Dual-readout DD4hep migration

Sanghyun Ko

Seoul National University

On behalf of the IDEA Dual-Readout Group

09 Oct 2020

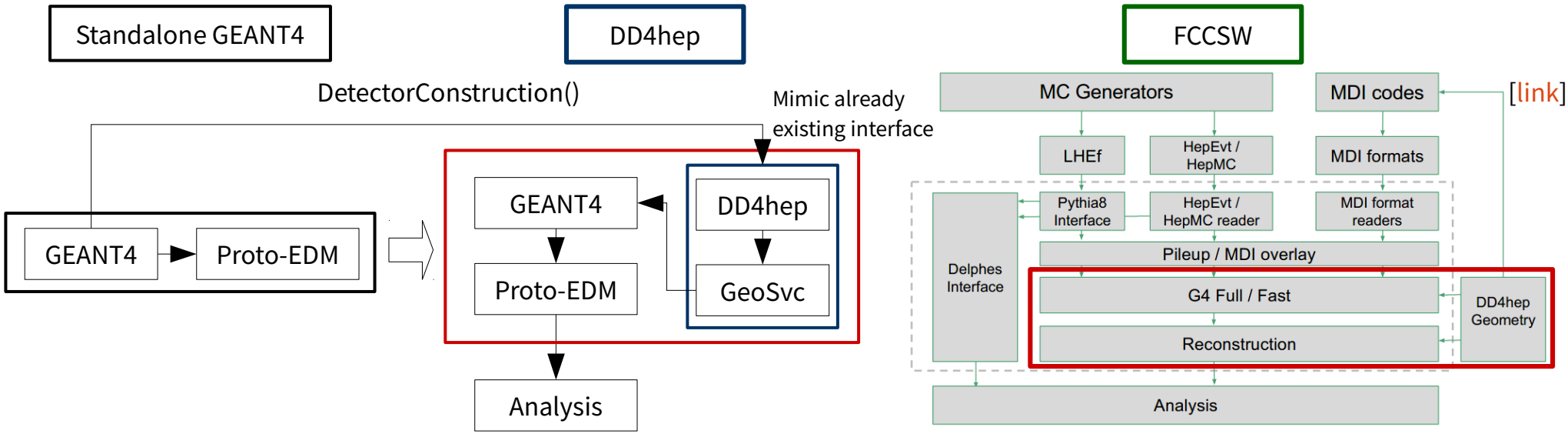
Migration to Key4HEP elements

Migration to Key4HEP elements

- Migration to centralized SW is a good opportunity to take a look into combined detector performances, e.g. tracker+calorimeter, ECAL+HCAL, .etc.
- However, migrating in a one big leap is not an ideal choice.
  - Core software is still rapidly evolving, may introduce extra efforts for integration & maintenance.
  - May need to test several options internally, usage of standalone SW is still too convenient to abandon.

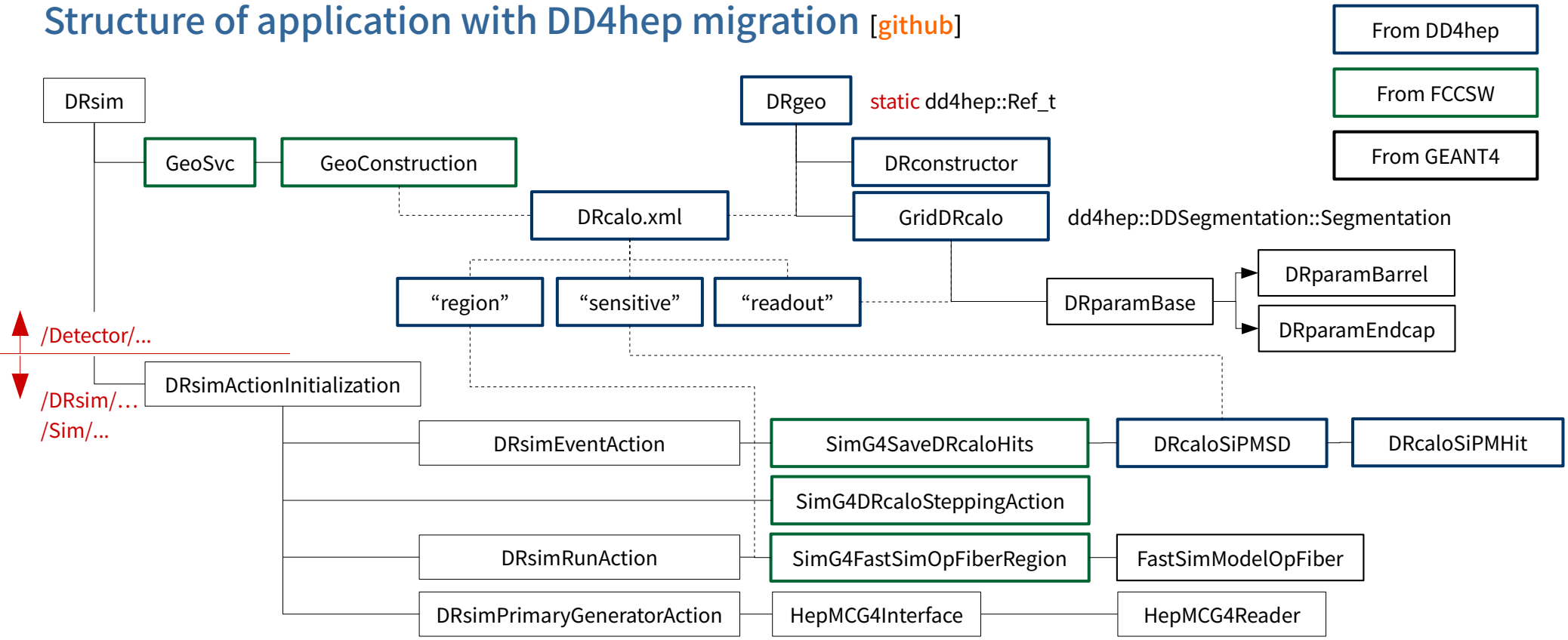
→ Start by migrating to necessary subset of Key4HEP elements (e.g. DD4HEP, EDM4HEP) based on the current standalone SW is a natural choice.

- DD4HEP & GeoSvc interface will provide great convenience already for plugging-in other detector components compared to G4 standalone.



Framework structure

Structure of application with DD4hep migration [github]

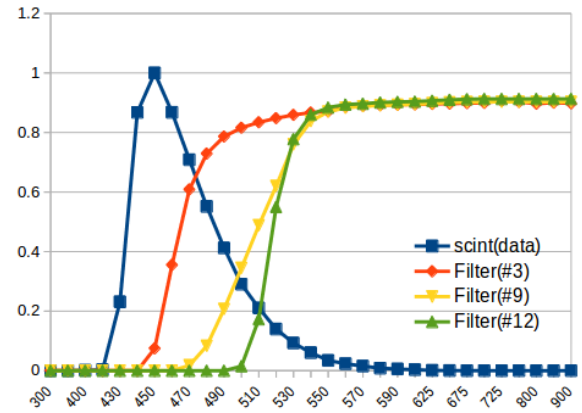


- Structure is aligned based on the call/usage of the classes (solid), links (dashed), and inheritance (arrows).
- DRcalo has an unique projective + grid geometry, dedicated DDSegmentation class is created.
- Interface between GEANT4-DD4hep, Fullsim/Fastsim actions followed FCCSW-way of implementation.
- Only core GEANT4 user actions are remained required to read/write files and run application without introducing external core SW framework.

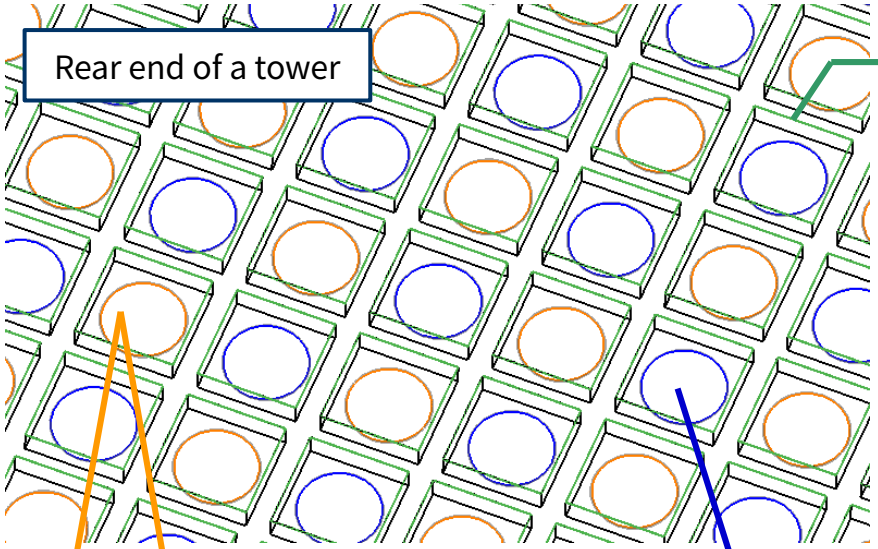
Required optical properties

GEANT4 setup – required optical properties for the simulation [Github]

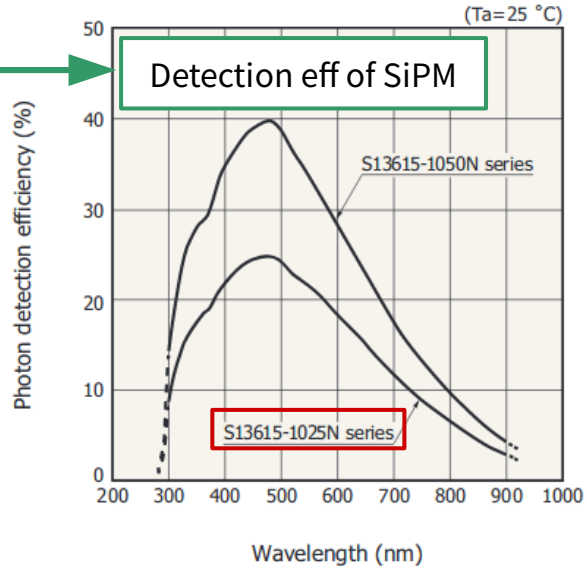
Transmission eff of filters



Rear end of a tower



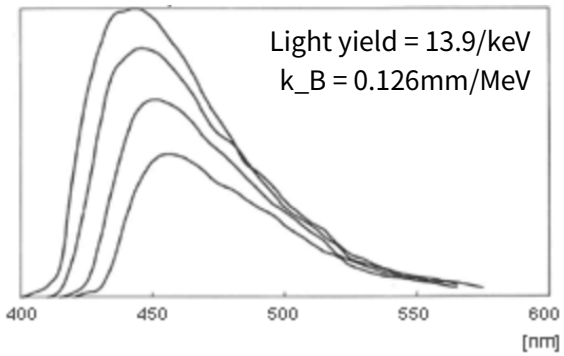
Detection eff of SiPM



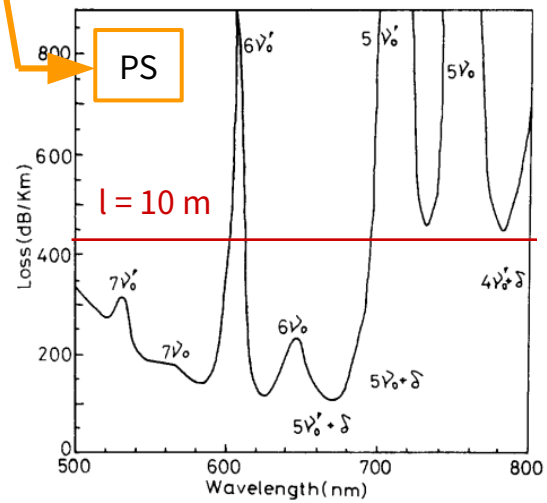
Attenuation loss diverges at 400nm → applied filter to S channel to mitigate it

Attenuation loss of Polystyrene (PS) & PMMA

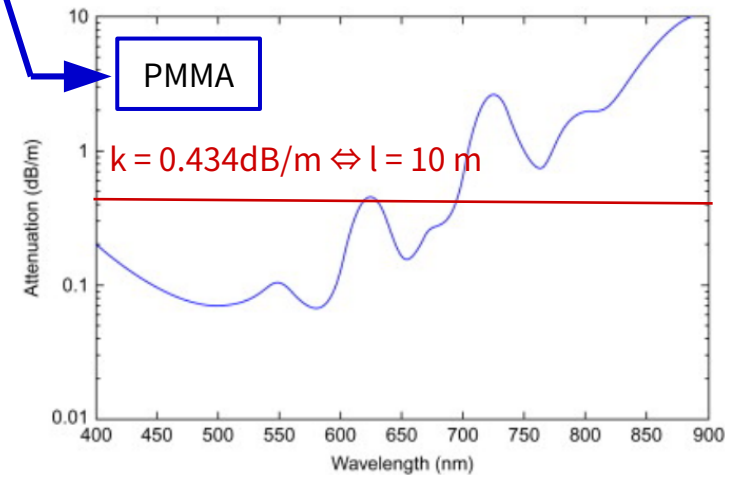
Scintillation spectra of PS



PS



PMMA



DD4hep vs GEANT4

Is detector description of DD4hep & GEANT4 1-to-1?

- Unfortunately not.
- No G4PVParameterised-like method in DD4hep.
  - Usually this does not matter, can be done by looping dd4hep::PlacedVolume.
  - Apple-to-apple geometry may lead to memory problem when the volume is sensitive.

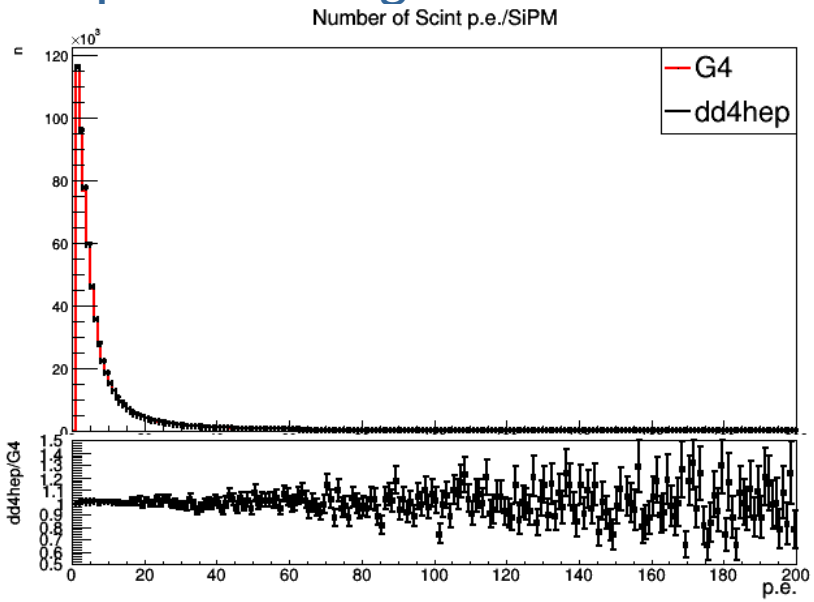
    (DDSegmentation can be used instead in the case)
- There is no corresponding method for setting Birk's constant in DD4hep.
  - In G4, the birk's constant of a material is set by
    - G4Material→GetIonisation()→SetBirksConstant();
  - However in DD4hep, Birk's constant cannot be properly initialized.
  - Still, Birk's constant can be initialized during runtime by
    - GetMaterial()→GetIonisation()→SetBirksConstant();

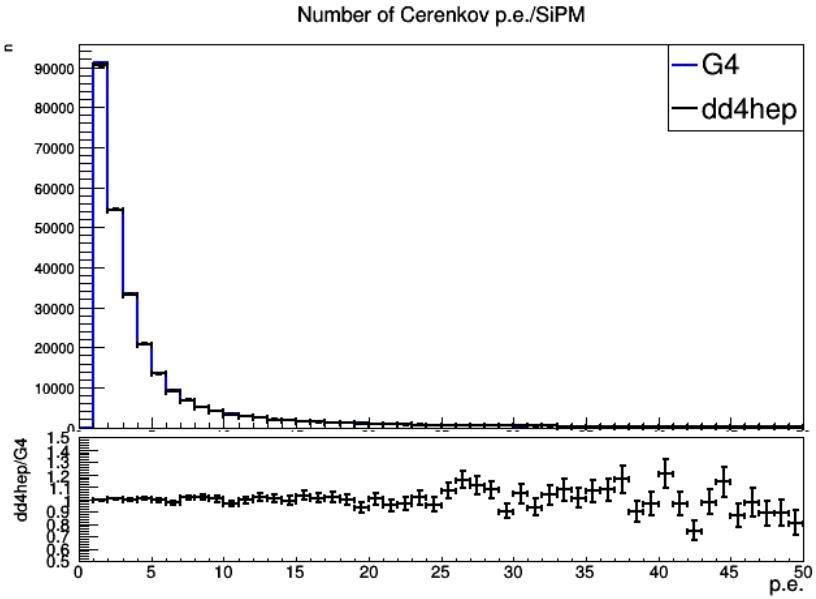
      (since G4Material is a singleton object)

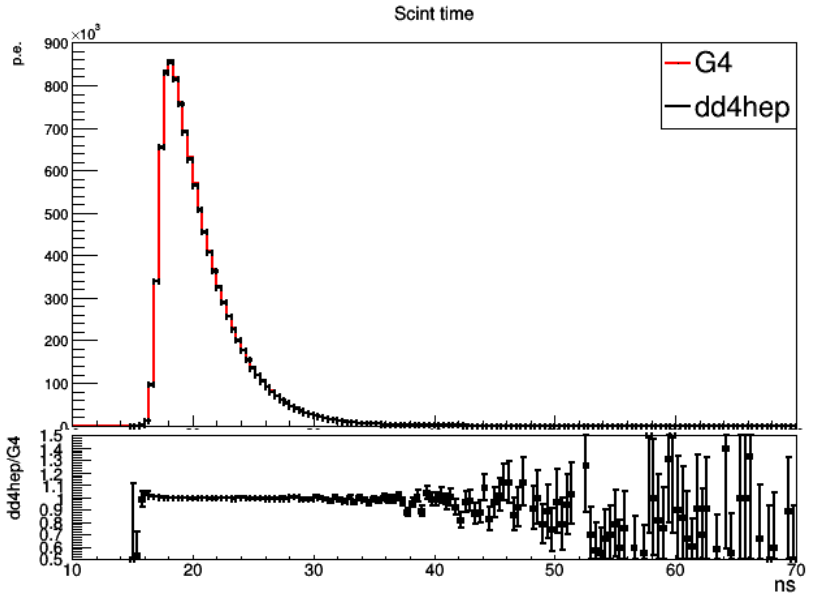
- Tested 20 GeV e- result of DD4hep, using calibration constants from GEANT4, with the first 5 towers of apple-to-apple geometry (due to memory problem).

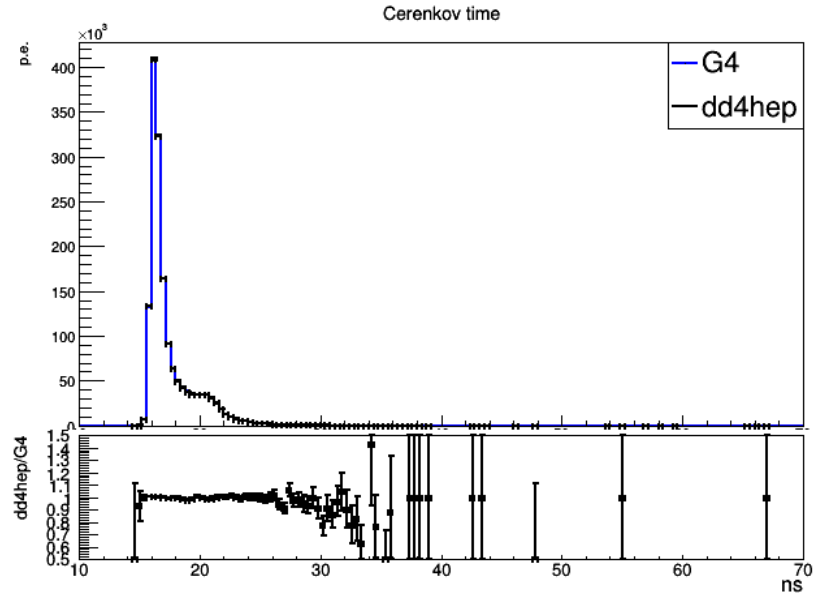
DD4hep vs GEANT4

DD4hep with setting Birk's constant



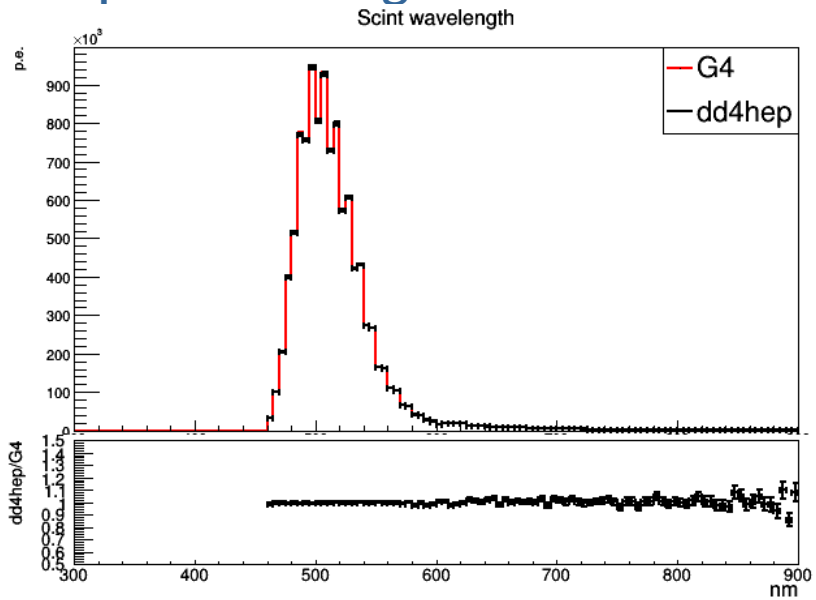


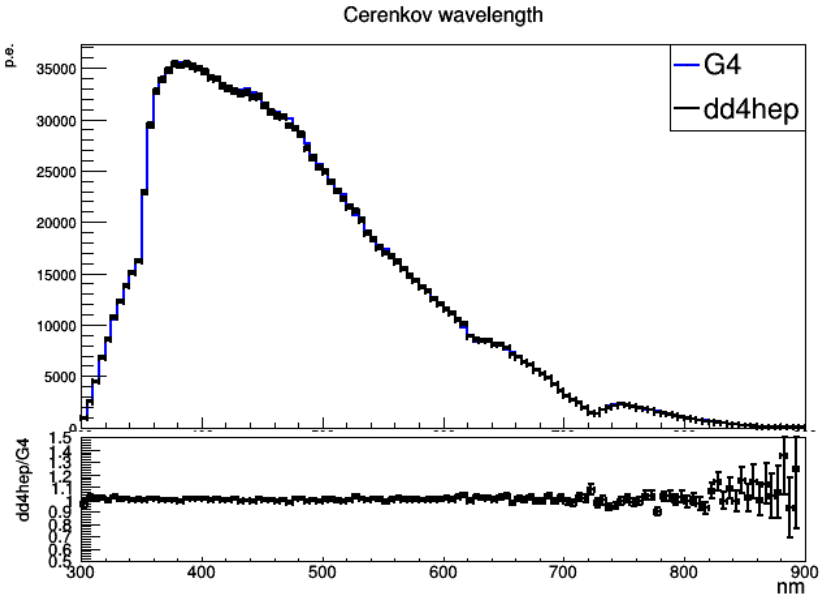


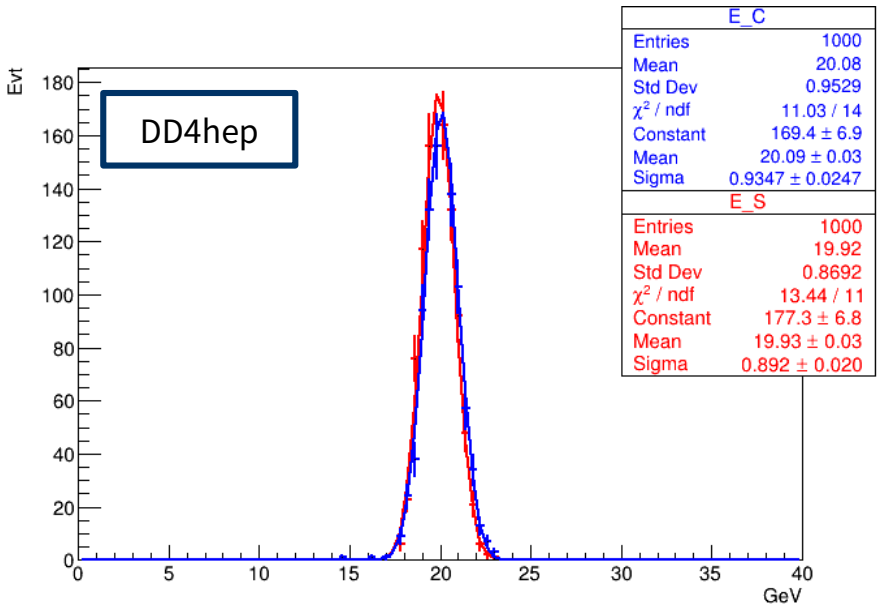


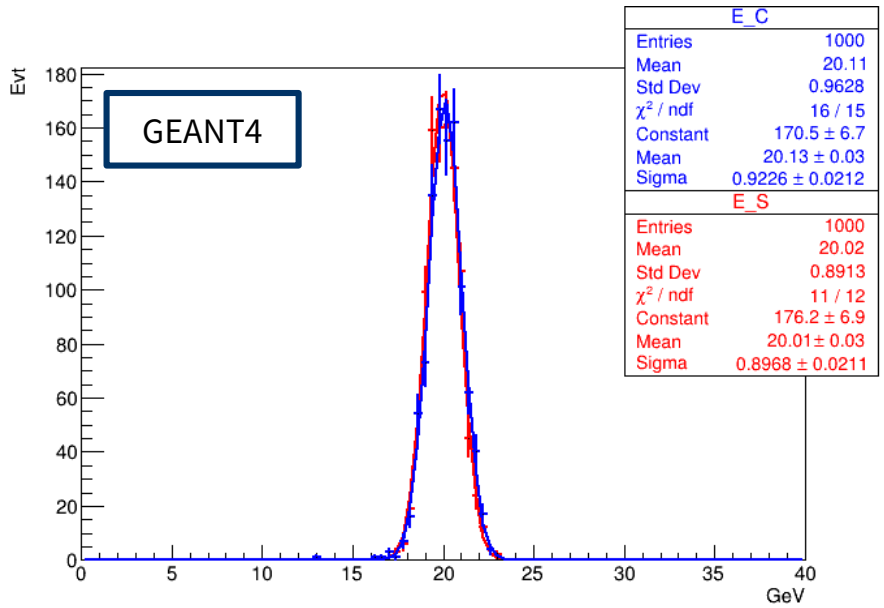
DD4hep vs GEANT4

DD4hep with setting Birk's constant









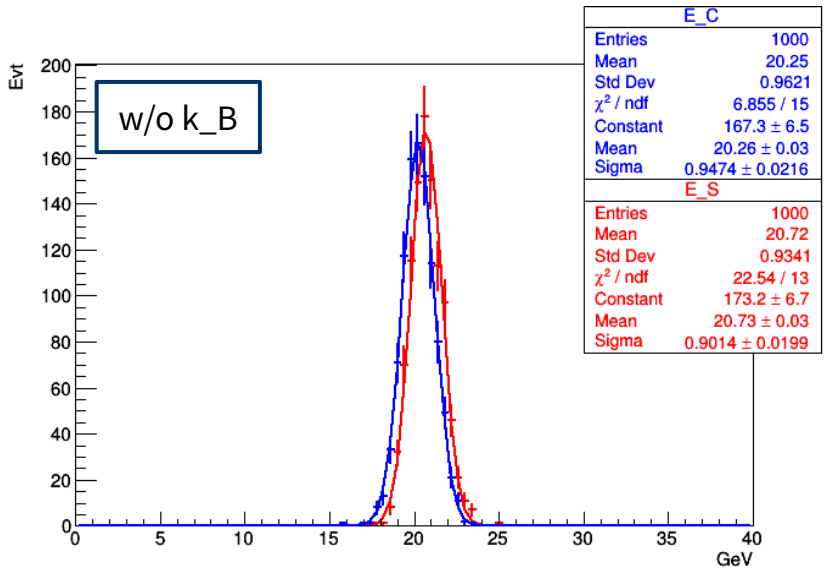
Issue 1 – Birk's constant

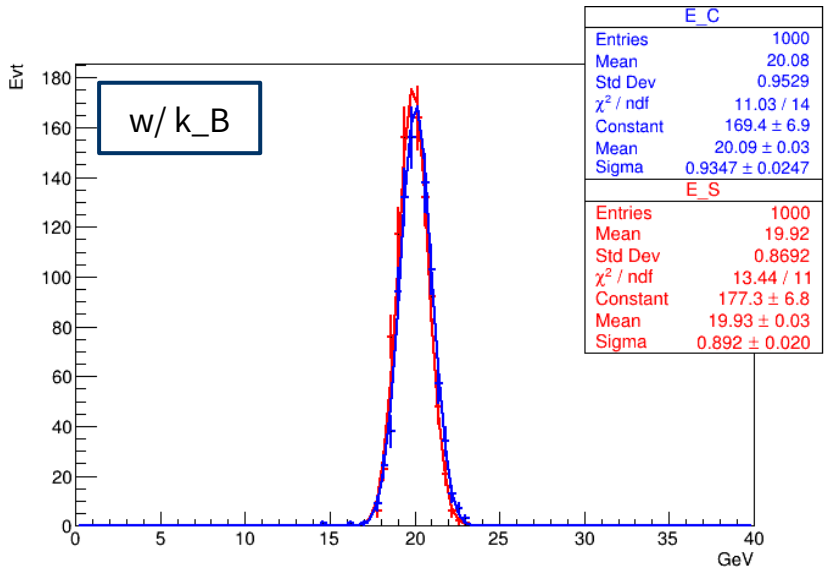
Apple-to-apple comparison of DD4hep vs G4

- After synchronizing the geometry, Birk's constant, the DD4hep completely reproduces the result of GEANT4.
- However, this does not mean that the CPU & memory optimization point of G4 is equal to that of DD4hep.
- Also there are several methods that absent in DD4hep while persist in GEANT4.

Setting Birk's constant in DD4hep

- Birk's constant can be set during runtime using the fact that G4Material is a singleton object.
- However, it would make more sense to set Birk's constant of materials at one of the DD4hep's compact files.
  - Currently, it is set at UserRunAction::BeginOfRunAction() [link]
  - Having xml parsers to wrap G4Material→GetIonisation()→SetBirksConstant() should be sufficient.

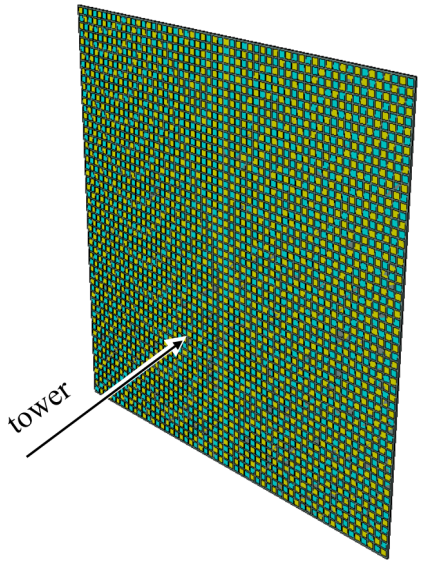


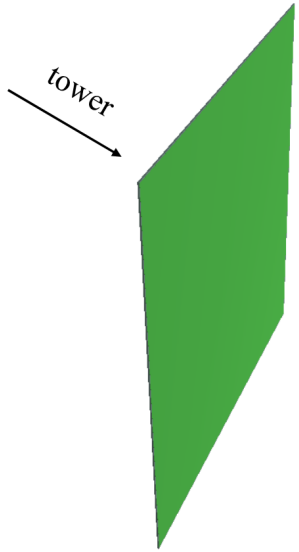


Issue 2 – G4PVParameterised

Memory optimization in DD4hep w/o G4PVParameterised

- For replicating many volumes, G4 provides an optimized method of G4VPhysicalVolume, G4PVParameterised, while DD4hep (essentially TGeo of ROOT) simply executes TGeoVolume→AddNode() multiple times.
- This does not make big difference UNLESS the volume is sensitive or has optical boundary surface.
- Replicating O(10M) of SiPMs explodes memory consumption.
  - Constructing the first 5 towers costs 8 GB ↔ 150 GB for full geometry! (4 GB for full geometry in GEANT4)
- Instead of replicating many sensitive detectors, need to segment a fewer number of sensitive detectors by utilizing DDSegmentation.
  - Able to reduce memory consumption to 4.8 GB in this way.
  - Requires shifts ~ 0.01mm when placing SiPMs – small enough to be mechanically tolerated.





- Have only single photosensitive surface per tower
- Segment the surface by utilizing DDSegmentation (GridDRcalo)

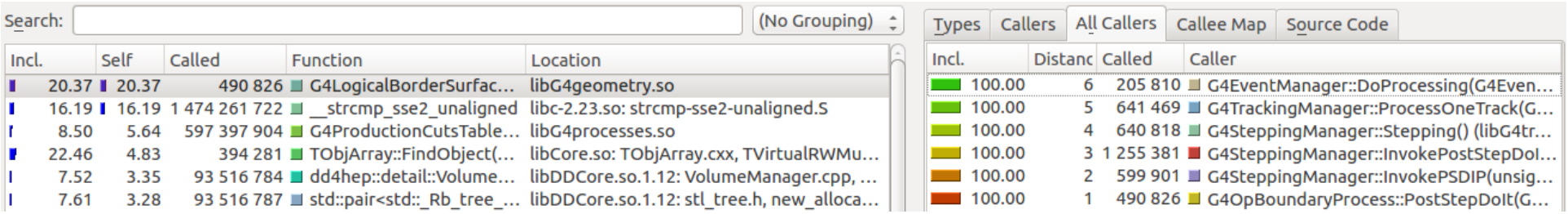
Picture by JW Park

Issue 2 – G4PVParameterised

CPU optimization in DD4hep w/o G4PVParameterised

- Using border surface (optical surface) for Kodak filters in DD4hep consumes 20.37 % + 16.19 % + 4.83 % ~ 40 % of total CPU time.

  (profiled using callgrind with a 100 MeV e- event)
- In G4, utilizing G4PVParameterised makes it able to reduce the # of activated border surfaces.
- However, in DD4hep the # of activated border surfaces is same with the # of Kodak filters (½ of # of fibres).
- Replaced border surface to skin surface (constructor of border surface needs physical volume, while skin surface needs only logical volume).
- Caveat: Need to apply √[transmittance] instead of the transmittance from experimental data.



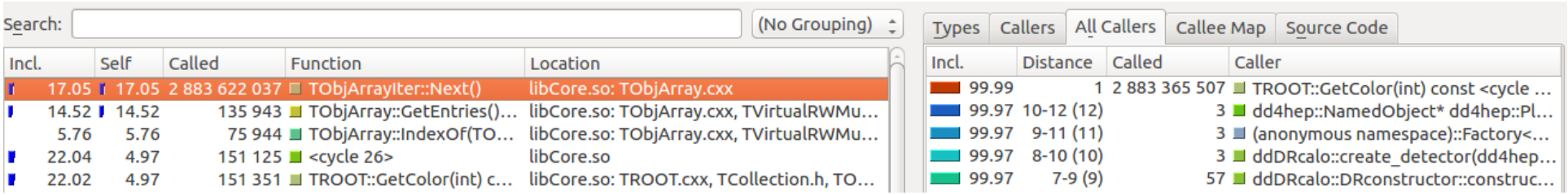
- Although there is makeshift/go-around for G4PVParmeterised, it is too useful not to have it in DD4hep in terms of memory & CPU optimization.
- Wrapper for TGeoVolumeMulti may be the potential alternative for G4PVParameterised for long-term solution.

Issue 3 – setVisAttributes

Avoid setVisAttributes()

- Not related to physics, purely technical.
- setVisAttributes() method in DD4hep (essentially SetFillColorAlpha() in ROOT) takes almost 50 % of total CPU time after replacing Border surface – even when visualization is not used.

  (Corresponding method in GEANT4 does not show this kind of behavior.)



- It seems that DD4hep/ROOT is creating new instance of TColor every time when setVisAttributes is called, regardless of which the same color already exists or not.
- Makeshift: Avoid calling setVisAttributes() method during simulation.

Performance

Current computing performance of dual-readout fullsim

By KY Hwang

	10GeV	20GeV	30GeV	50GeV	70GeV
# of threads			2		
# of event in 1 file			2		
Average memory			~5GB		
Average time taken per 1 event	~ 10min	~ 12 min	~ 16min	~ 22 min	~ 28 min
Average time taken per 1000 evet (w/ 100 cpus)	~ 1h 40min	~ 2h	~ 2h 40min	~ 3h 40 min	~ 4h 40min

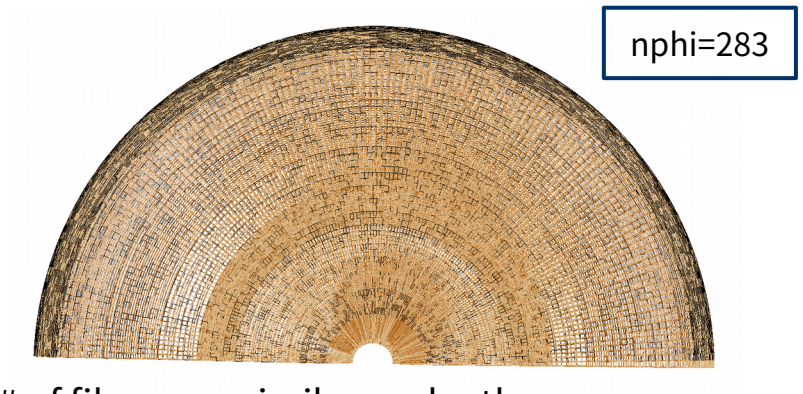
- Computing performance is tested after optimization with electron gun of energy from 10 GeV to 70 GeV.
  - At local institution server with Intel Xeon CPU E5-2680 v2 2.8GHz
  - Fast simulation for optical photon is applied
  - Time taken for geometry construction ~ 7 min
  - Time taken for 10 GeV simulation w/o geometry construction ~ 3 min
- Although calorimeter is the most heavy consumer of fullsim routine, there is still more room to be improved.
  - Exploring multi-threading
  - Improving fast simulation for optical photon tracking

Issue 4 – std::strcmp & placeVolume

Slow down of dd4hep::Volume::placeVolume w/ many daughter volumes

- Although most of the issues had go-around, there is also an issue which could not be resolved.
- dd4hep::Volume::placeVolume is slowed down when the number of daughter volume in the volume is large.

- There is an open topic regarding geometry, about the number of rotation in ϕ direction at the moment.



- Total # of fibres are similar each other.
- The larger single tower is, the larger number of daughter volumes (fibres) in the tower.

- With a small number of daughters, initializing DD4hep geometry (practically) takes similar time to GEANT4.
- However, when the # of daughters per tower exceeds around 10k, it slows down and practically unusable [link].

Issue 4 – std::strcmp & placeVolume

Slow down of dd4hep::Volume::placeVolume w/ many daughter volumes

- Profiled CPU for DD4hep & standalone GEANT4 with only single tower with nphi=36, by shooting Geantino.

Self	Called	Function	Location
46.99	5 113 555 948	__strcmp_sse2_unaligned	libc-2.23.so: strcmp-sse2-unaligned.S
28.13	224 673	TObjArray::FindObject(...	libCore.so: TObjArray.cxx, TVirtualRWMutex.h
7.73	5 112 023 019	TNamed::GetName() co...	libHepMC3rootIO.so.2
4.27	48 668	TObjArray::GetEntries()...	libCore.so: TObjArray.cxx, TVirtualRWMutex.h
2.11	63 948	__gnu_cxx::__normal_it...	libG4geometry.so
1.96	31 978	TGeoVoxelFinder::SortA...	libGeom.so: TGeoVoxelFinder.cxx, TGeoVolume.h, TObjArray...
1.34	1 625 988	__memset_avx2	libc-2.23.so: memset-avx2.S
0.98	19 425 058	_int_malloc	libc-2.23.so: malloc.c
0.71	331 025	G4hPairProductionMod...	libG4processes.so

DD4hep

Self	Called	Function	Location
22.69	320 441	G4hPairProductionMod...	libG4processes.so
10.17	34 554	__gnu_cxx::__normal_it...	libG4geometry.so
8.59	121 234	G4MuPairProductionM...	libG4processes.so
4.18	9 953 712	G4Region::BelongsTo(G...	libG4geometry.so
3.43	88 712	G4BoundingEnvelope::...	libG4geometry.so
3.38	2 277 785	_int_malloc	libc-2.23.so: malloc.c

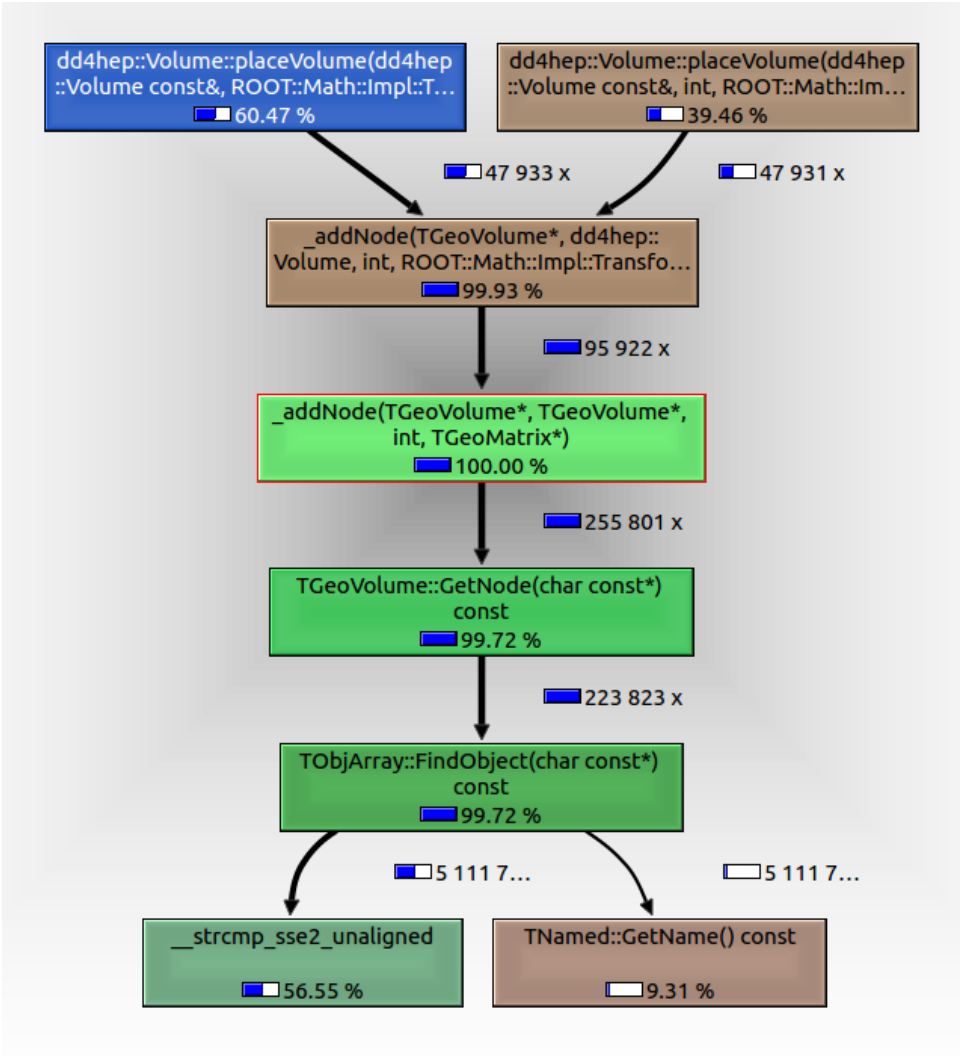
GEANT4

- It turns out that for DD4hep (TGeo of ROOT), std::strcmp is taking most of the time while constructing geometry.
- Meanwhile GEANT4 uses __gnu_cxx::__normal_iterator for navigating G4LogicalVolume* (for G4Region).

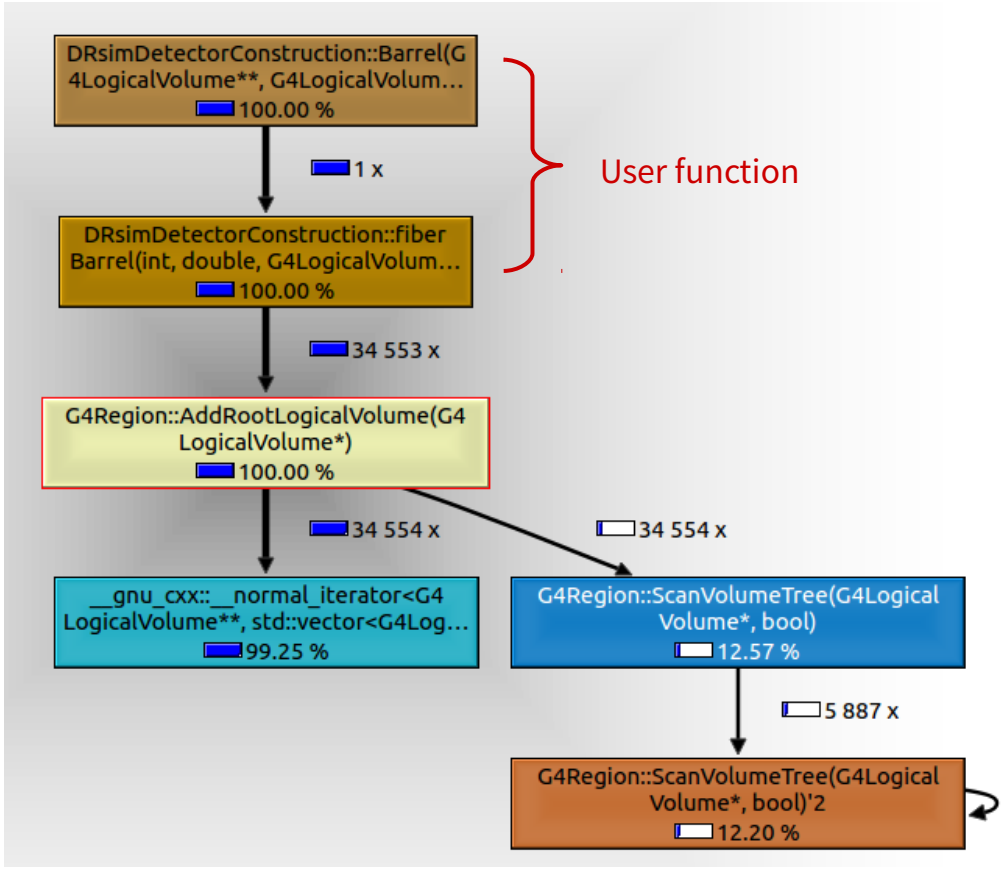
Issue 4 – std::strcmp & placeVolume

Slow down of dd4hep::Volume::placeVolume w/ many daughter volumes

DD4hep



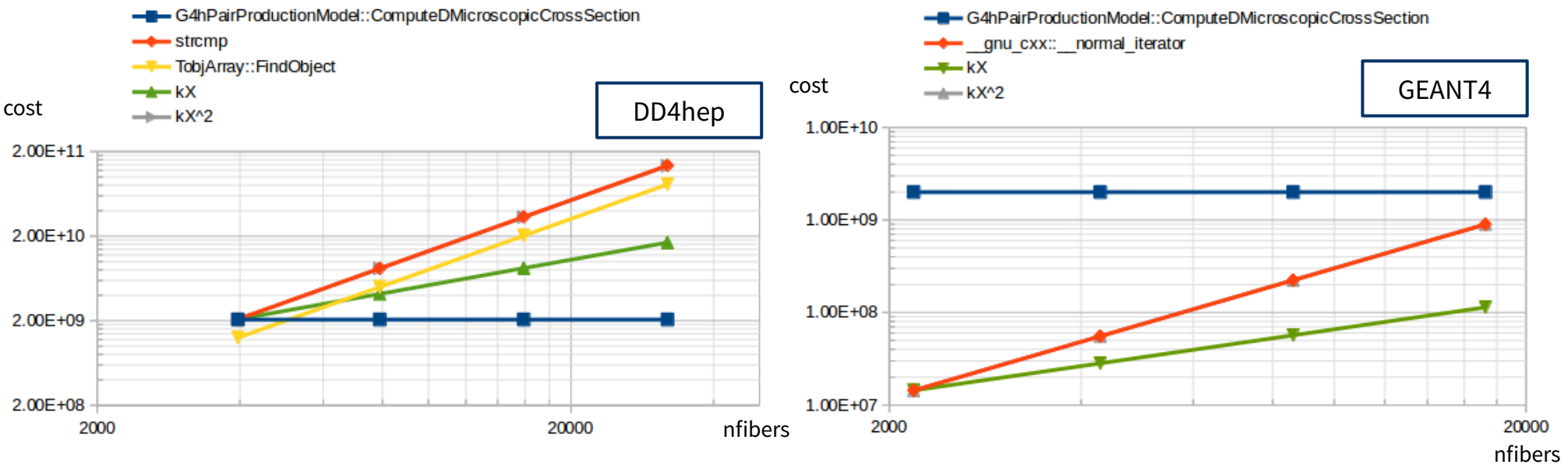
GEANT4



Issue 4 – std::strcmp & placeVolume

Slow down of dd4hep::Volume::placeVolume w/ many daughter volumes

- To profile the effect of the # of daughter volumes within the dd4hep::Volume, measured the CPU cost as a function of the # of fibres inside the tower (~ # of daughter volumes).



- A unit CPU cost is defined as the cost corresponds to 64bit operation.
- G4hPairProductionModel is a method of initializing GEANT4's physics model, constant everywhere (reference).
  - When # of fibres is O(1000), DD4hep geometry construction takes similar order to physics initialization.
  - However, if # of fibres exceeds ~ 10000, it becomes more than significantly slower than standalone GEANT4!

Summary

DD4hep migration of dual-readout calorimeter

- DD4hep reproduces the result of GEANT4 well with apple-to-apple geometry.
- Application is optimized with minor changes of several makeshifts.

Issues encountered while migrating to DD4hep

- No method to set Birk's constant
- No analogous method to G4PVParameterised
- setVisAttributes creates a new instance of TColor every time called
- std::strcmp of TGeo slows down DD4hep's geometry construction phase

Next action items for migration & integration

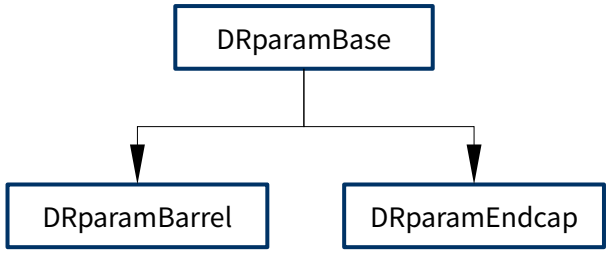
- Adopting the language of Gaudi framework for each method
- Migrate to centralized description of event data model (EDM4HEP)
  - Left for a room for another discussion in the near future :)

Backups

Converging frameworks

Endcap geometry

- Endcap is now migrated to the ‘rectangular’ shape.
- Barrel & Endcap parameterization is inherited from the base class.
  - Allows room for alternative parameterization.
  - Protected member variables with virtual initialization function for trapezoids.

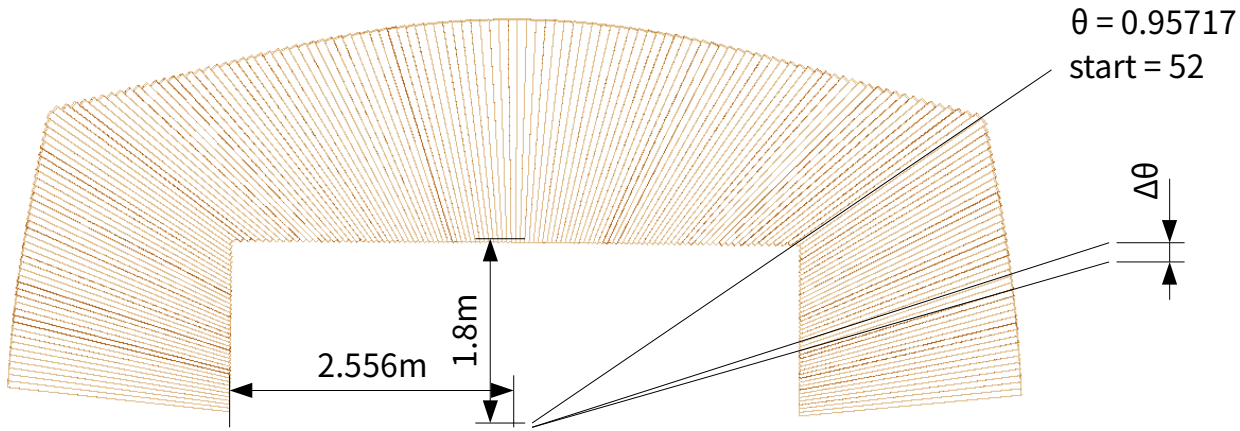


- Compact description contains input parameters to base parameterization class.

```
<barrel height="2.*m" rmin="1.8*m" nphi="283" theta="0.0" start="0" material="Copper" vis="TowerVis">
  <deltatheta deltatheta="0.02222"/>
  <deltatheta deltatheta="0.02220"/>
  <deltatheta deltatheta="0.02217"/>
```

```
<endcap height="2.*m" rmin="2.556*m" nphi="283" theta="0.95717" start="52" material="Copper" vis="TowerVis">
  <deltatheta deltatheta="0.01280"/>
  <deltatheta deltatheta="0.01280"/>
```

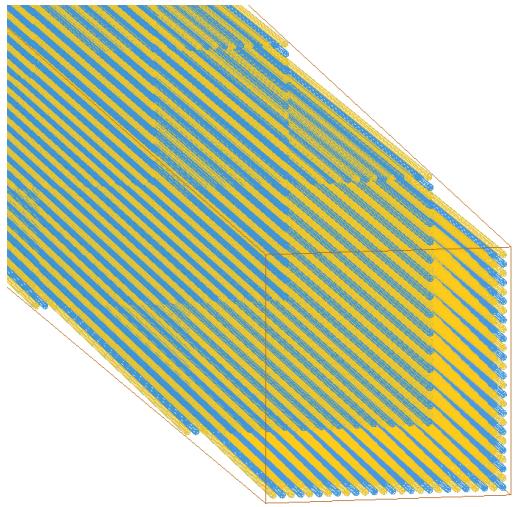
  - Most of the inputs can be controlled by the xml file w/o recompile (e.g. Δθ, # of rotations in ϕ, etc.).



Converging frameworks

Replacing intersection solid to tube

- Putting fiber as intersection solid is inefficient & mechanically makes no sense.
- Replaced fibers to regular-shaped tubes (as already done in INFN).



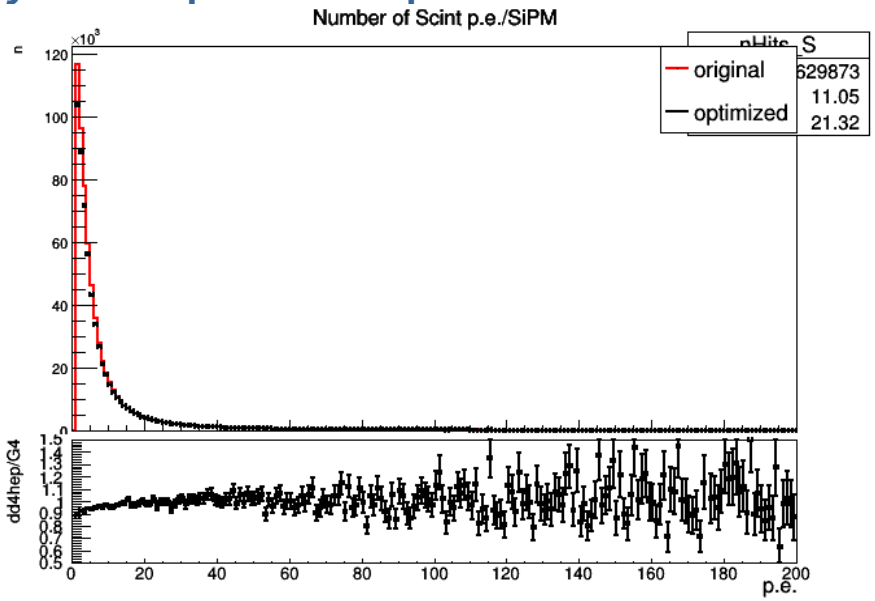
Counting optical photons

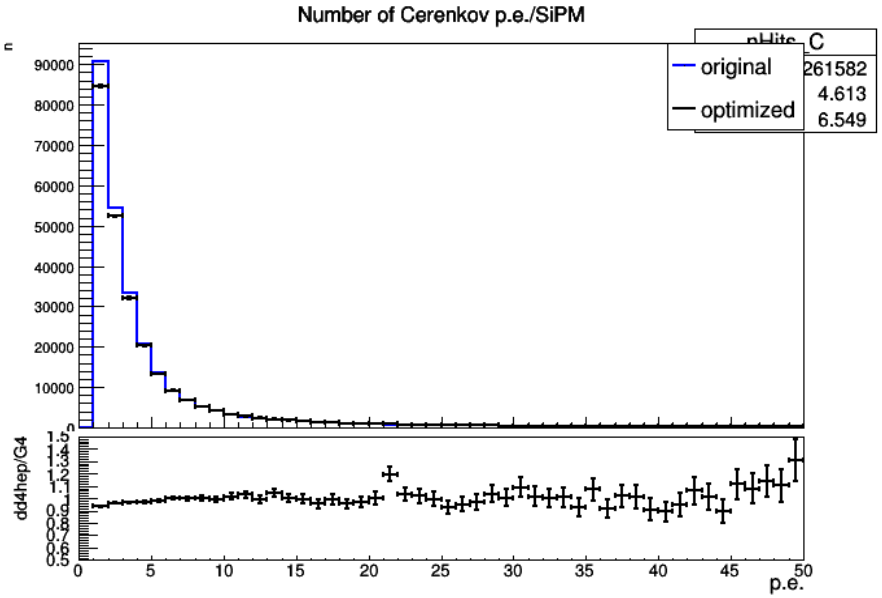
- There is an open choice of the way how the light yield of each channel is estimated.
  - Fully transport optical photons through the fibres
  - Smear the energy deposits within the fibres based on Birk's law
- At the moment, the framework will transport optical photons through the fibres by default (w/ Fastsim).

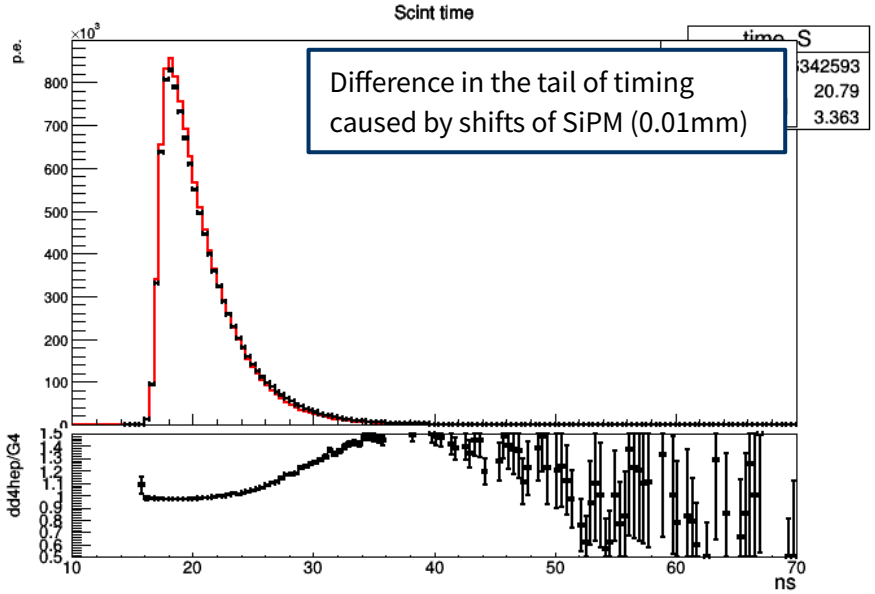
- Still, there is a way that both methods can be used (or switched on/off) simultaneously.
  - Already made energy deposits in each Č/S fibre stored in outputs.
  - A smearing sequence can be implemented in SimG4DRcaloSteppingAction [link] with minimal efforts. (purely GEANT4, nothing to do with DD4hep)
  - Proto-EDM can be extended so that either information is available w/o extra action.

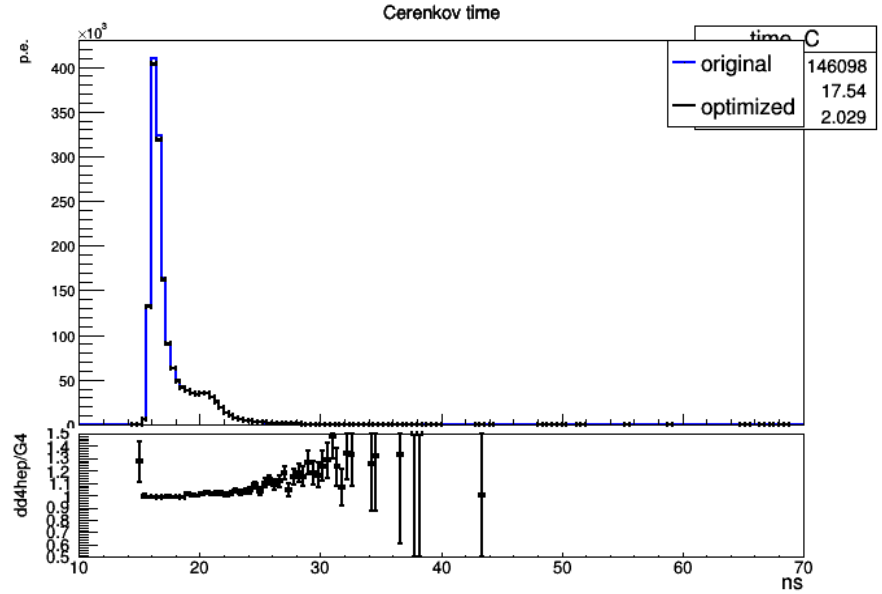
Physics impacts

Physics impacts of optimization



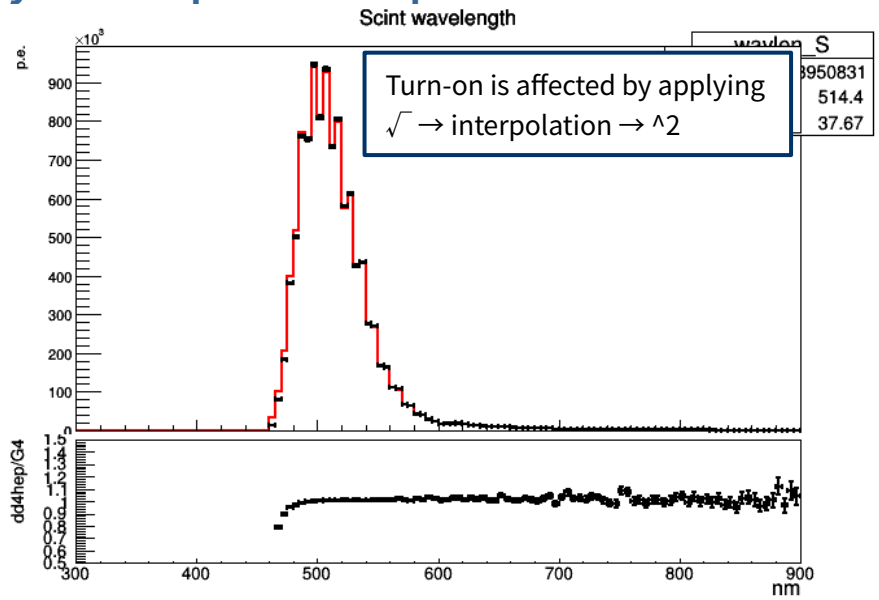


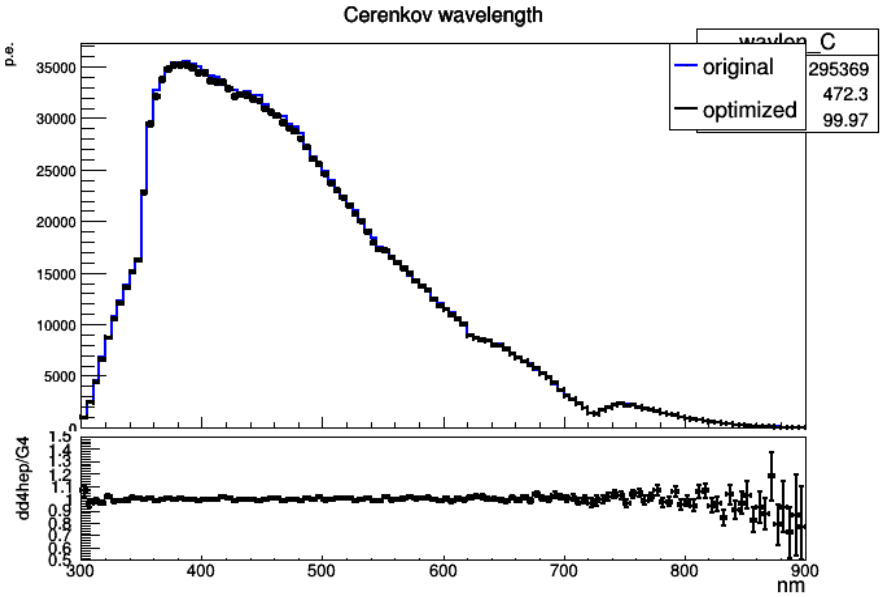


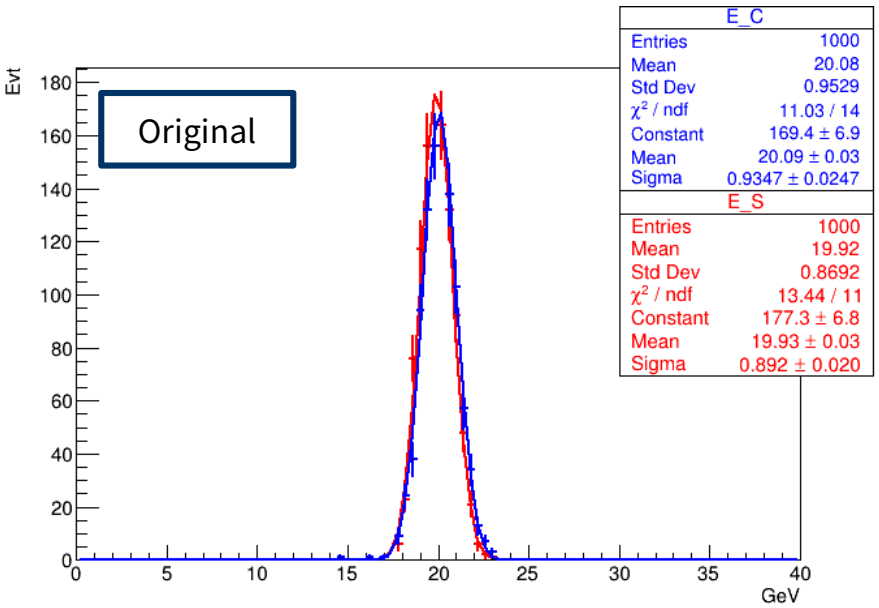


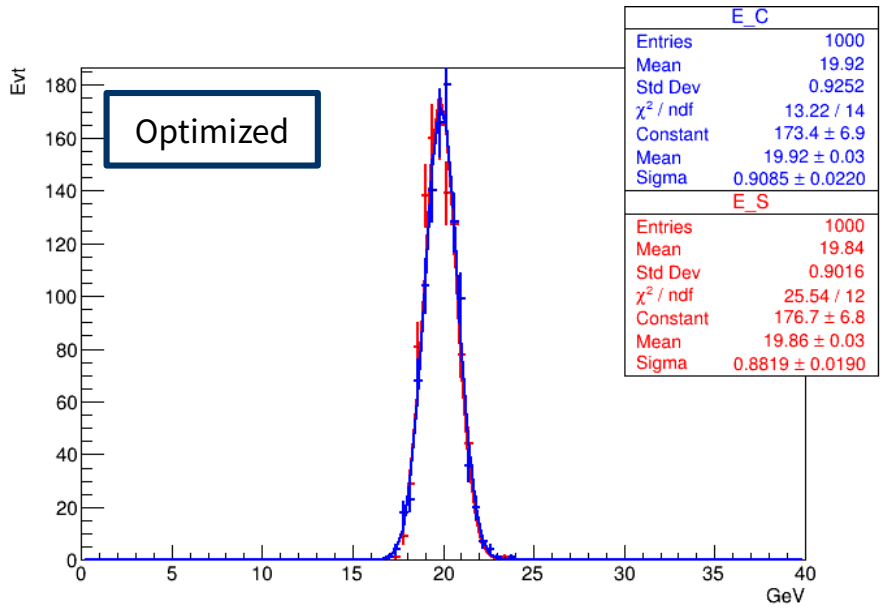
Physics impacts

Physics impacts of optimization









Issues w/ DD4hep & EDM4hep

Summary of the list of issues experienced with DD4hep & EDM4hep

- No G4PVParameterised-like method in DD4hep.
- G4Material→GetIonisation()→SetBirksConstant() not available in DD4hep.
- Unusual # of calls TObjArrayIter::Next() when setVisAttributes() method is frequently used in DD4hep.

- Problem with incorporating MC truth energy deposit (SimHit) & # of photon counts (RawHit) simultaneously.
  - Typical DD4hep examples retrieve MC truth energy deposit of sensitive materials only (e.g. scintillator).
  - However, energy deposit of absorber materials (e.g. Cu) & # of photons counted in the readout system are needed as well (before digitization).
  - This is not possible since there can be only one SD & readout per DD4hep detector description.
  - As a makeshift, MC truth energy deposits are retrieved at UserSteppingAction of GEANT4 at the moment.

edm4hep.yaml

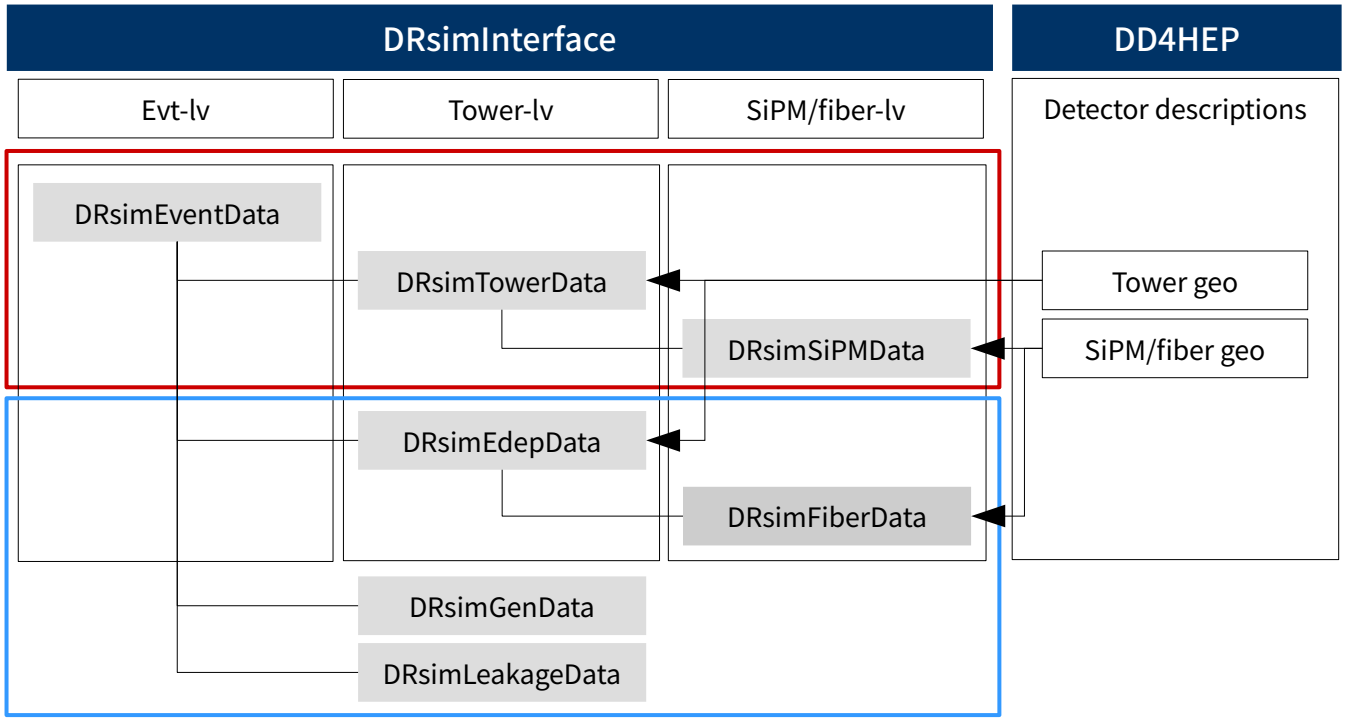
```
#------------ SimCalorimeterHit
edm4hep::SimCalorimeterHit:
  Description: "Simulated calorimeter hit"
  Author : "F.Gaede, DESY"
  Members:
    - unsigned long long cellID       //ID of the sensor that created this hit
    - float energy                    //energy of the hit in [GeV].
    - edm4hep::Vector3f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


#------------  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.
```

EDM with DD4HEP

Effect of DD4HEP to proto-EDM

- Dual-readout calorimeter has very complicated geometry with ~ O(100M) of channels.

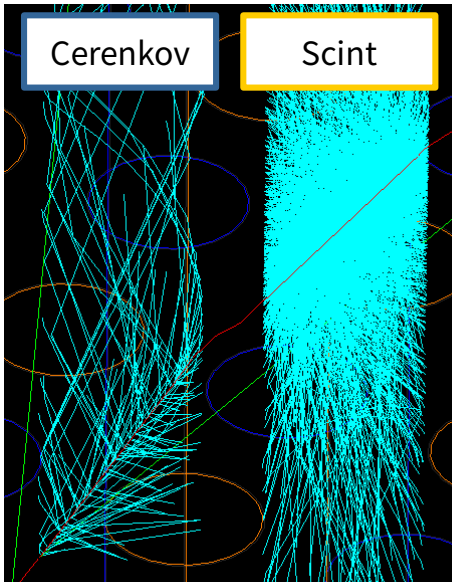
  i.e. geometry information (e.g. position of fibres/SiPMs) ideally should not be repeated in every event.
- Instead of saving geometry information in EDM, geometry information will be loaded by DD4HEP using a unique readout ID with a encoded 64bit integer that contains all necessary informations to identify it.
- In this way DD4HEP can be interfaced to both of GEANT4 & reconstruction codes to utilize geometry information.

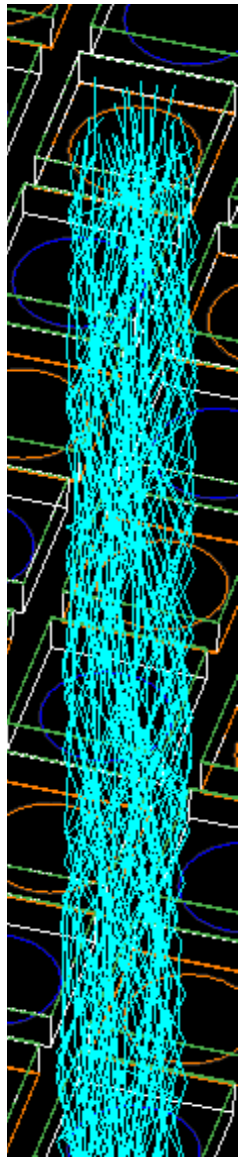


Speeding up optical photons tracking

Details needed to simulate dual-readout calorimeter

- Cerenkov & scintillation processes.
- Light attenuation of Polystyrene & PMMA.
- Transmission of optical surfaces, e.g. yellow filter, SiPM.
- Total internal reflection inside optical fibers.
  - Numerical aperture is important for the yield of Cerenkov channel.





CPU consumption for tracking optical photons

- A drawback for detailed simulation is CPU consumption caused by tracking optical photons.
- Single photon generates ~ O(10k) tracks for tracking, while there are ~ O(10k) optical photons per GeV of incident particle, results ~ O(100M) tracks per GeV.
- It takes 304 ± 88 min in average to produce an event! (20 GeV e-).

  → Developed a fast simulation module [Github] to compute efficiently while keeping the details.
- The module skips the intermediate tracking of optical photons and predicts the final position, time and momentum – by utilizing the fact that fibers are cylinder shape (i.e. facet normal always heads the center of the fiber).

  → Reduces CPU time to 4.62 ± 1.17 min per event (20 GeV e-) with identical results to full tracking.

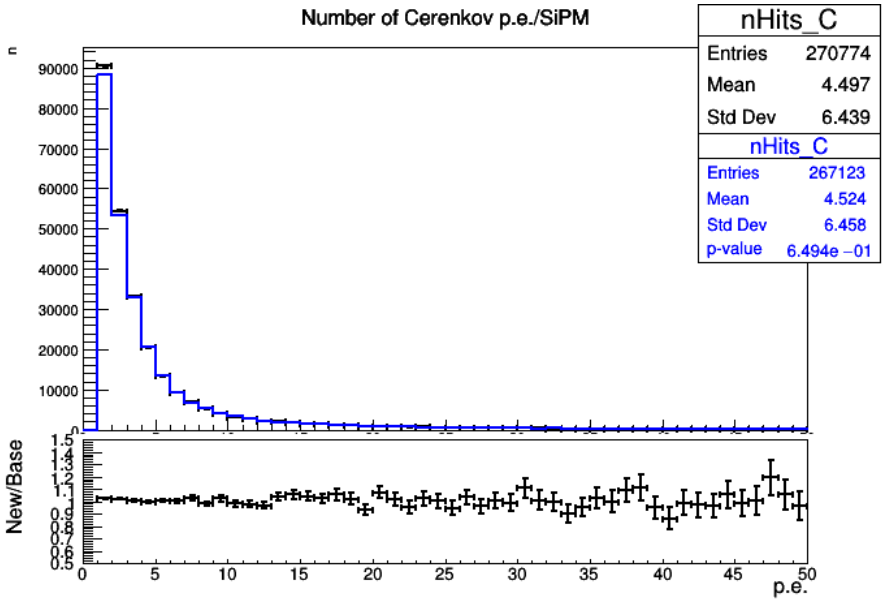
Speeding up optical photons tracking

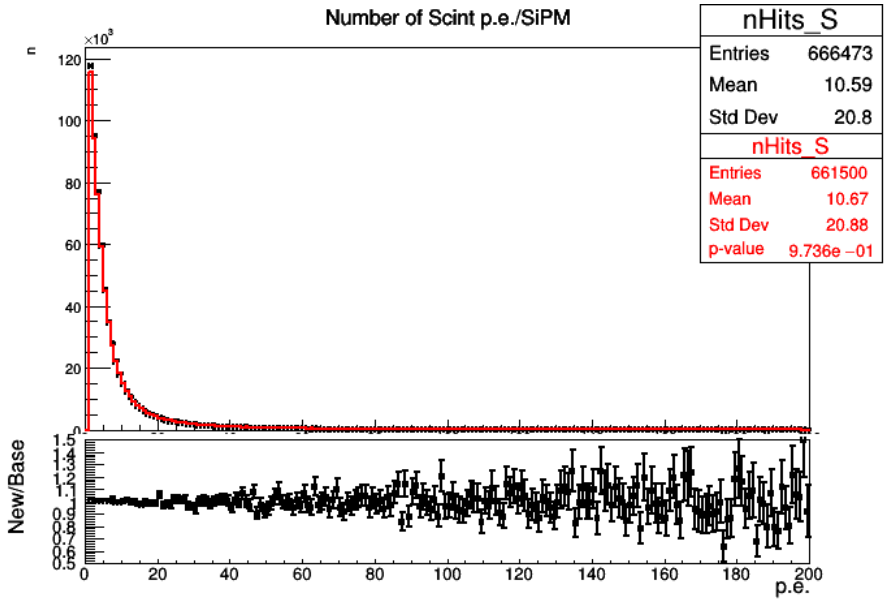
Comparison to full tracking & conventional fast simulation (GFlash)

- Role of the developed fastsim module is limited to managing optical photon transportation inside optical fibers.
- EM/hadronic physics is same to that of full simulation of G4, i.e. does NOT utilize any shower parameterization.

	GEANT4 fullsim	Fast Op transport	GFlash
Shower physics	Full tracking		Parameterization
Relative differences to GEANT4 fullsim	N/A	Optical photons inside optical fibers	EM/hadronic particles in the region of interest

- Comparison of Fast Op transport to full tracking shows good agreement.

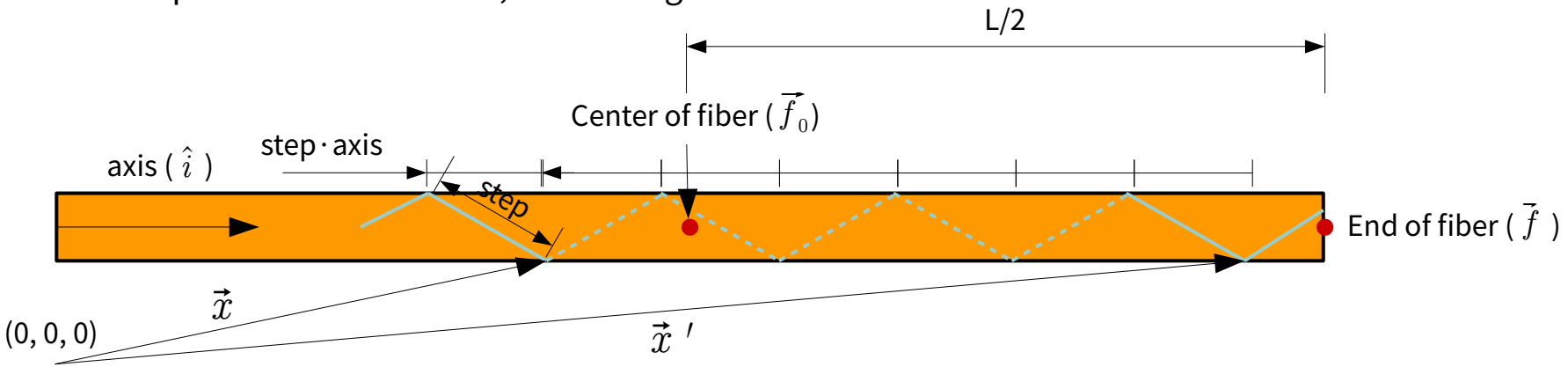




Fast simulation

Estimating the target point of translation for fast optical photon transportation

- Based on the postulate that the step length of individual track remains same throughout whole transportation, the point of translation can be estimated easily.
- $\vec{f} = \vec{f}_0 + L/2\,\hat{i}$
- $\vec{f}_0$ & $\hat{i}$ can be obtained by G4TouchableHandle (touchable→GetHistory()→GetTopTransform().Inverse().TransformPoint/Axis(x, y, z))
- # of expected reflections = std::floor($\frac{(\vec{f} - \vec{x})\cdot\hat{i}}{\overrightarrow{step}\cdot\hat{i}}$)
- $\vec{x}\,' = \vec{x} + (\overrightarrow{step}\cdot\hat{i})\hat{i} \times$ # of expected reflections
- $t' = t$ + step/velocity $\times$ # of expected reflections
- User can require n times more total internal reflections by using (# of expected reflections – n).
  - n = 2 is sufficient to make sure everything works.
- If # of expected reflections < n, do nothing.



Fast simulation

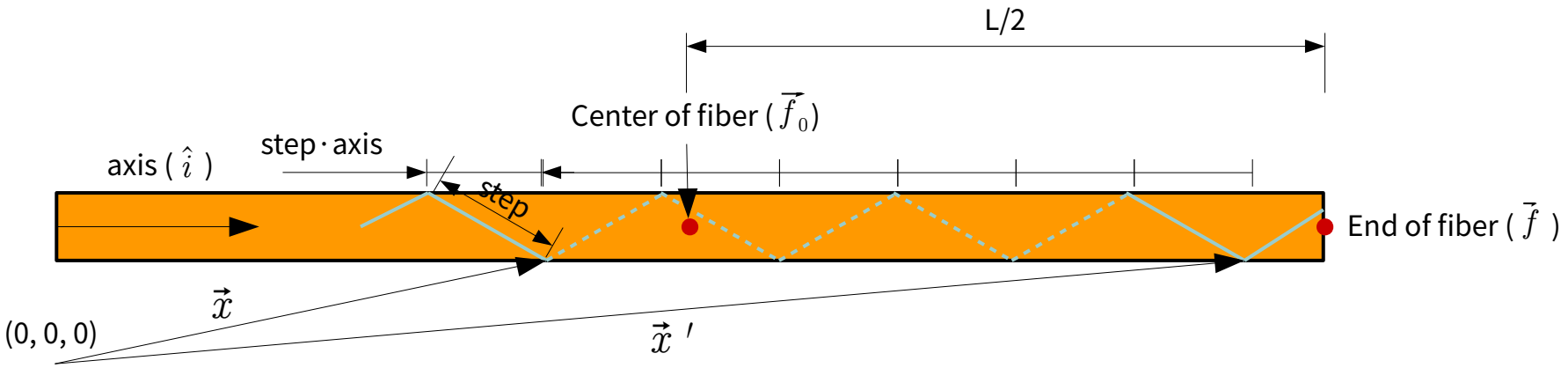
Checking absorption probability of an optical photon

- Skipping intermediate tracking of optical photon forces to check absorption probability by the model.
- In GEANT4, interaction probability with a matter of a particle is given as a 'lifetime' as a unit of interaction length.

  i.e., # of interaction length left = -std::log(G4UniformRand())
- The particle is killed when the travel length exceeds # of interaction length left.

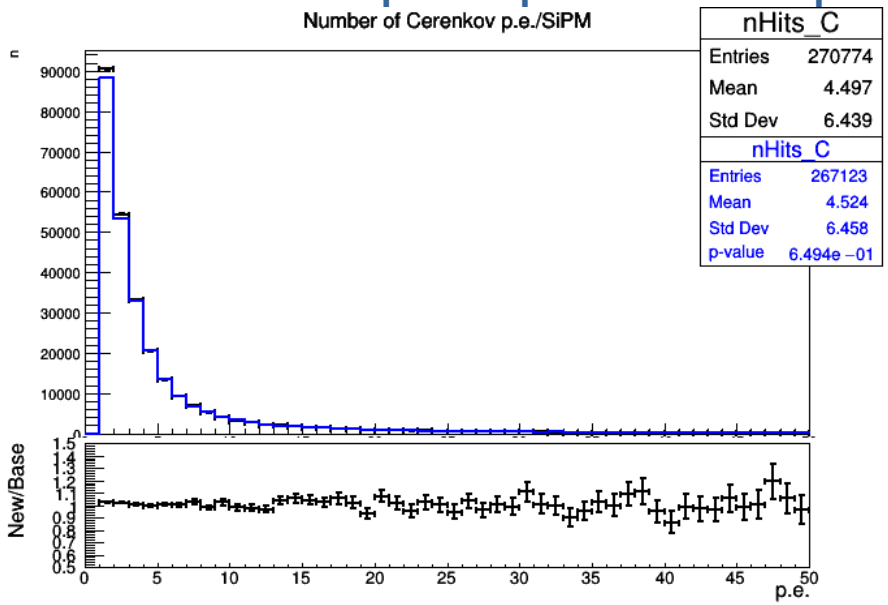
- For a fast transported optical photon, absorption can be checked via
  - # of expected reflections × steplength / attenuation length > # of interaction length left
- Attenuation length of a material can be accessed using G4MaterialPropertyTable.

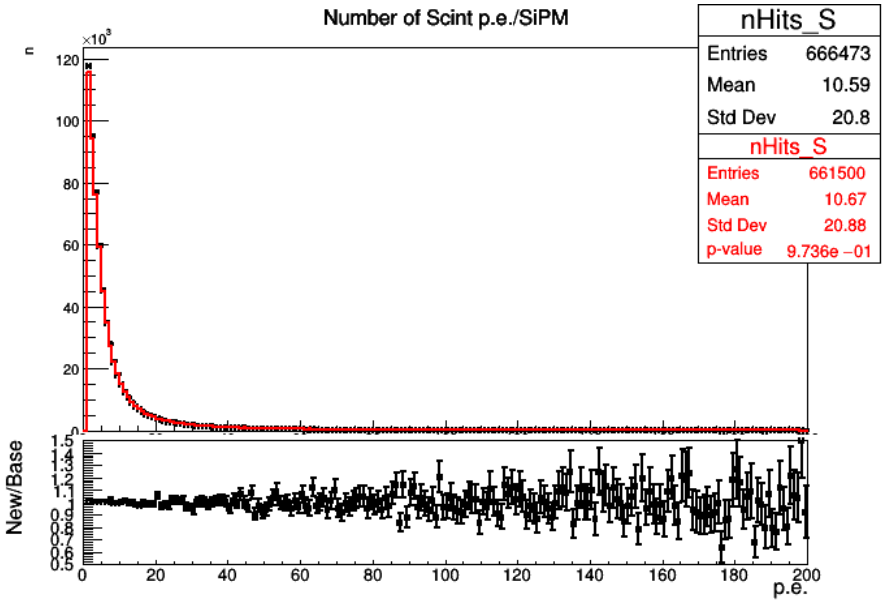
  matPropTable→GetProperty(kABSLENGTH)→Value(momentum)

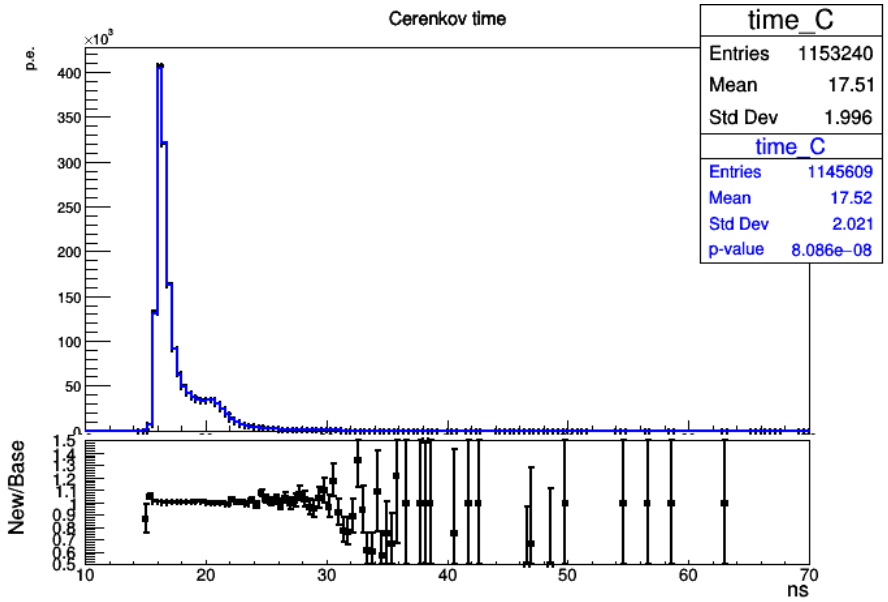


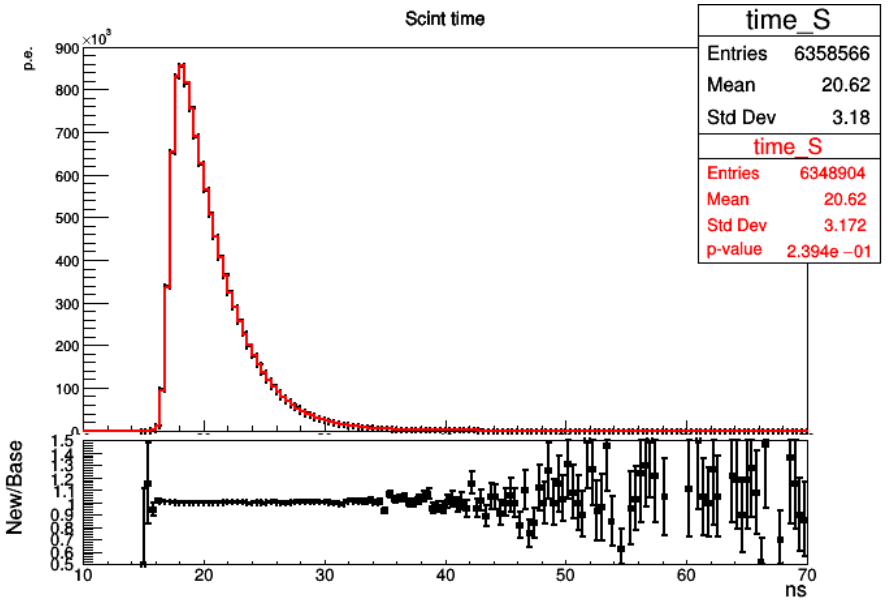
Fast simulation

Validation of fast optical photon transportation



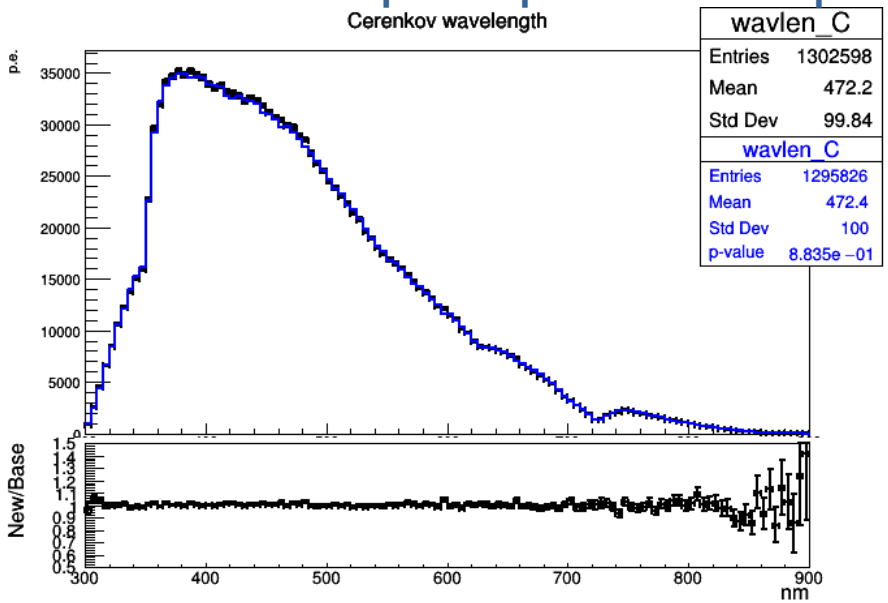


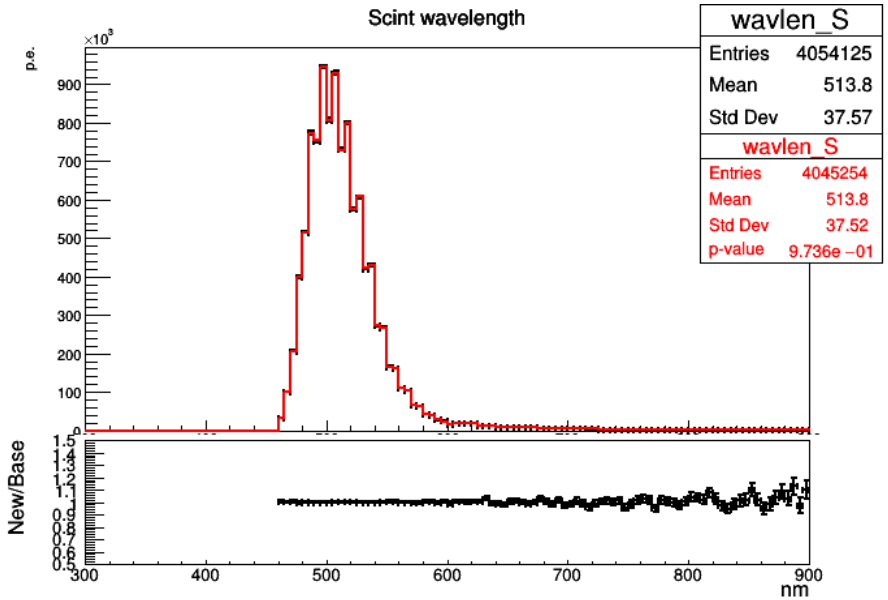




Fast simulation

Validation of fast optical photon transportation





Improvement in CPU consumption using fast optical photon transportation

- It takes 4.62 ± 1.17 min in average to produce an event (tested with 1000 of 20 GeV electron events).
- While it was 304 ± 88 min when using full tracking with the same server.
- Almost ~ 70 times faster than full tracking!

- Initial proposal of the idea was presented at GEANT4 R&D meeting [link][Github].
- Planning to promote the development as a generic plug-in or module of GEANT4 for optical fiber simulation under the supervision of GEANT4 experts.

Title

Text

$$formula$$