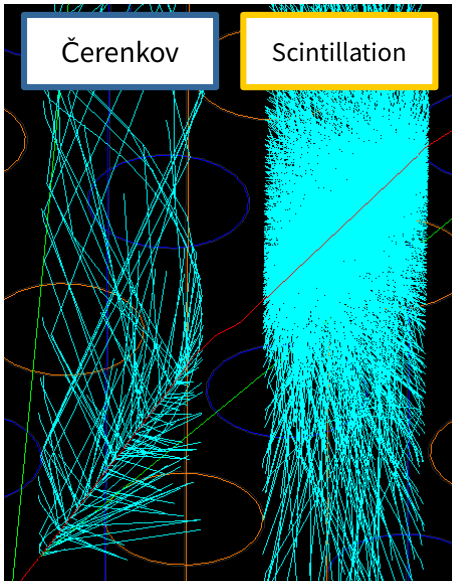
  $\underbrace{\text{O(10k) tracks/photon}}_{\rightarrow \text{Due to total internal reflections}} \times \underbrace{\text{O(10k) photons/GeV}}_{\rightarrow \text{Due to scintillation yield}} \simeq \text{O(100M) tracks/GeV}$

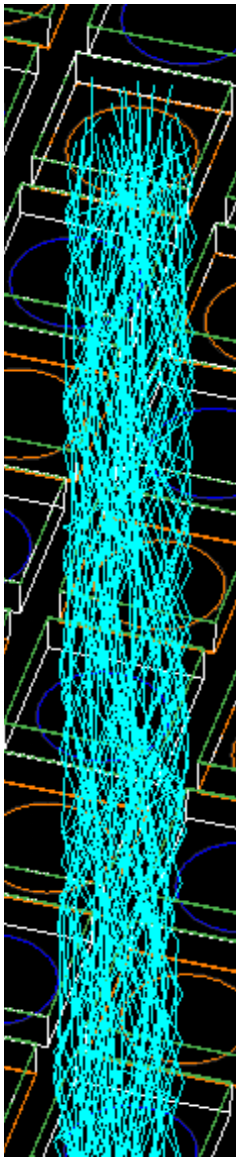
- Having reasonable CPU cost is important

  Conventional solution is killing optical photons after counting

  → not an optimal solution for dual-readout calorimeter

  - Numerical aperture is important for the yield of Čerenkov signal
  - Light attenuation of fibres & timing of optical photons



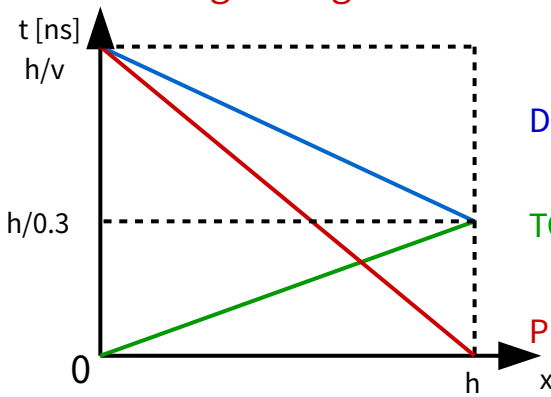


Important for a longitudinally unsegmented calorimeter

Measuring depth with dual-readout calorimeter

- Depth x can be represented as a function of detection time

  → simulating timing structure of optical photons is essential



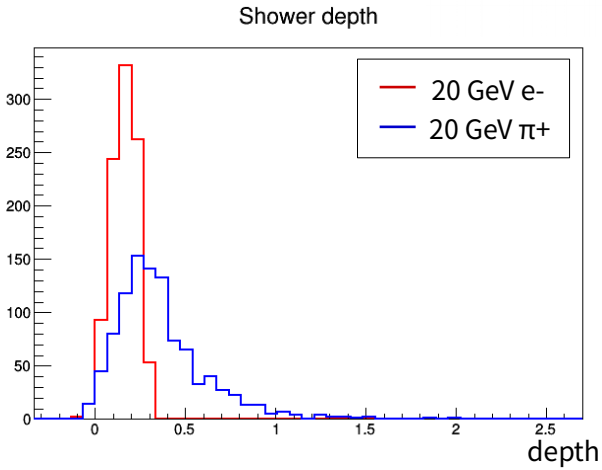
Detection time

II

TOF of π+ in vacuum/tower

+

Propagation time of optical photons

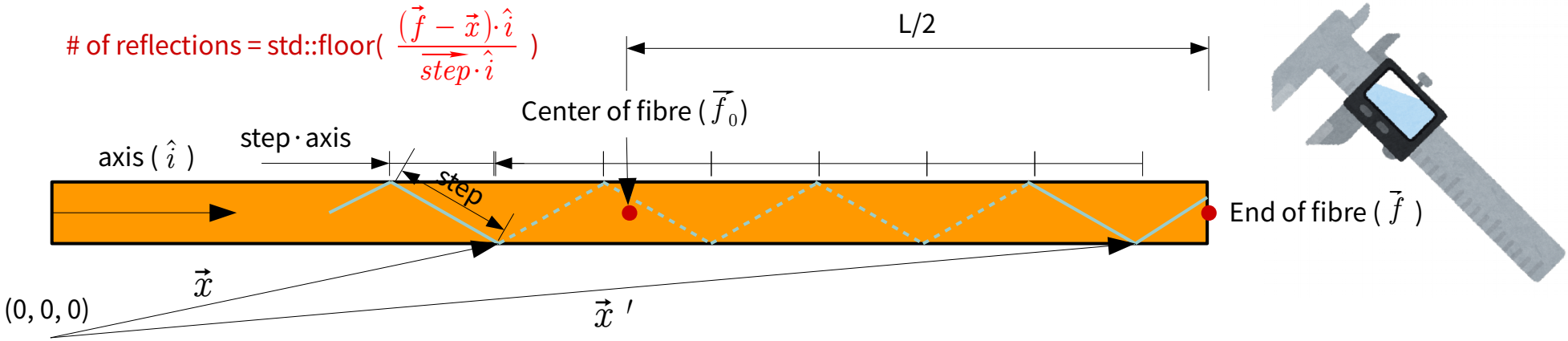


Speeding up optical photon tracking

Developing fast simulation for optical photon tracking

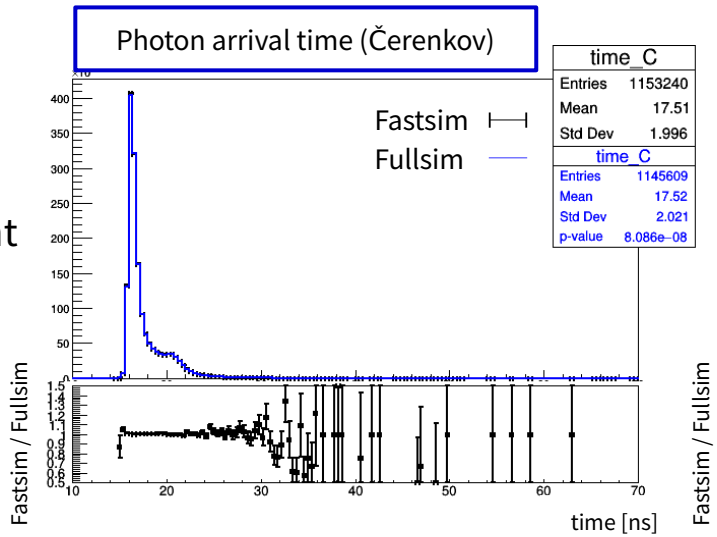
- Simulating photon propagation is necessary, but dominates CPU consumption
- Yet, propagation of optical photons in fibres can be estimated, skipping full tracking

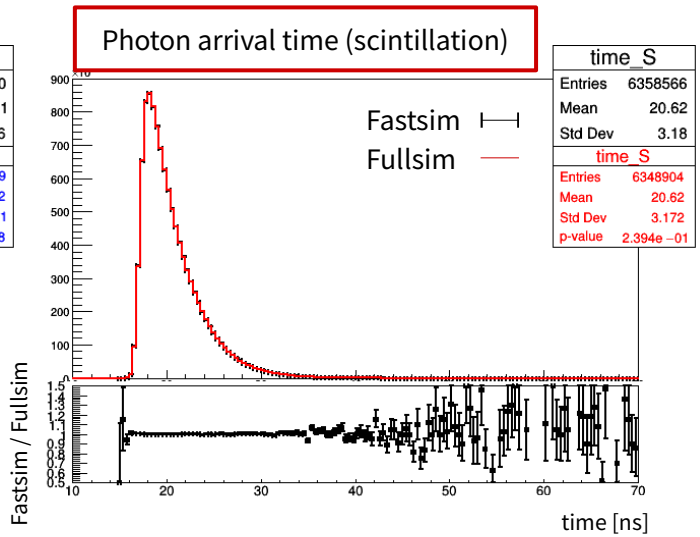
  → Developing a fast simulation module presented at GEANT4 R&D meeting at May 2020



- Preliminary Fastsim model shows excellent agreement with Fullsim
- Takes ~ 4 mins to simulate an event of 20 GeV e-

  → more efforts for further improvement on-going





Backups

Text

$$formula$$