# *Software Framework of the DarkSHINE Experiment*

**Yulei Zhang**

*Shanghai Jiao Tong University*

January 20th , 2023

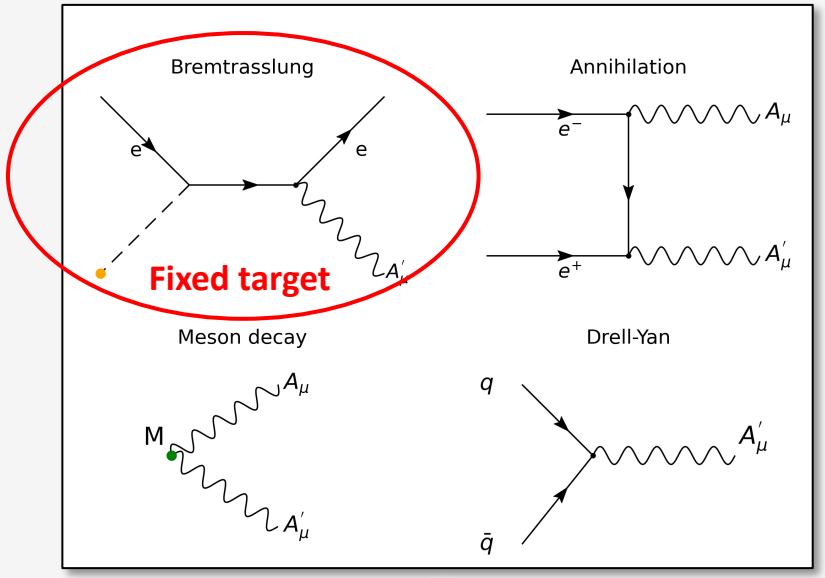
# What is DarkSHINE?

A proposed fixed-target experiment

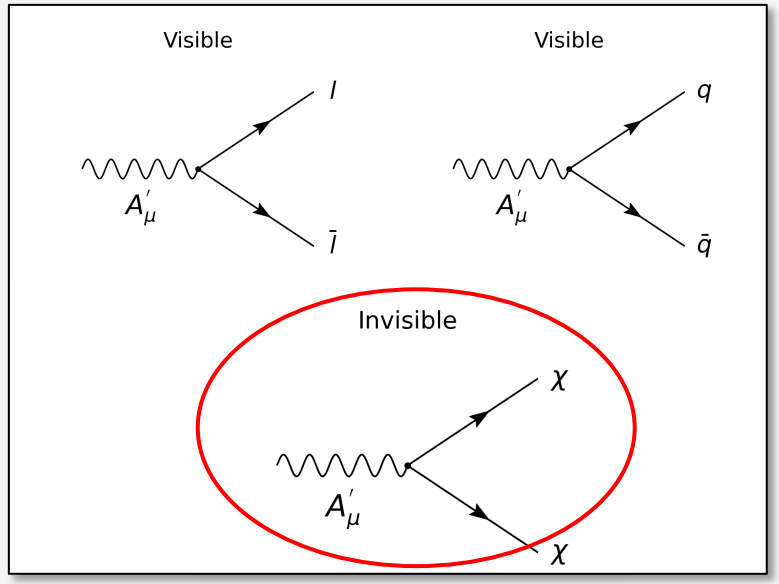
Dr. Jing CHEN's talk [link]

Prof. Shu LI's talk [link]

# Physics process and anticipated signatures




- **Bremsstrahlung, $eZ \to eZA'$ & $pZ \to pZA'$, fixed-target experiment**
- Annihilation, $e^+e^- \to A'\gamma$, $e^+e^-$ collider
- Drell-Yan, $q\bar{q} \to A'$, hadron collider / fixed nuclear target w/ proton-beam
- Meson decay, $\pi^0 \to A'\gamma$ or $\eta \to A'\gamma$ (w/ $m_{A'} < m_{\pi,\eta}$), any experiment w/ high meson production rates



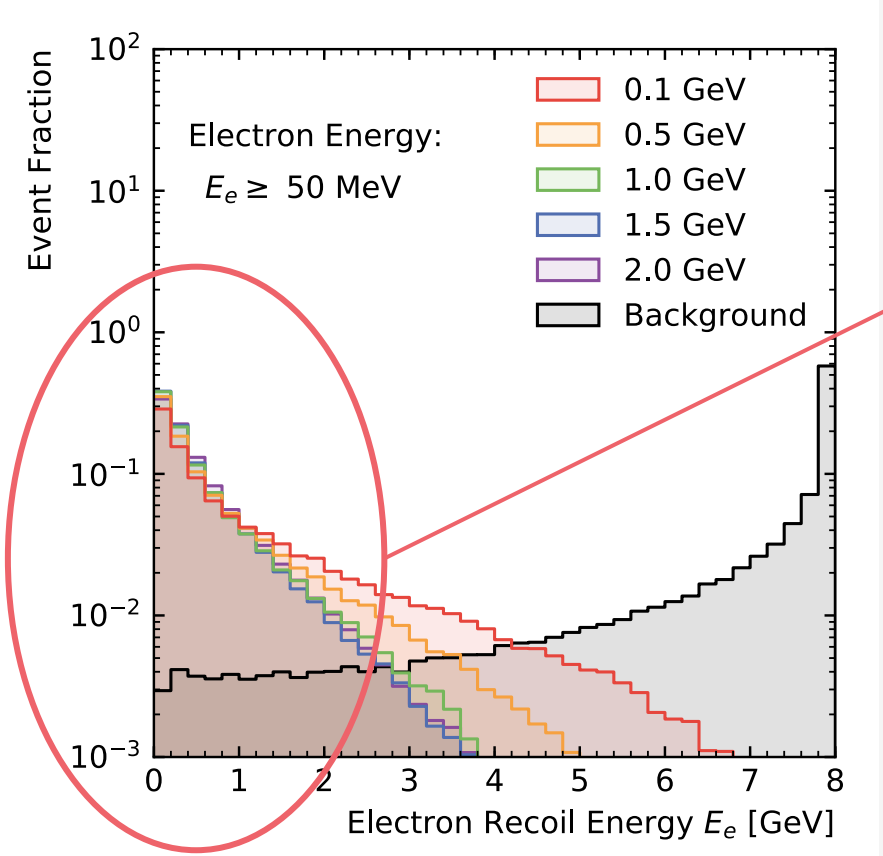

- **Visible decay**

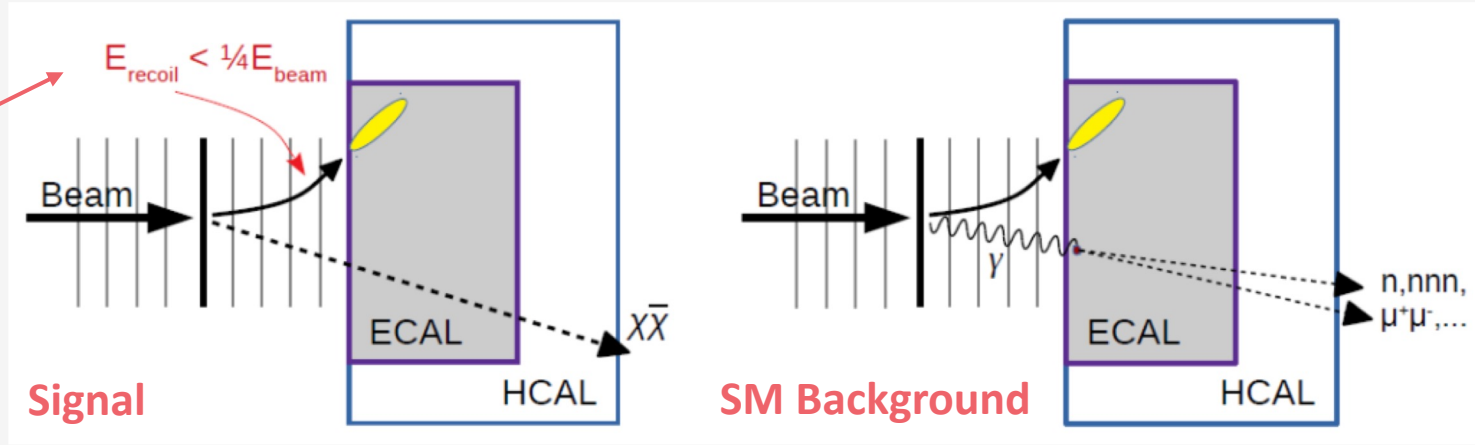
Two interaction vertices → production rate highly suppressed

- **Invisible decay**

One interaction vertex → interaction probability enhanced → **Better sensitivity!**

# Signal & Background Signatures

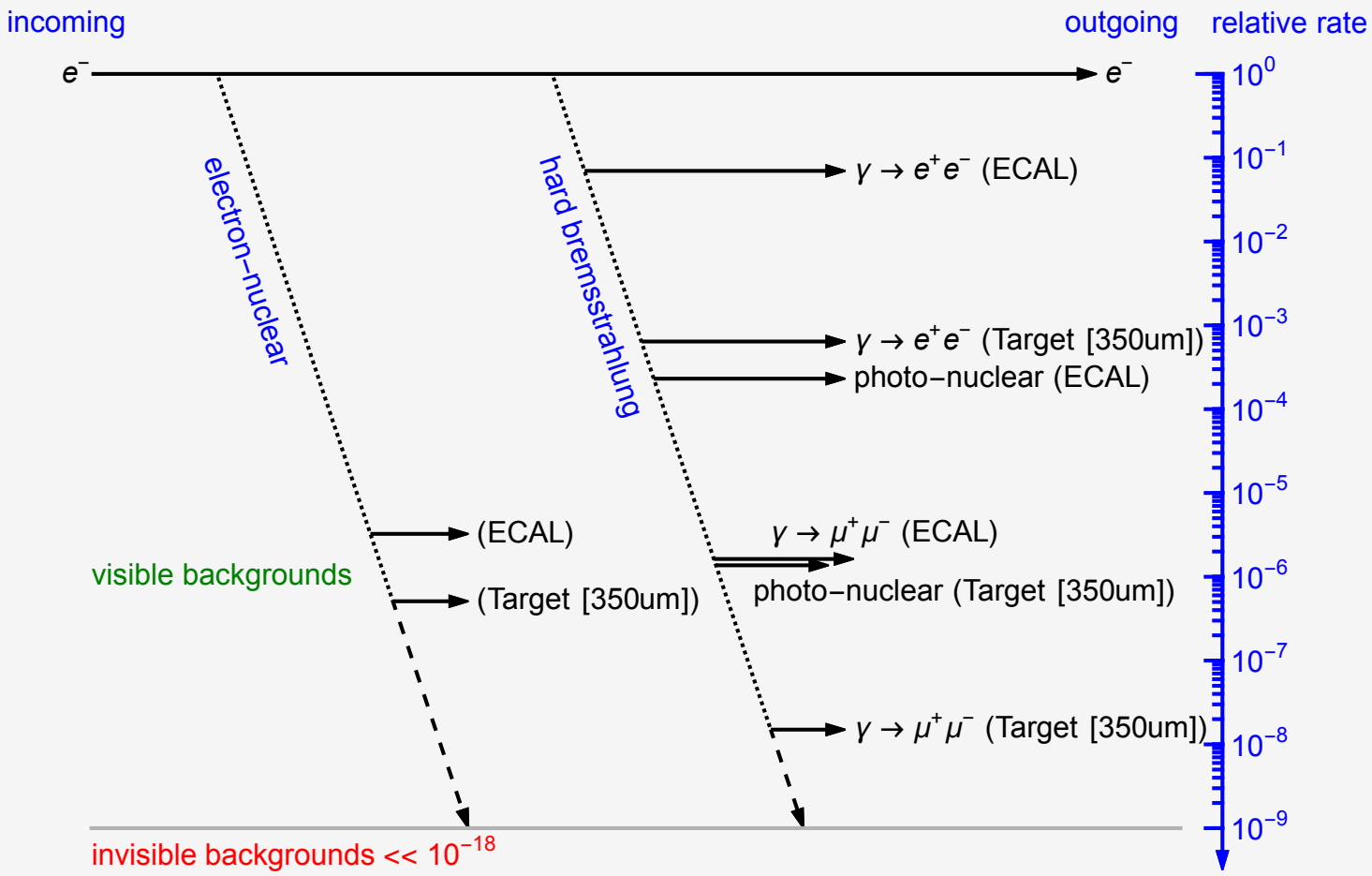



- Electron energy: 8 GeV, $\mathbf{3\times10^{14}}$ EOT per year (expected)
- Most of the incident momentum is transferred to $A'$.
- **Key signatures:** soft recoil electron, large missing energy & $p_T$.

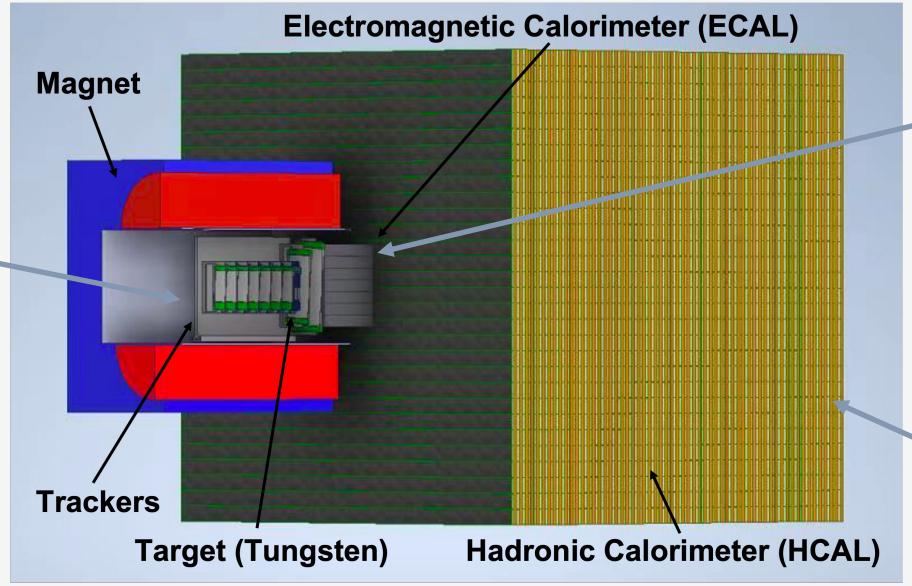
# Standard Model Backgrounds

- **Leading Background:**
  - Hard Bremsstrahlung
- **Rare SM processes:**
  - Photonuclear *(w. hard-brem $\gamma$)*
  - Electronuclear
  - $\gamma \rightarrow \mu\mu$ *(w. hard-brem $\gamma$)*
- **Invisible Background:**
  - $eN \rightarrow e\nu\bar{\nu}N$ (trident process)
  - Moller/Brem + CCQE



# Detector Conceptual Design

**Tracking system**
Measure the track of the incident and recoil electrons.




**Electromagnetic calorimeter**
Measure the deposited energy: electron and photon.

**Hadronic calorimeter**
Measure the deposited energy: muon and hadron backgrounds.

**Additional system:**
Readout electronics, trigger system, TDAQ, magnetic system (1.5 T), etc.

# DarkSHINE Event Display

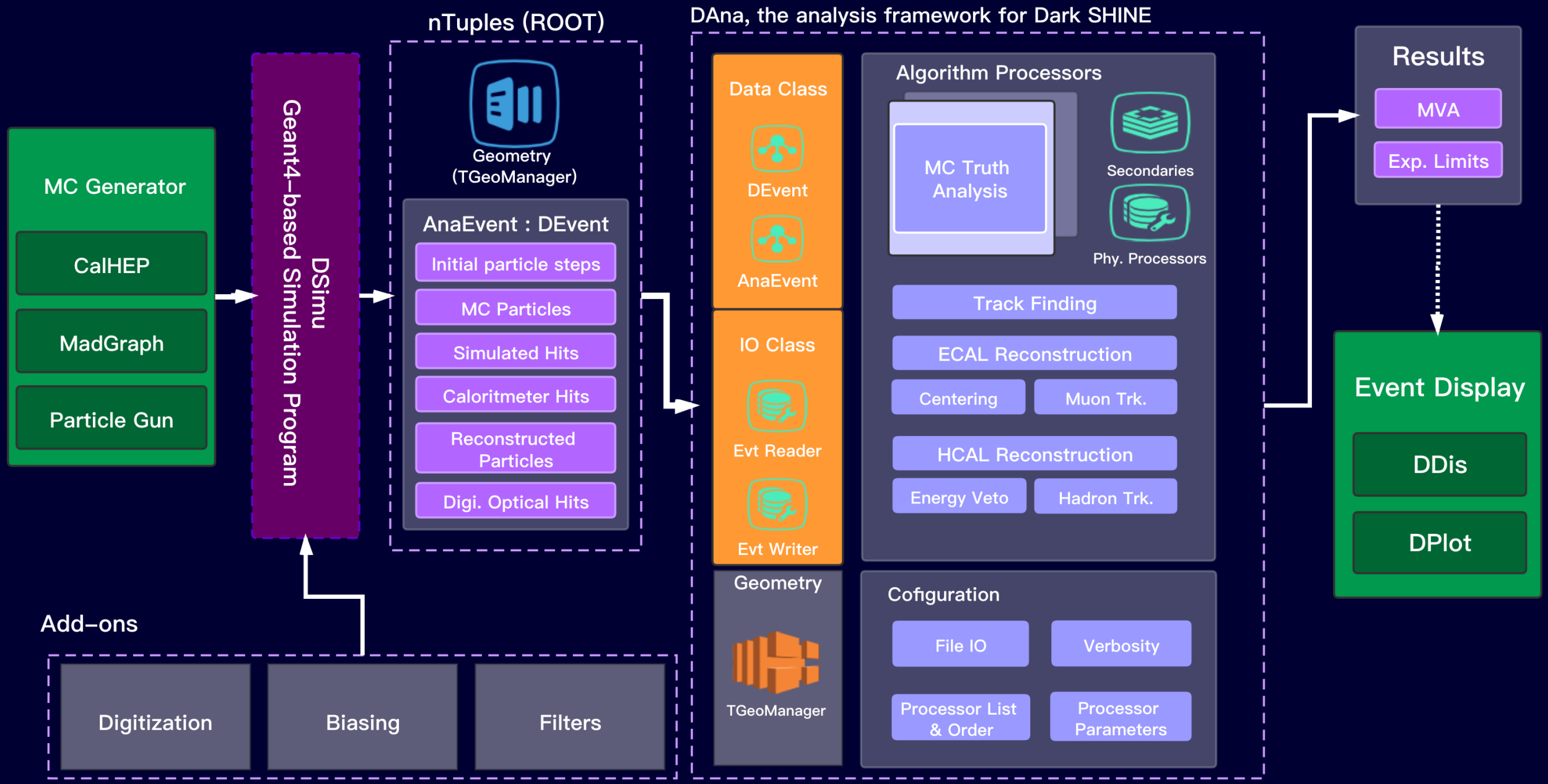
# Software Framework

DSS: DarkSHINE Software

# What is DarkSHINE Software

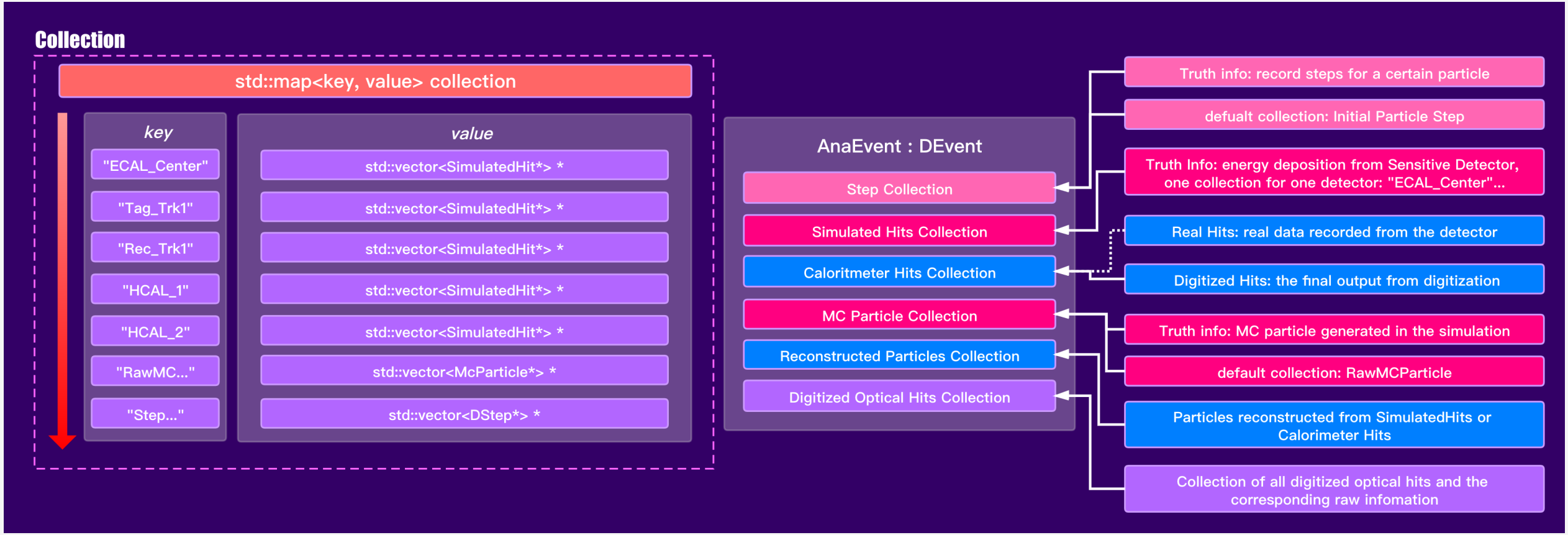
DarkSHINE Software is a software package, including five parts: **DSimu**, **DAna**, **DDis,** and **DPlot**.

- **DSimu** is the simulation program based on Geant4, characterized by Dark SHINE detector, controlled by yaml configuration.
- **DAna** is a *framework for the analysis and reconstruction tools*. It requires the output ROOT file (involving Geometry, DMagnet and DEvent) from **DSimu**.
- **DDis** is the event display for DSS. (requires Geometry and DEvent)
- **DPlot** is a quick plotting program for newbies and lazy man.
- **DEvent** is the *generic data structure* in DSS.


nTuples (ROOT)
DAna, the analysis framework for Dark SHINE
MC Generator
CalHEP
MadGraph
Particle Gun
DSimu
Geant4-based Simulation Program
Geometry
(TGeoManager)
AnaEvent : DEvent
Initial particle steps
MC Particles
Simulated Hits
Caloritmeter Hits
Reconstructed Particles
Digi. Optical Hits
Data Class
DEvent
AnaEvent
IO Class
Evt Reader
Evt Writer
Geometry
TGeoManager
Algorithm Processors
MC Truth Analysis
Secondaries
Phy. Processors
Track Finding
ECAL Reconstruction
Centering
Muon Trk.
HCAL Reconstruction
Energy Veto
Hadron Trk.
Cofiguration
File IO
Verbosity
Processor List & Order
Processor Parameters
Results
MVA
Exp. Limits
Event Display
DDis
DPlot
Add-ons
Digitization
Biasing
Filters

# DEvent

- Customized event data structure based on ROOT
- Optimized data structure & memory (*Memory alignment technique*: 60.1 KB/event → 13.7 KB/event)

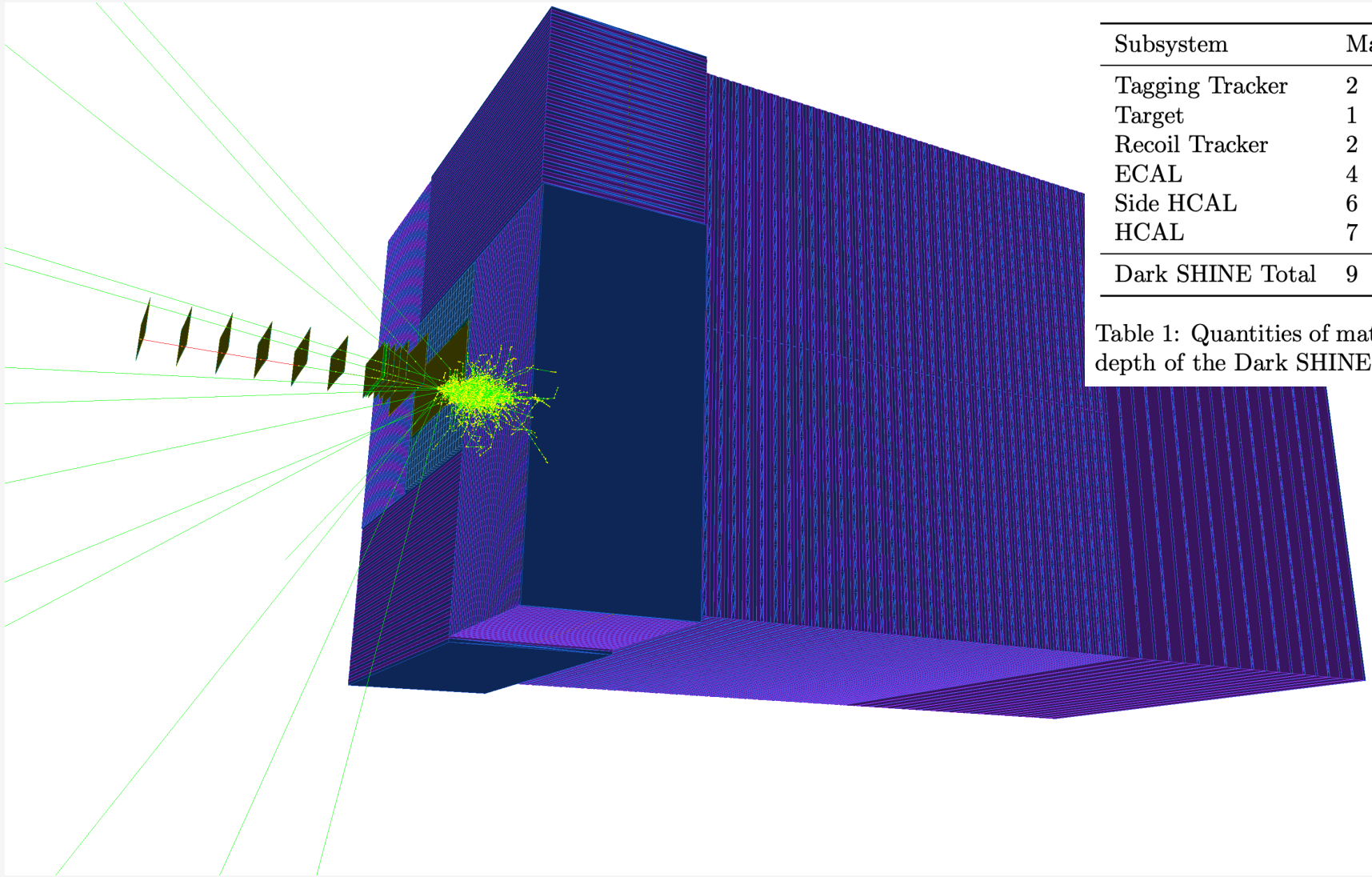


# DSimu

- Independently developed based on Geant4 with C++17 standard.
- Key Features:
  - **Multi-Threading supported**
  - **BSM signal process simulation**: integrated with MadGraph/CalHEP (Look-up table)
  - **Rare SM processes simulation**: Biasing technique for certain particle in certain logical volume
  - **Truth Filter**: filtering events of interest based on truth information on-the-fly
  - **Flexible Detector Construction**:
    - Parameterized placement
    - Imported from GDML
    - Build from configuration file
  - **Bounding Volume Hierarchy**
  - **DMagnet**: Non-uniform magnetic field (*imported from Mathematica*)
  - **Full Optical Physics Simulation**

*Credit: Xvliang ZHU*

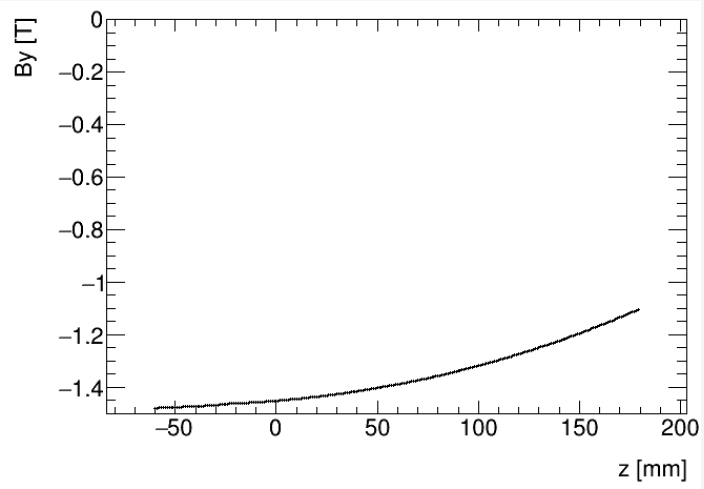
# Detector Construction Overview

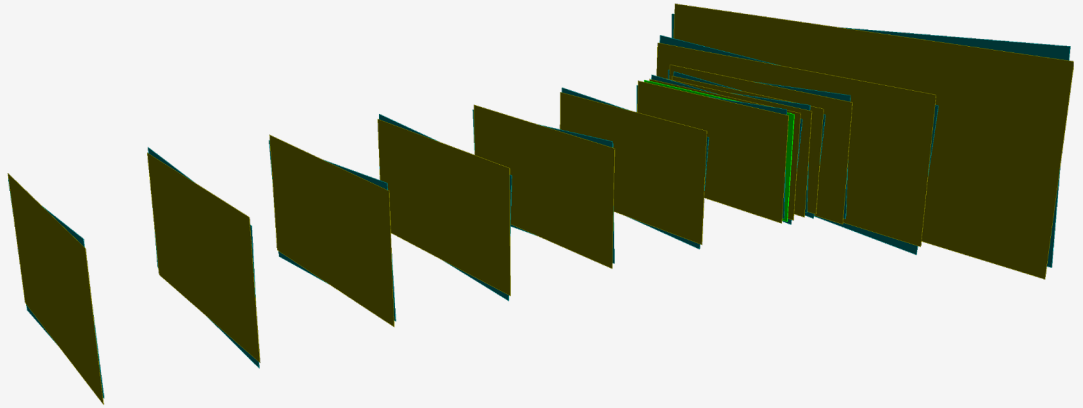


| Subsystem | Materials | Logical Vol. | Physical Vol. | Vol. Depth |
|---|---|---|---|---|
| Tagging Tracker | 2 | 43 | 94753 | 3 |
| Target | 1 | 1 | 1 | 0 |
| Recoil Tracker | 2 | 37 | 114749 | 3 |
| ECAL | 4 | 5 | 14653 | 3 |
| Side HCAL | 6 | 31 | 624 | 2 |
| HCAL | 7 | 96 | 27234 | 4 |
| Dark SHINE Total | 9 | 214 | 252015 | 5 |

Table 1: Quantities of materials, logical volumes, physical volumes, and volume depth of the Dark SHINE detector in simulation.

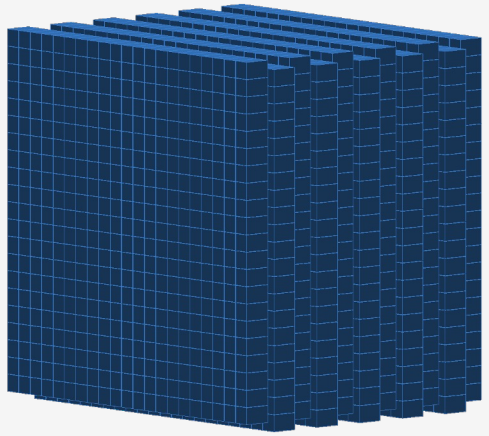
# Tracker Region



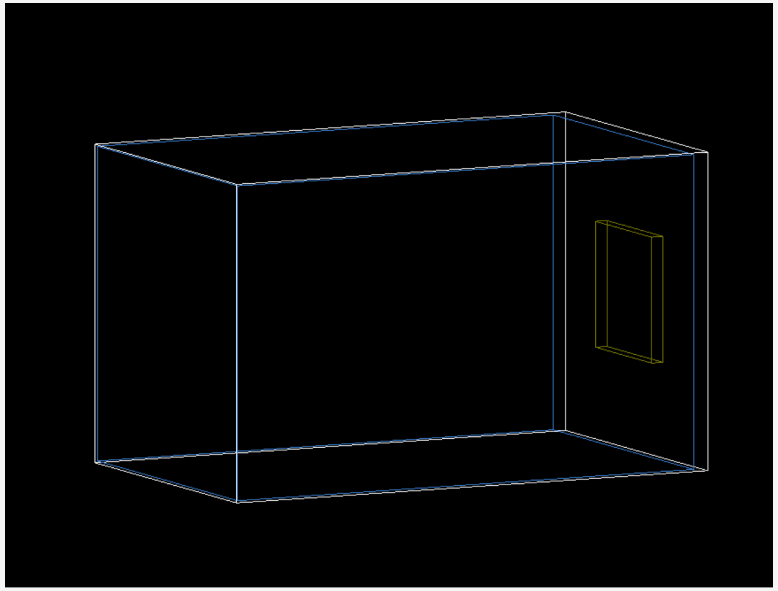



| | Material | Region Material | Center z | Size | Layer Number | Strip distance | Strip Number | Angle |
|---|---|---|---|---|---|---|---|---|
| **Target** | W | Vacuum | 0 | 20cm x 10 cm x 350 µm | 1 | - | - | - |
| **Tagging Tracker** | Si | Vacuum | -607.83 mm ~ -7.83 mm | 20.1 cm x 10 cm x **150 µm** | 7x2 | 30 µm | 6700 | ±0.05 radian |
| **Recoil Tracker** | Si | Vacuum | 7.73 mm ~ 180.23 mm | 20.1 ~ 50.1 cm x 10 ~ 20 cm x **150 µm** | 6x2 | 30 µm | 6700 ~ 16700 | ±0.05 radian |

# Staggered ECAL Region

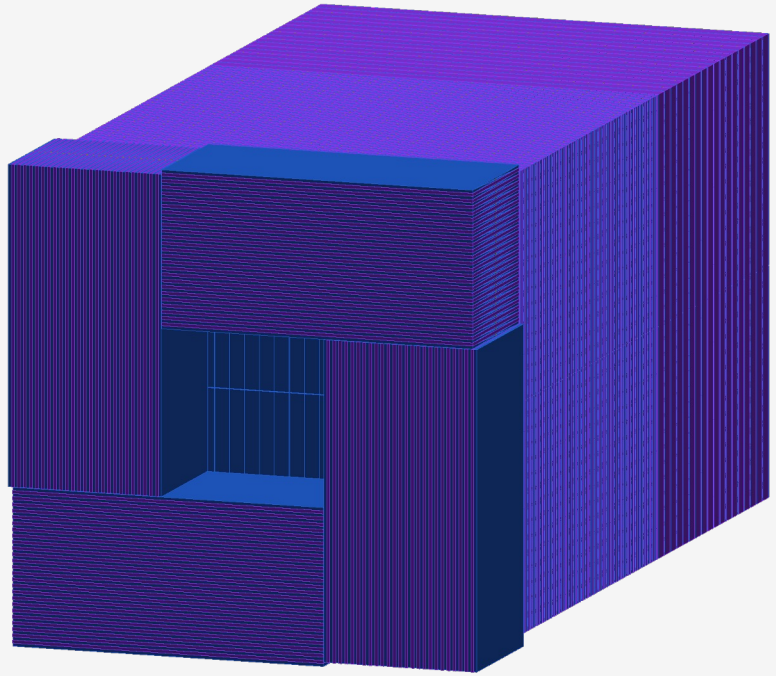


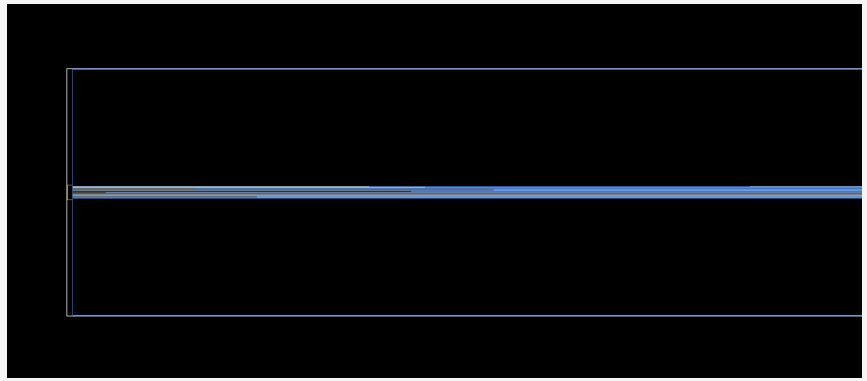
| | Material | Region Material | Center z | Cell Size | Cell Number | Cell Gap |
|---|---|---|---|---|---|---|
| **Wrapper** | C | Carbon-Fiber | 409.09 mm | 2.53 cm x 2.53 cm x 4.13 cm | 21 x 21 x 11 | 0.1 mm |
| **APD** | Si | | | 1 cm x 1 cm x 0.1 cm | | |
| **ECAL Crystal** | LYSO | | | 2.5 cm x 2.5 cm X 4.0 cm | | |



# HCAL Region

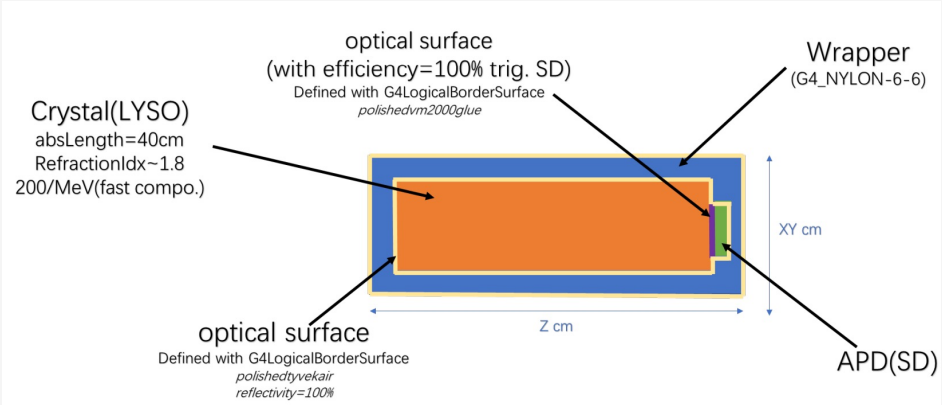
| | Material | Region Material | Center z | Cell Size | Region Size | Module Gap |
|---|---|---|---|---|---|---|
| **Wrapper** | C | Carbon-Fiber | 2703.1 mm | 1.03 cm x 5.03 cm x 75.55 cm | ~1.5 m x 1.5 m x 2.5 m | 0.5 mm |
| **APD** | Si | | | 3mm x 3mm x 1 mm | | |
| **HCAL Crystal** | Polystyrene | | | 1 cm x 5 cm x 75.42 cm | | |
| **SideHCAL Wrapper** | C | | 409.085 mm | 1.03 cm x 45.531 cm x 105.03 cm | ~1.5 m x 1.5 m x 0.45 m | |
| **SideHCAL Crystal** | Polystyrene | | | 1 cm x 45.511 cm x 105 cm | | |

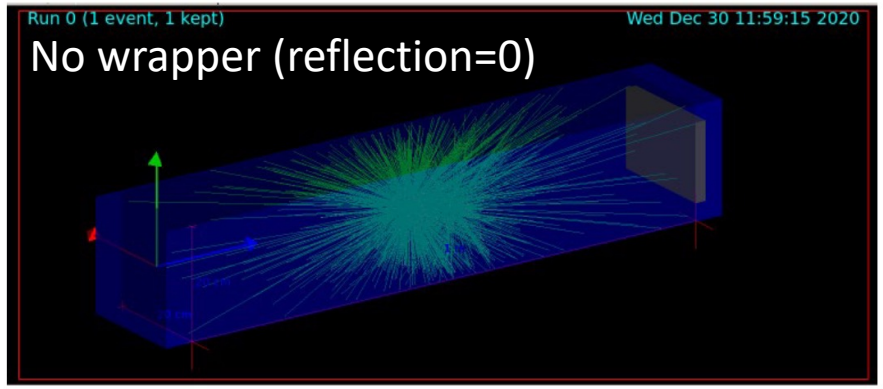




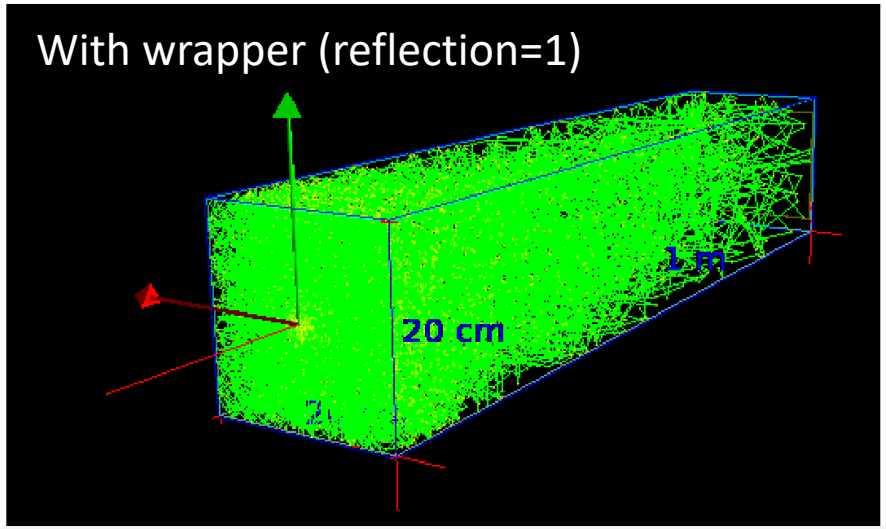
# Optical Physics Simulation

- Optical simulation implemented in each single unit of ECAL
  - ✓ Realistic parameters set-up and cross-check with measurement and datasets, including:
  - ➢ Crystal optical property (light yield, decay length, decay time)
  - ➢ SiPM digitization (dynamic range, noise)
  - ➢ Wrapper (reflection)
  - ➢ Optical grease (transparency)
- Parametrization method applied in full simulation
  - Smearing & Calibration effect implemented with parameters extracted from optical simulation
  - Generally compatible with full optical simulation → **much faster**!




- Demonstration only.
- Actual reflection set in simulation according to manufacturer (~0.98)




*Credit: Qibin LIU*

# Bounding Volume Hierarchy

- Detector Volumes are manually grouped as small sets and enclosed within larger bounding Volumes.
- Resulting in more efficient G4 Step Transportation.

General simulation speed ×6

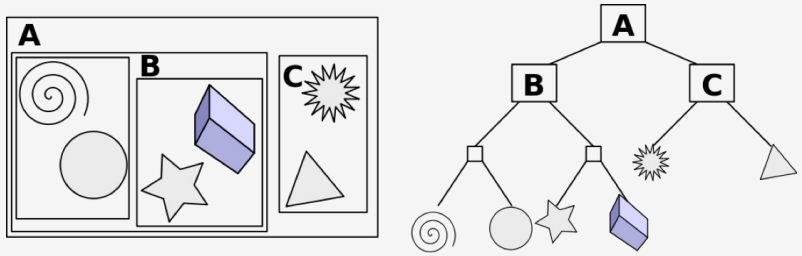
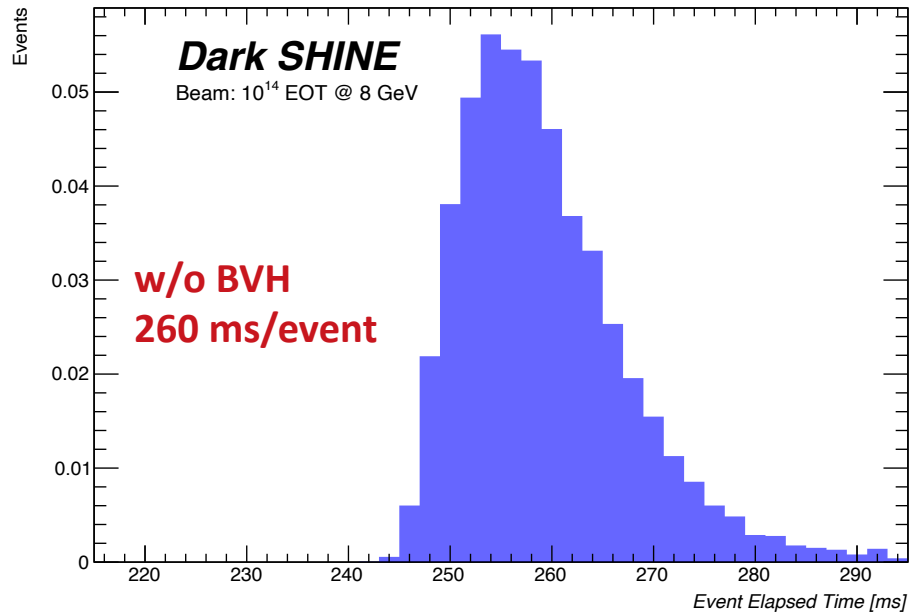
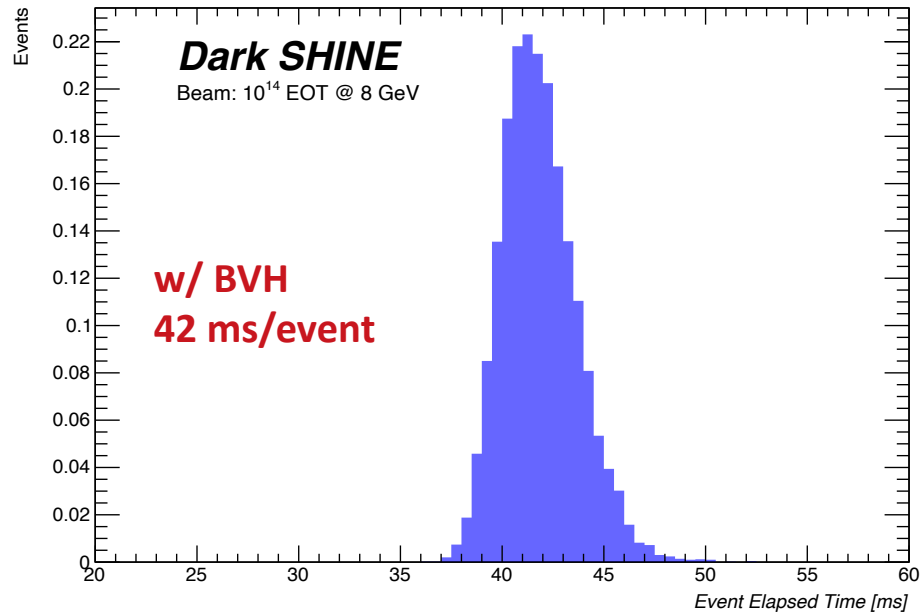
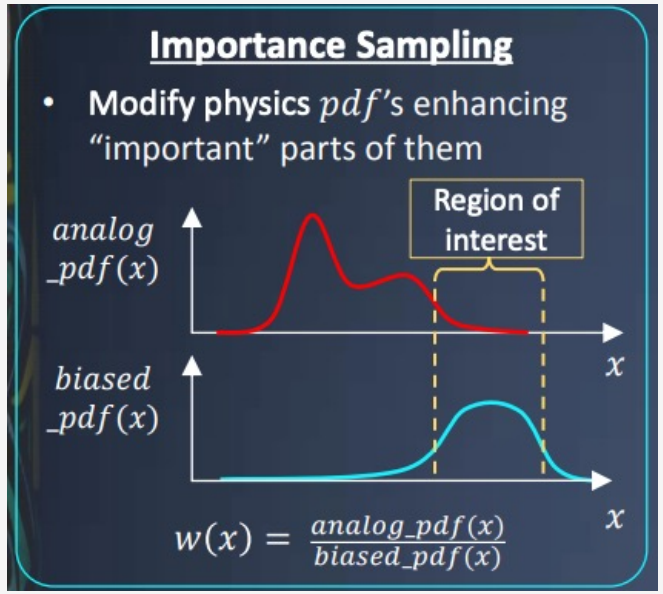



*Figure: An Illustration of the tree structure of bounding volume hierarchy (cite: wiki)*

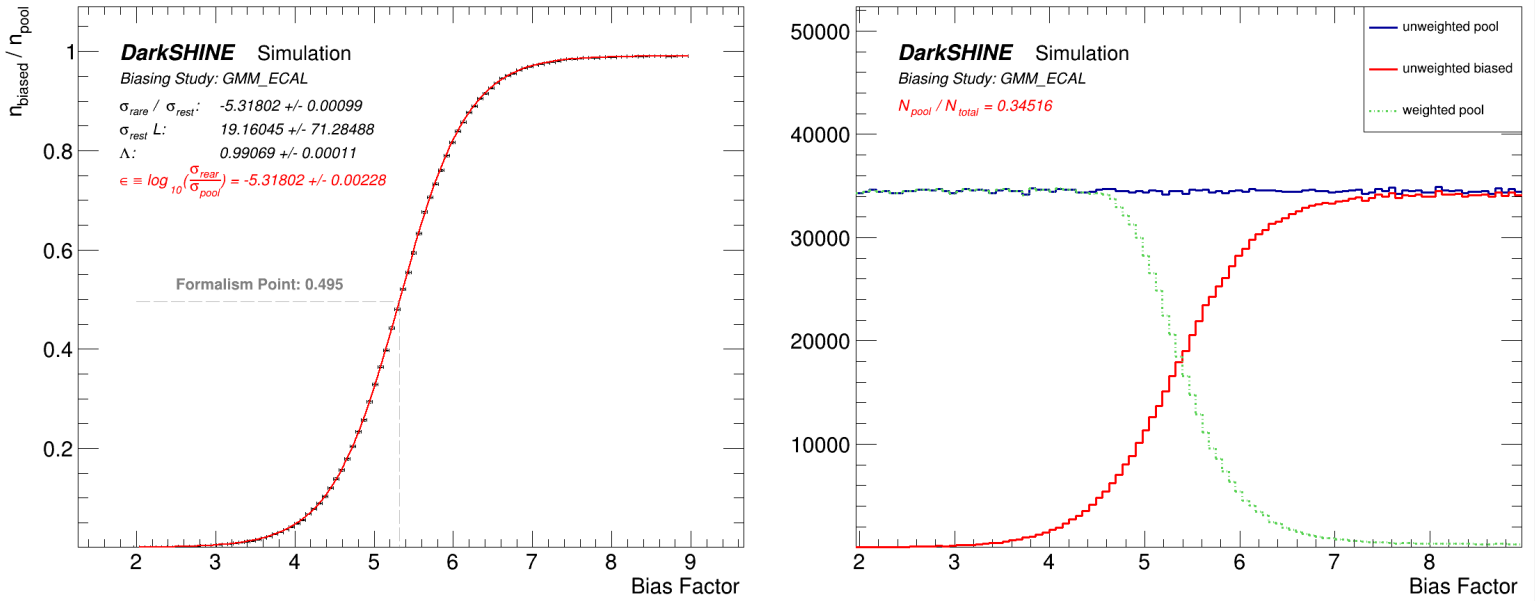


# Event Biasing




- Rare SM processes Biasing:
  - Through importance sampling to increase cross-section for a given process in the region of interest.
  - Apply a bias factor to certain process to reduce the interaction length.




- $\sigma_{biased}$ : cross-section of biased events
- $\sigma_{pool}$ : cross-section of interested events
- $\epsilon$ : analog ratio between biased and interested events
- Formalism point : ratio corresponded to the $\epsilon$

rel. to inclusive: $2.35\times10^{-8}$

# Truth Filter

## General Event Filter → General simulation speed $\times 15$

- Using Stepping Action and Stacking Action
- **Veto_ECAL**: **Abort** event immediately when truth ECAL deposit energy > 4 GeV
- **Veto_missP**: **Abort** event immediately when incident particle reach ECAL surface with truth $\Delta E < 4\ \mathrm{GeV}$

## Customized Event Filter

- Abort event if no particle/ process of interests is generated within certain energy interval or detector volume
- Filter on primary particle (for hard-brem $\gamma$)
- Filter on secondary particle (for rare processes from hard-brem $\gamma$)
- Together with event biasing → rare process simulation

# Sample Production

🏆 Target EOT: $10^{14}$

| Name | Process Branching Ratio | Biasing Factor | Filter Efficiency | Equivalent Event Number | Beam On Number | Estimate Time [16000 core hour] | Time per Event [ms] |
|---|---|---|---|---|---|---|---|
| Inclusive | 1.00E+00 | 1E+00 | 100% | - | - | - | 79.19 |
| **Inclusive w/ ECAL trigger** | 1.00E+00 | 1E+00 | 100% | **1E+11** | 1.00E+11 | **66.620** | 38.37 |
| **Inclusive w/ ECAL+missP trigger** | 1.00E+00 | 1E+00 | 100% | **1E+12** | 1.00E+12 | **90.592** | 5.22 |
| **GMM Target (with hardbrem) w/ ECAL+missP trigger** | 1.50E-08 | 1E+08 | 6.557% | 🏆 **1E+14** | 1.53E+07 | **0.001** | 3.17 |
| **GMM ECAL (with hardbrem) w/ ECAL+missP trigger** | 1.63E-06 | 1E+07 | 16.333% | 🏆 **1E+14** | 6.12E+07 | **0.005** | 4.47 |
| **PN Target (with hardbrem) w/ ECAL+missP trigger** | 1.37E-06 | 1E+06 | 6.466% | 🏆 **1E+14** | 1.55E+09 | **0.128** | 4.75 |
| **PN ECAL (with hardbrem) w/ ECAL+missP trigger** | 2.31E-04 | 1E+05 | 16.446% | 🏆 **1E+14** | 6.08E+09 | **0.737** | 6.98 |
| **EN Target (E > 4GeV) w/ ECAL+missP trigger** | 5.10E-07 | 1E+05 | 1.47% | 🏆 **1E+14** | 6.08E+10 | **1.646** | 1.39 |
| **EN ECAL (E > 4GeV) w/ ECAL+missP trigger** | 3.25E-06 | 1E+05 | 0.56% | 🏆 **1E+14** | 1.79E+11 | **1.025** | 0.33 |

# Multi-Threading for DSimu (MTDSimu)

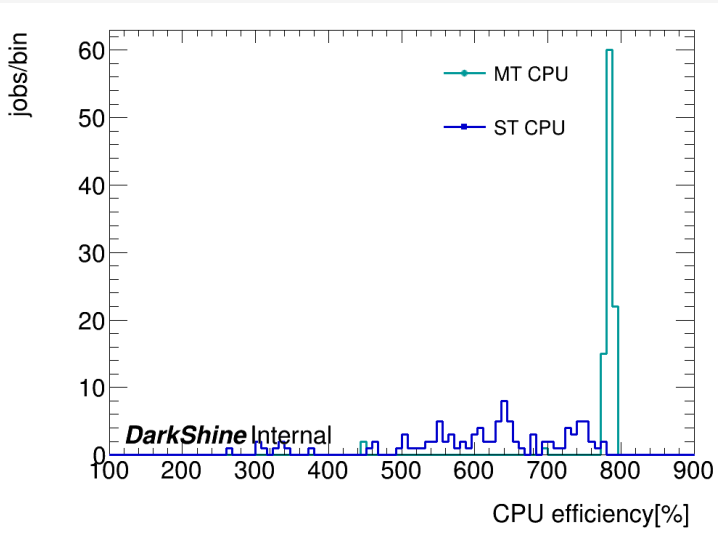
- Based on Geant4 Multi-Thread functionality (G4RunManager → G4MTRunManager)
  - Allocator/Singleton → ThreadLocal
  - Physics List, Random Number, Primary Generation Action
- Test $10^6$ events based on single-thread and multi-thread

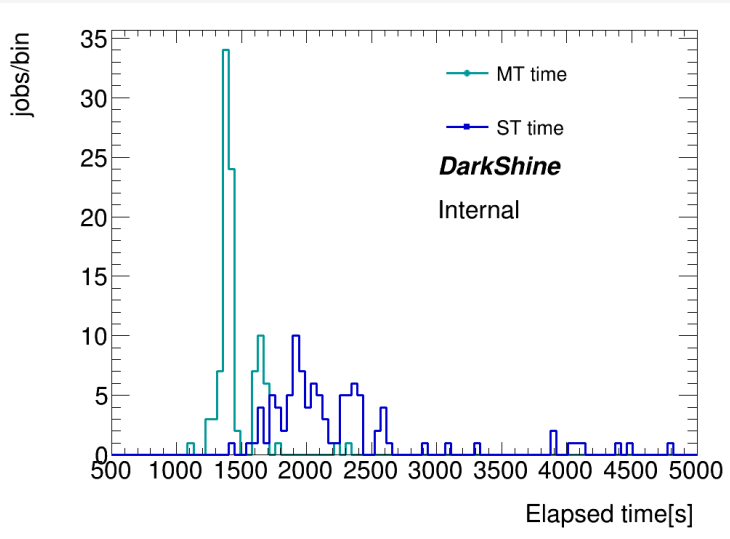
Time/Memory Validation

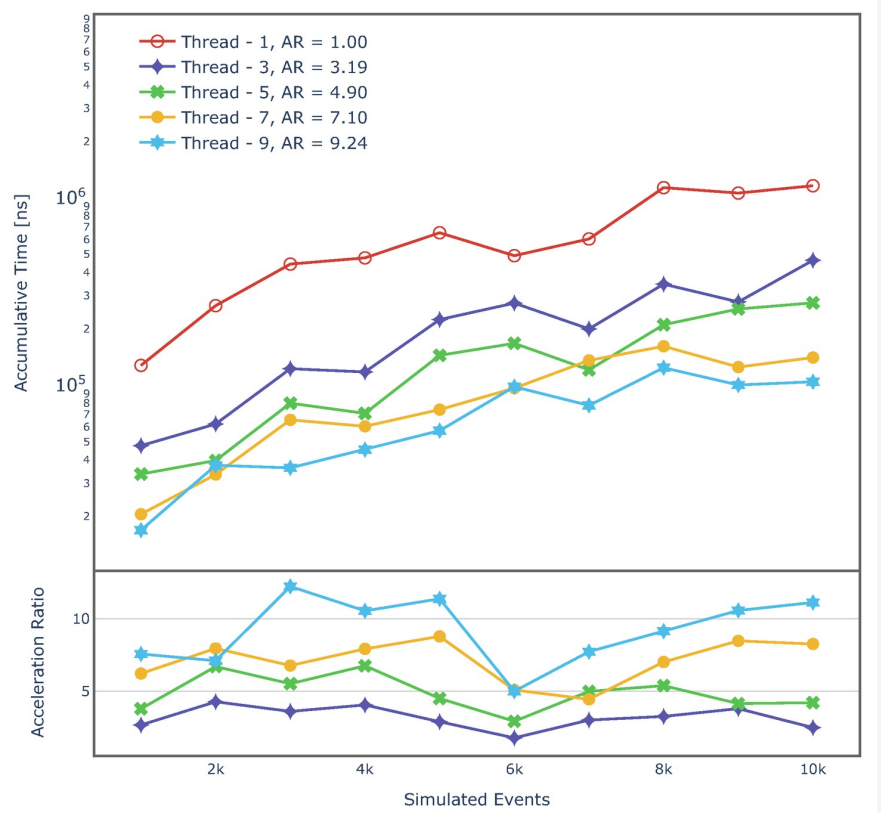
✅ Inclusive Process Validation

❑ Rare Process Validation

| Type | Total elapsed time [s] | CPU Usage | Money [¥] |
|---|---|---|---|
| 8 threads, 1 job | 15328 | 785% | 1.7 |
| 1 thread, 8 jobs | 16648 | 646% | 1.85 |

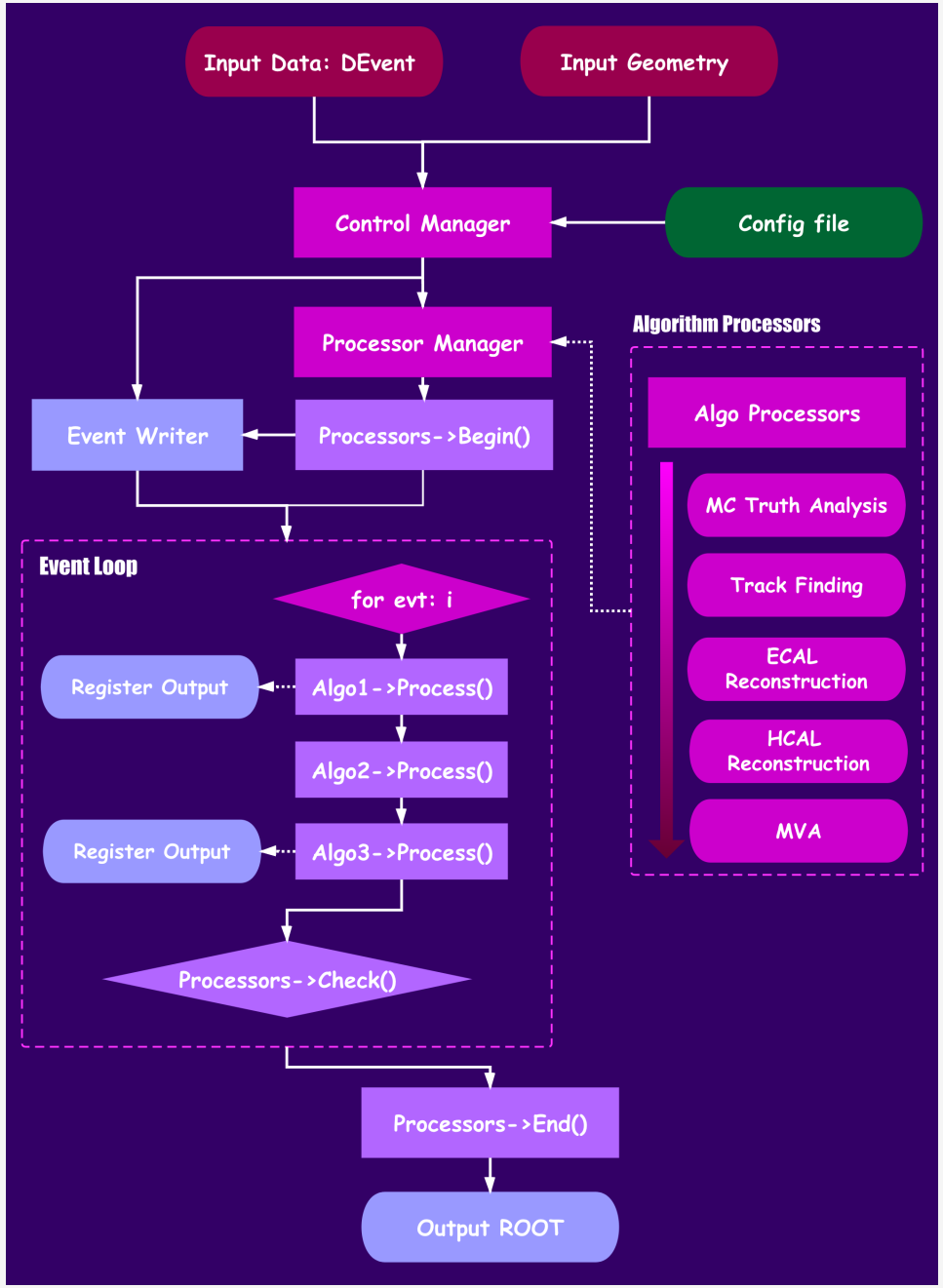


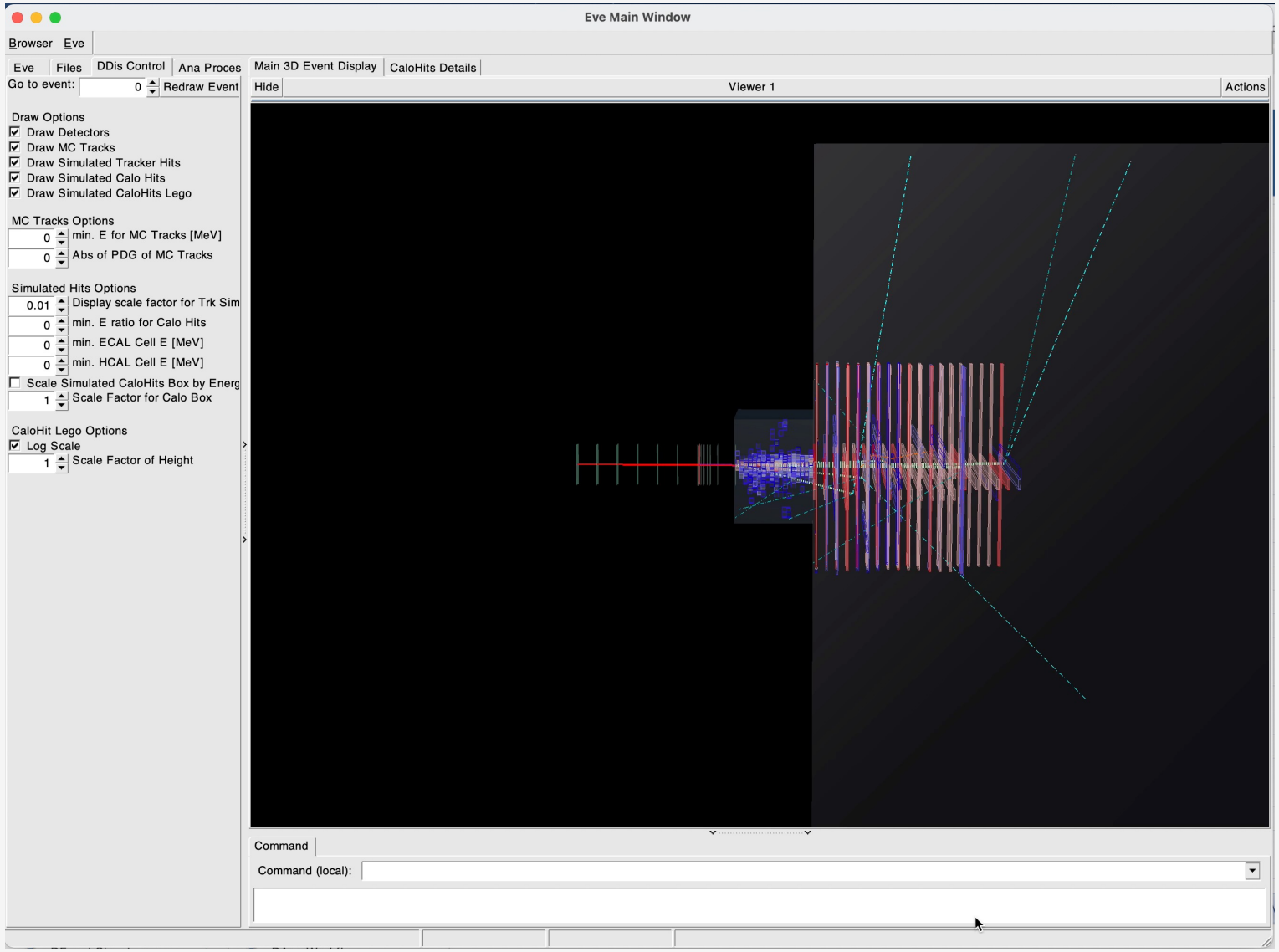
# DAna

- Reconstruction and Analysis Framework for DarkSHINE Software
- Algorithm Processors:
  - Called subsequently
  - Analyzed data can be shared within the event
- Featured processors:
  - MC Truth Analysis
  - Digitizer
  - Track Reconstruction
  - ECAL Reconstruction
  - HCAL Reconstruction
  - Data Exporter for Machine Learning
  - *Neural network integration (work in progress)*
    - *"Application of Graph Neural Networks in Dark Photon Search with Visible Decays at Future Beam Dump Experiment". Springer Nature (CCIS, EI), 2023*
  - *ACTS integration (work in progress)*



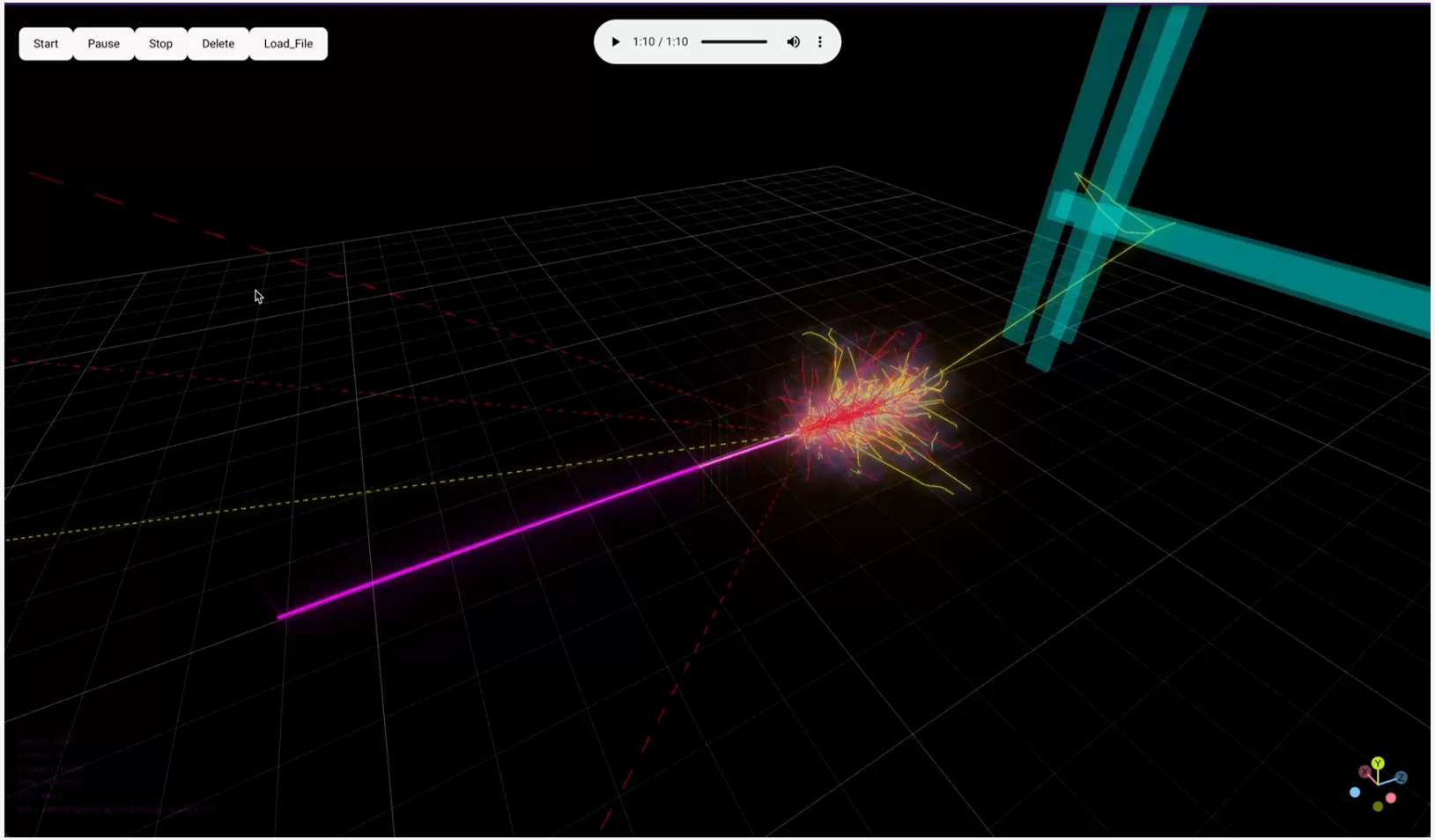
# Event Display

- Event Display for DarkSHINE Software based on QT and ROOT
- Read Geometry and Event Collection from output of **DSimu**
- Draw event one by one
- Customized draw options
- Support online algorithm processors



# Web-based Event Display

- New Event Display based on WEB
- Don’t need C++ & ROOT any more 😆
- Using three.js to render detector hits and MC particles



# What is a processor?

- Processor is a user-defined classes derived from the base class ***AnaProcessor***
  - *DP_ana/include/Core/AnaProcessor.h*
- 4 virtual functions (You need to implement the definition):
  - void Begin () → call at the **beginning** of the analysis program (DAna)
  - void ProcessEvt (AnaEvent *) → call for **each event** (a Event Loop)
  - void CheckEvt (AnaEvent *) → after all processors being called for certain event
  - void End () → call at the **end** of the analysis, when all the events are looped
- An event writer class to interact with output ROOT file
  - shared_ptr<EventStoreAndWriter> EvtWrt;
- The Data Structure used in DAna Algorithm: ***AnaEvent*** (***DEvent***)
  - *Utility/Object/include/Object/versions/DEvent_b1_5.h*

# How to write a processor?

- An example processor in *Algorithms/ExampleProcessor*
- Register your processor in *DP_ana/src/ControlManager.cpp* Line 82

```
1  // Processors
2  #include "Algo/ExampleProcessor.h"
3  #include "Algo/MCTruthAnalysis.h"
4  #include "Algo/RecECAL.h"
5  #include "Algo/RecHCAL.h"
6  #include "Algo/Digitizer.h"
7  #include "Algo/TrackingProcessor.h"
8  #include "Algo/CutFlowAnalysis.h"
9  #include "Algo/GNN_DataExporter.h"
10
11
12 /* Initialize and Select the AnaProcessors to use*/
13 /* Explicitly declare processors with name */
14 /* DEFINE ALGO PROCESSOR HERE */
15 algo->RegisterAnaProcessor(shared_ptr<GNN_DataExporter>(new GNN_DataExporter("GNN_DataExporter", EvtWrt)));
16 algo->RegisterAnaProcessor(shared_ptr<Digitizer>(new Digitizer("Digitizer", EvtWrt)));
17
18 #ifdef BUILD_HDF5
19 algo->RegisterAnaProcessor(shared_ptr<ECAL_ML_IO>(new ECAL_ML_IO("ECAL_ML_IO", EvtWrt)), false); // not add to default
20 #endif
21
22 algo->RegisterAnaProcessor(shared_ptr<MCTruthAnalysis>(new MCTruthAnalysis("MCTruthAnalysis", EvtWrt)));
23 algo->RegisterAnaProcessor(shared_ptr<TrackingProcessor>(new TrackingProcessor("Tracking", EvtWrt)));
24 algo->RegisterAnaProcessor(shared_ptr<RecECAL>(new RecECAL("RecECAL", EvtWrt)));
25 algo->RegisterAnaProcessor(shared_ptr<RecHCAL>(new RecHCAL("RecHCAL", EvtWrt)));
26 algo->RegisterAnaProcessor(shared_ptr<CutFlowAnalysis>(new CutFlowAnalysis("CutFlowAnalysis", EvtWrt)));
27
```

Remember to include the header file for your processor

# How to write a processor?

- Register parameters for this processor, in begin()
- Can modify some numbers without re-compilation, but define in runtime

```cpp
/*
 * How to register parameters?
 *
 * RegisterIntParamter -> register int type variables
 * RegisterIntParameter( "intVar", // variable name
 *                       "Int Variable", // variable description
 *                       &intVar,  // variable address
 *                       0);       // varaible default value
 *
 */

// Register Int parameter
RegisterIntParameter("intVar", "Int Variable", &intVar, 0);
RegisterIntParameter("Verbose", "Verbosity Variable", &verbose, 0);

// Register Double parameter
RegisterDoubleParameter("DoubleVar", "Double Var", &doubleVar, 0.);

// Register String Parameter
RegisterStringParameter("StrVar", "String Variable", &strVar, "test");
```

"*DAna -x*" → will show all current processors and their parameters

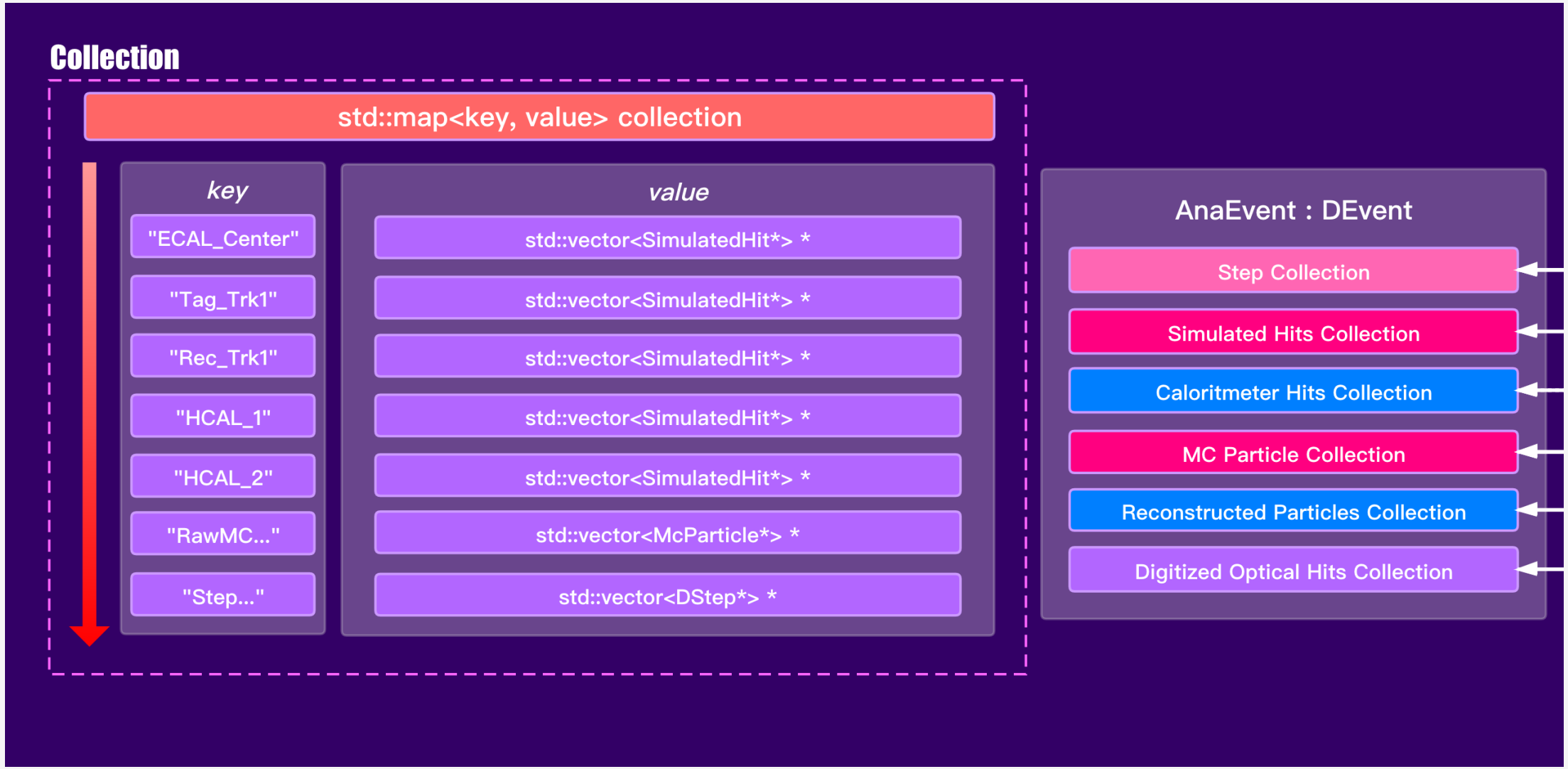
```
### Algorithm List
###
# CutFlowAnalysis: None
# Digitizer: Digitizer for Calorimeter(ECAL) with optical process
# ECAL_ML_IO: ECAL ML I/O for Calorimeter(ECAL)
# GNN_DataExporter: Export wanted data.
# MCTruthAnalysis: MC Truth Analysis
# RecECAL: ECAL Reconstruction Processor
# RecHCAL: Hadronic Calorimeter Reconstruction Processor
# Tracking: Tracking by Yi-Fan Zhu
###
Algorithm.List = GNN_DataExporter Digitizer MCTruthAnalysis Tracking RecECAL RecHCAL CutFlowAnalysis

### Algorithm Configuration


Digitizer.ApplytoECAL = 1  # Smearing on ECAL
Digitizer.ApplytoHCAL = 1  # Smearing on HCAL
Digitizer.RandomSeed = 0  # Random seed for smearing
Digitizer.pedestal = 0  # pedestal
Digitizer.rangeMax = 2048  # rangeMax
Digitizer.rangeMin = -2047  # rangeMin
Digitizer.Calibration_Factor = 1  # Calibration Factor
Digitizer.Nominal_Yield = 20000  # Nominal Yield of material, like 20000/MeV for LYSO
Digitizer.voltageToADC = 1.2207  # voltageToADC: fullRangeMV/ADCbits
Digitizer.CalibrationFile =   # YAML file for reference

ECAL_ML_IO.RW = 1  # 0(Read) in or 1(Write) out
ECAL_ML_IO.chunk = 1000  # how many event in the buffer
ECAL_ML_IO.ConditionValueExport = 8e+06  # The condition value which will be saved to the hdf5 in the conditionDS
ECAL_ML_IO.ThresoldExport = 0  # The thresold value used for export to hdf5
ECAL_ML_IO.ThresoldImport = 0  # The thresold value used for import to hit
ECAL_ML_IO.CollectionExport = ECAL_FS0  # Calorimeter (ECAL) Collection to Use for export
ECAL_ML_IO.CollectionImport = mECAL  # Calorimeter (ECAL) Collection to Import to
ECAL_ML_IO.ConditionBranch = cond_E  # The branch name of condition which will be saved to root
ECAL_ML_IO.ConditionDS = condition  # The dataset name of condition which will be read from/write to hdf5
ECAL_ML_IO.EnergyDS = energy  # The dataset name of Energy which will be write/read of hdf5
ECAL_ML_IO.Hdf5FileExport = export.h5  # Hdf5File to write out
ECAL_ML_IO.Hdf5FileImport = import.h5  # Hdf5File to read in
```

# How to write a processor?

- Register output variables in the output file with **DAna** (*dp_ana.root* by default)
  - *DP_ana/include/Event/EventStoreAndWriter.h*
- *EvtWrt* is from DAna and passing from Constructor, so just use it and don't need any other operation on it.

```
// void RegisterDoubleVariable(const std::string &VarName, double *address, const std::string &LeafType, bool active = true);
EvtWrt->RegisterDoubleVariable("Initial_Px", &Initial_Px, "Initial_Px/D");
EvtWrt->RegisterDoubleVariable("Initial_Py", &Initial_Py, "Initial_Py/D");
EvtWrt->RegisterDoubleVariable("Initial_Pz", &Initial_Pz, "Initial_Pz/D");
// void RegisterIntVariable(const std::string &VarName, int *address, const std::string &LeafType, bool active = true);
EvtWrt->RegisterIntVariable("ECAL_COL_SIZE", &ecal_col_size, "ECAL_COL_SIZE/I");
// template<class T>
// void RegisterOutVariable(const std::string &var_name, T *address, const std::string &var_help = "", const std::string &leaf_type
  = "", bool active = true)
EvtWrt->RegisterOutVariable("ECAL_E_total", &E_total, "[0] Truth total ECAL energy; [1-4] 4 sets of resolution smearing.");
EvtWrt->RegisterOutVariable("ECAL_ClusterSub_E" ,&ECAL_ClusterSub_E ,"Sub-Cluster Energy/MeV (used when tracker-match enabled)");
EvtWrt->RegisterOutVariable("ECAL_ClusterSub_X" ,&ECAL_ClusterSub_X ,"Sub-Cluster Center X/cm (used when tracker-match enabled)");
EvtWrt->RegisterOutVariable("ECAL_ClusterSub_Y" ,&ECAL_ClusterSub_Y ,"Sub-Cluster Center Y/cm (used when tracker-match enabled)");
```

# How to Access Data in Processor



# How to Access Data in Processor

- *Utility/Object/include/Object/versions/SimulatedHit_b1_5.h*
- *Utility/Object/include/Object/versions/McParticle_b1_5.h*
- *Utility/Object/include/Object/DTruth.h*

```
1 const auto &Collection = evt->getSimulatedHitCollection();
2
3 auto tag1_col = Collection.at("TagTrk1");
4 auto rec1_col = Collection.at("RecTrk1");
5 auto ecal_col = Collection.at("ECAL");
6
7 auto mc_particles = evt->getMcParticleCollection().at("RawMCParticle");
8
9 const auto &TruthCollection = evt->getTruthInfo();
```