http://www.geant4.org

EIC Geometry Exchange

Andrea Dotti (adotti@slac.stanford.edu) ; SD/EPP/Computing

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Introduction

NP/EIC simulations landscape:

EIC *framework* is not yet existing and technology choices are not yet made: **complete freedom to develop a comprehensive new system with innovative technologies**

Current HEP-NP frameworks/applications *do exist* and we want to:

- exchange information about geometry design
- use these frameworks to study initial EIC concepts
- minimize code changes

develop simple interface that can be integrated with these code

The challenge

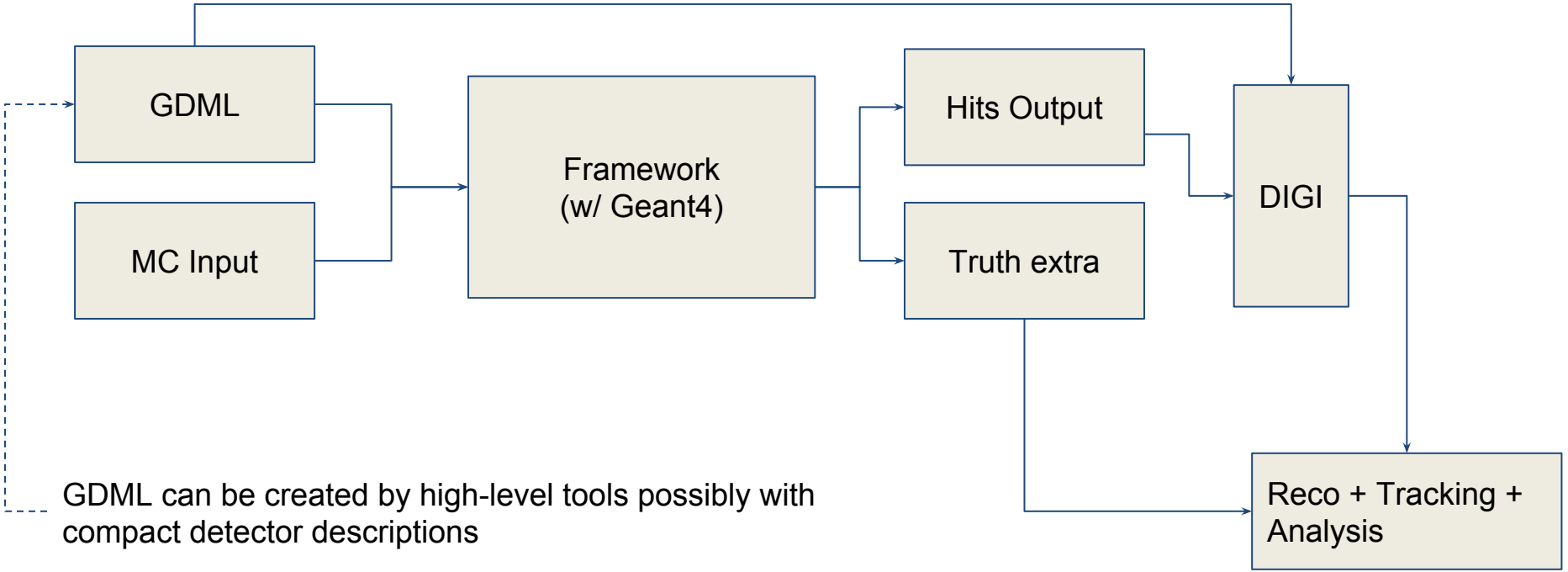
The only constraint we know is that Geant4 will be used at EIC for simulations

Focus is agree on **simple** and **universal** exchange format

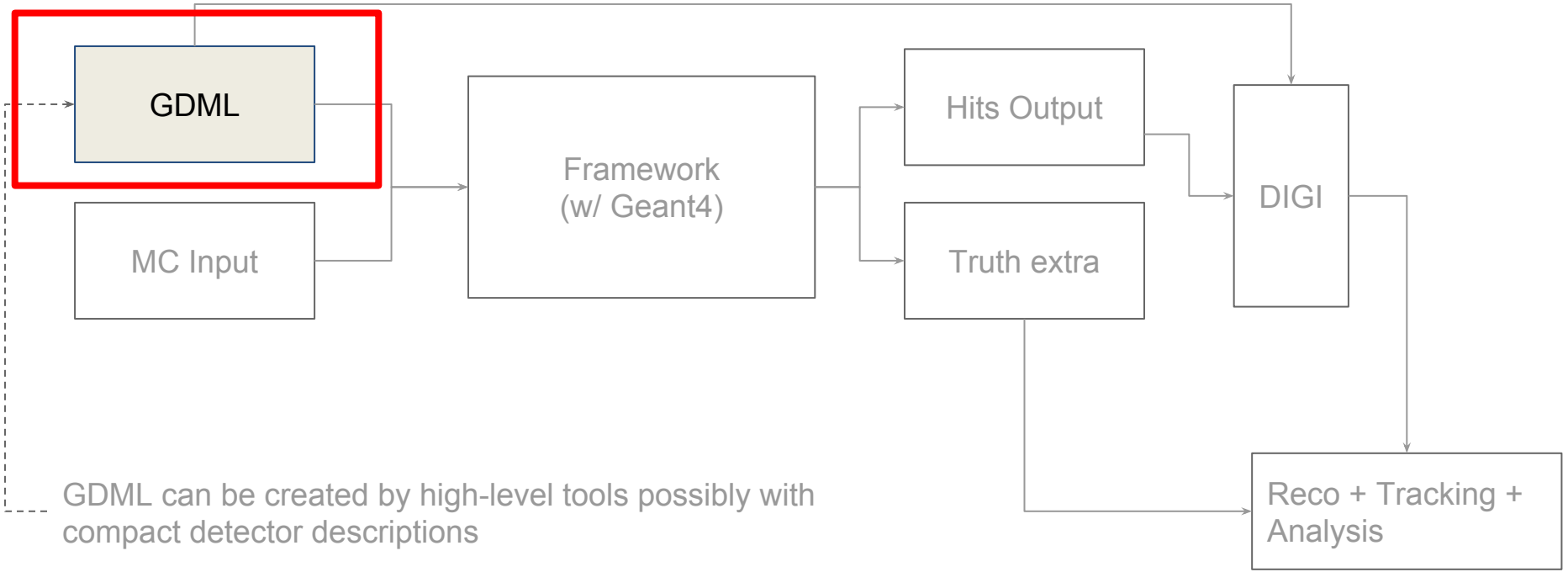
Fact: if we want all current existing frameworks/applications to participate we have two options, ROOT or GDML

Since GDML is considered the “universal” format understood by everybody, we start from there

Vision


GDML
MC Input
Framework
(w/ Geant4)
Hits Output
Truth extra
DIGI
Reco + Tracking +
Analysis
GDML can be created by high-level tools possibly with
compact detector descriptions

Vision


GDML
MC Input
Framework
(w/ Geant4)
Hits Output
Truth extra
DIGI
Reco + Tracking +
Analysis
GDML can be created by high-level tools possibly with
compact detector descriptions

GDML

GDML

A schema-based XML data format to describe the **geometry** of a typical HEP/NP detector

Shapes (e.g. CSG, tessellation, …) can be described together with: material association (down to atom/isotopes), relation between geometry components (*this volume is daughter of that*)

Maintained at CERN as a separate project (not part of G4, ROOT), see: https://gdml.web.cern.ch/GDML/

Geant4 and ROOT support GDML to write and read

- Geant4 ⇒ `G4GDMLParser` for both write and read
- ROOT ⇒ `TGDMLParse` to read and `writer.py` for TGeo conversion

GDML Limitations

It was developed to allow for automatic *dump* of C++-constructed geometry (e.g. from Geant4 application):

- some optimizations can not preserved: dump of geometry structure (*loop un-rolling*)
- for very complex/realistic geometries GDML becomes too complex to be edited by hand
- Hits/SD (see later)

The role of GDML for EIC

Your preferred framework/application should use the most appropriate way of defining geometry: extension of GDML, pure constructive, CAD, database.

You should only care to be able to export/import GDML to exchange information with other groups

If you use Geant4 you are covered

https://geant4.web.cern.ch/geant4/UserDocumentation/Doxygen/examples_doc/html/Examples_gdml.html

Writing GDML from Geant4

Given a Geant4 logical volume (e.g. the detector), from anywhere in your code:

```
{
    G4GDMLParser parser;
    parser.Write( gdml_file_name , logicalVolumePointer );
}
```

Reading GDML into Geant4

The usual detector construction is replaced/integrated with:

```
[...] MyDetector::Construct() {
    G4GDMLParser parser;
    parser.Read( gdml_file_name );
    return parser.GetWorldVolume();
}
```

Can be used to read only a component of the detector

Main limitation: hits and SD

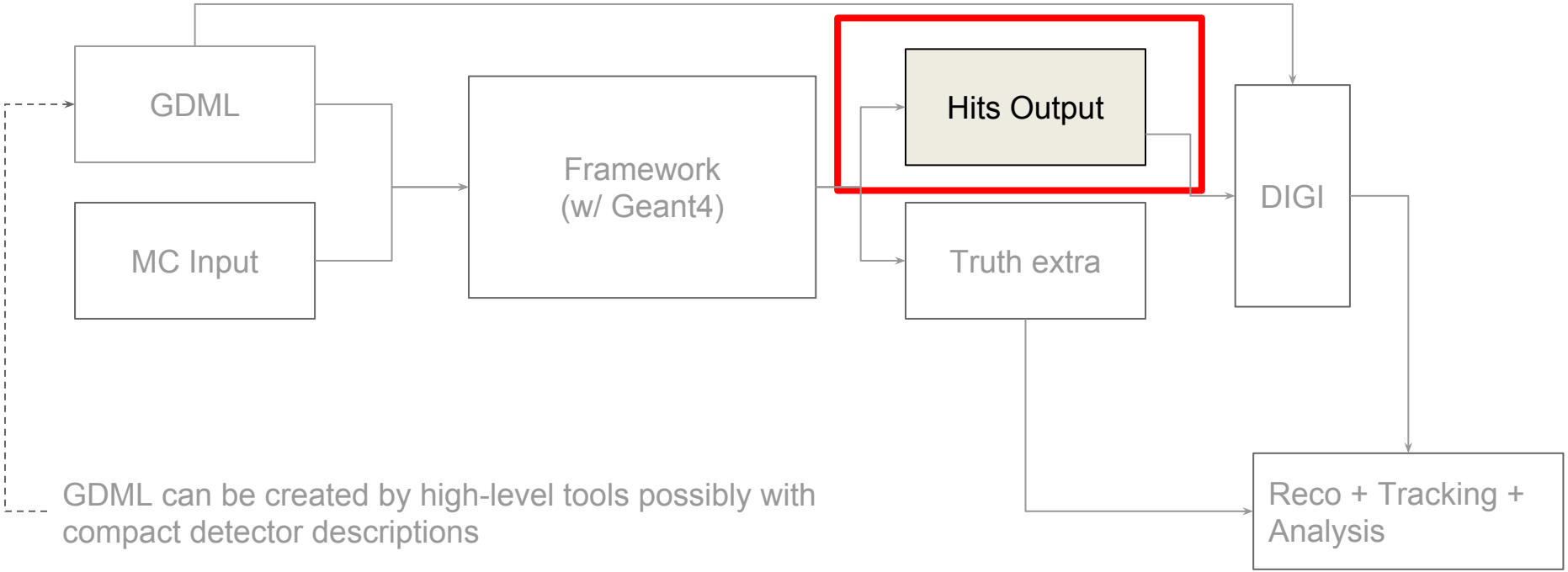
Describe geometry in terms of shapes, materials and (hierarchical) structure is not enough

Main issue is how to associate Sensitivity and **exchange this information between applications/detectors**

Start with GDML

Already available: examples/extended/persistency/gdml/G04 :
"Simple example showing how to associate detector sensitivity to a logical-volume, making use of the auxiliary-information."

Vision


GDML
MC Input
Framework
(w/ Geant4)
Hits Output
Truth extra
DIGI
Reco + Tracking +
Analysis
GDML can be created by high-level tools possibly with
compact detector descriptions

Hits

What a Geant4 hit is?

A G4Hit represents the energy deposit from a *step* in space and time

A G4Hit is **not** the response of the detector (e.g. pulse shape, digital output, response of a PMT)

- However often some low-level detector effects (Birks' saturation, some zero suppression) are included in hits for convenience

A Sensitive Detector is a Geant4 class to transform G4Step in a G4Hit

- With some detectors (e.g. calorimeters) it is impractical to transform each single G4Step to a hit (too many), thus accumulation is preferred (e.g. one hit per calorimeter cell, that accumulates all energy deposits)

Why GDML does not contain SD

Sensitive Detectors are experiment specific, they are **algorithms** and not data structure

What can be persistified in GDML is the **association** between a logical volume and the (named) corresponding SD. The actual SD code has to be provided via library

Proposal

Provide a lightweight library that depends only on Geant4:

1. that is distributed and maintained by the EIC Software Consortium
2. that can be used by any Geant4 existing framework (without requiring too many changes in user code)
3. that defines a minimalistic common data structure of hits
4. that provides storage of hits in ROOT files

What should be in a hit

ID	Int
Volume ID(s)	[]*sizeof(int) **OR []*string**
PreStepPoint {x,y,z,t} (both global/local)	8*sizeof(double)
PostStepPoint {x,y,z,t} (both global/local)	8*sizeof(double)
ΔE	1*sizeof(double)
PDG code	sizeof(long) //needed for ions
G4Track ID	1*sizeof(int)

SLAC slic developers will help on this!

What should be in a hit

ID	Int
Volume ID(s)	[]*sizeof(int) **OR []*string**
PreStepPoint {x,y,z,t} (both global/local)	8*sizeof(double)
PostStepPoint {x,y,z,t} (both global/local)	8*sizeof(double)
ΔE	1*sizeof(double)
PDG code	sizeof(long) //needed for ions
G4Track ID	1*sizeof(int)

Alternatively replace with median position and step length (some thoughts needed)

What should be in a hit

ID	Int
Volume ID(s)	[]*sizeof(int) **OR []*string**
PreStepPoint {x,y,z,t} (both global/local)	8*sizeof(double)
PostStepPoint {x,y,z,t} (both global/local)	8*sizeof(double)
ΔE	1*sizeof(double)
PDG code	sizeof(long) //needed for ions
G4Track ID	1*sizeof(int)

Identifies the track originating the hit: connection with *truth*

Some pseudo-code

```
class G4EicTrackHit : public G4VHit {
    //All usual G4 stuff
    virtual G4bool WriteHitToRootFile(){
        //Here there is the dependency on ROOT.
        //Write out a object that can be read on ROOT
        //and does not depend on G4
    }
    virtual G4bool WriteHitToAnotherFormat() {..}
}
```

What this will look like

GDML (see: examples/extended/persistency/gdml/G04):

```
<structure>
  <volume name="Boxvol" >
   <materialref ref="Air" />
   <solidref ref="Box" />
   <auxiliary auxtype="SD:EICG4SD:Tracker" auxvalue="MyTracker"/>
  </volume>
```

Translates to: compile and link your application against *libEICG4SD.so* that contains the *Tracker* Sensitive Detector object, create an instance named *MyTracker*

Note: if you do not link against library, the auxiliary tag will be ignored

Simple solution based on **naming convention** that does not require changes in GDML nor Geant4

How to modify existing user code

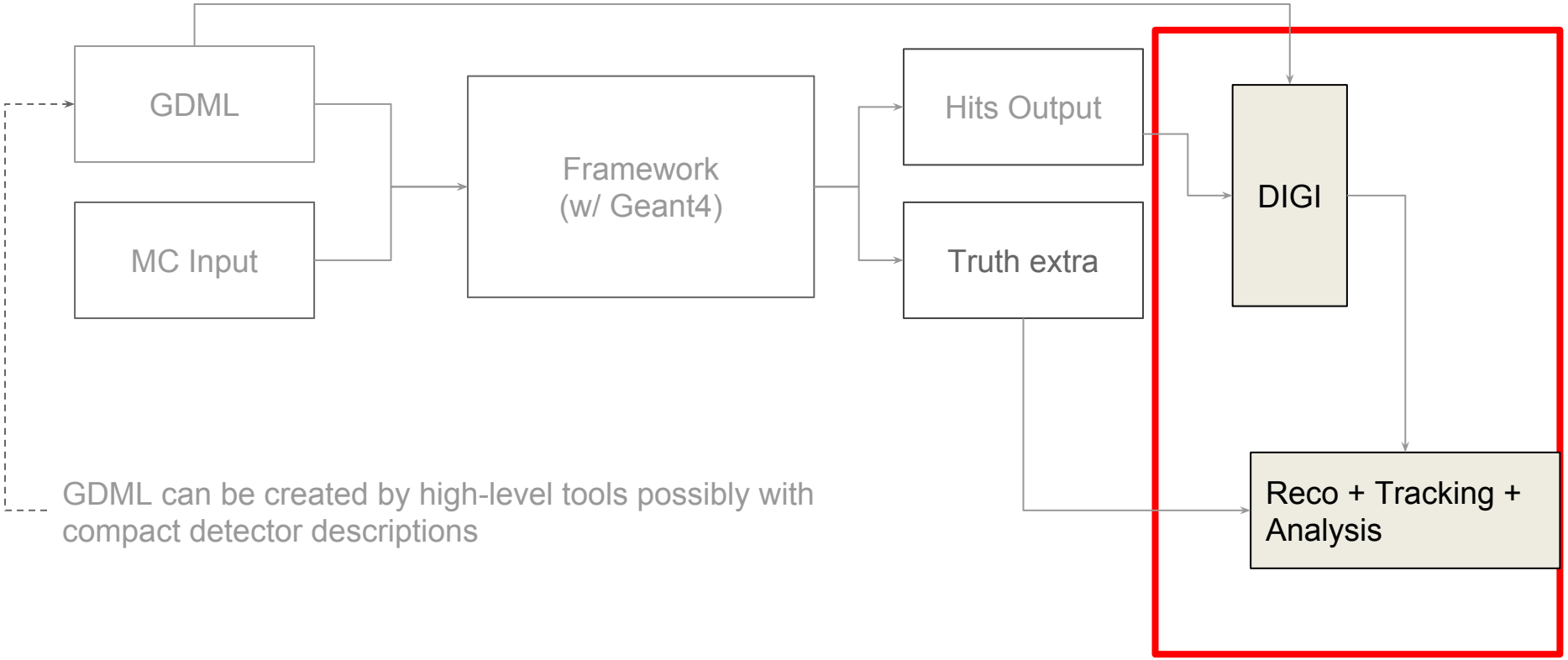
Starting from Geant4 version 10.3 it is possible to have multiple Sensitive Detectors associated to any given Logical Volume

EICG4SD functionality can be added to existing software in *parasitic mode*, existing frameworks and applications will continue to produce the existing hits files

Possible API (only an idea):

```
[...] MyDetector::Construct() {
    /… Usual stuff
    EICG4SDUtility.HandleSD( gdml_file_name , DetLVPtr );
    return fWorld;
}
```

Vision


GDML
MC Input
Framework
(w/ Geant4)
Hits Output
Truth extra
DIGI
Reco + Tracking +
Analysis
GDML can be created by high-level tools possibly with
compact detector descriptions

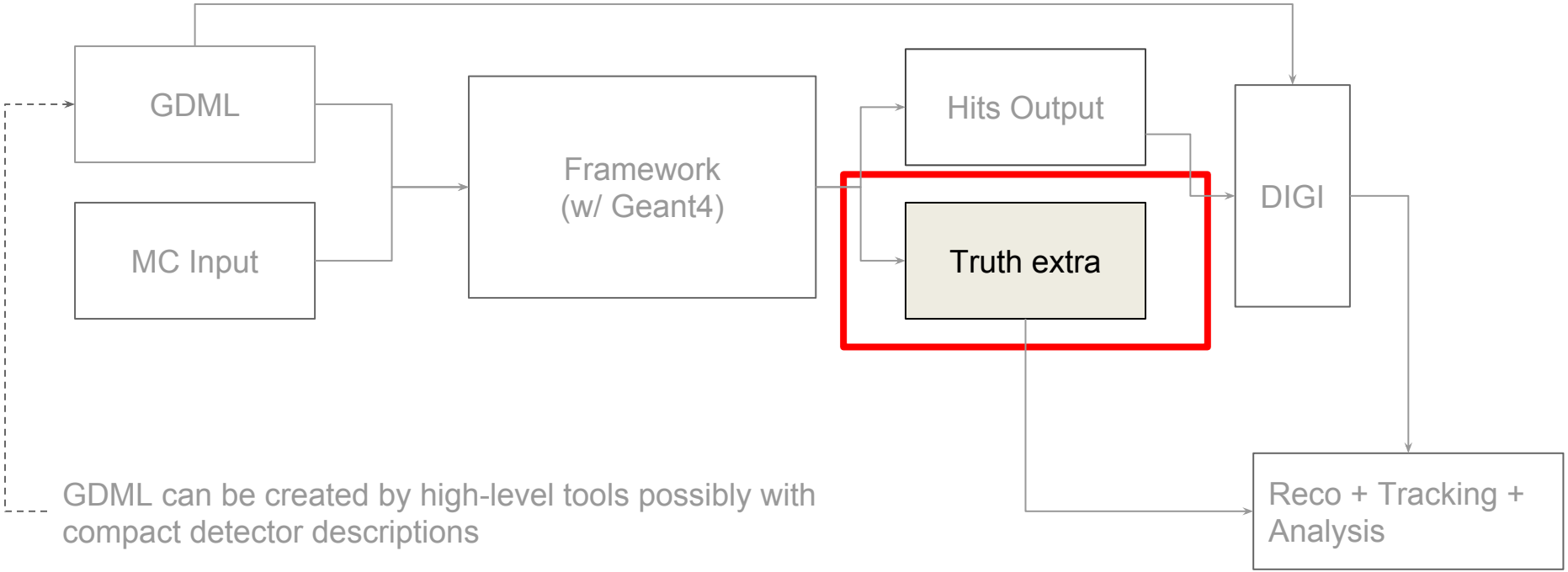
Reading hits

With ROOT files, it *should be trivial*[*] to read back hits once their data-layout has been decided

A separate library can be created for this with no dependence on Geant4

[*]: Not a ROOT guy, no direct experience

Vision


GDML
MC Input
Framework
(w/ Geant4)
Hits Output
Truth extra
DIGI
Reco + Tracking +
Analysis
GDML can be created by high-level tools possibly with
compact detector descriptions

MCTruth

Connection with MCTruth

What we need: mapping between the Geant4 (primary) track and the generator (e.g. PYTHIA) track id

Provided with a separate library: starting from EICROOT and Fun4All equivalent functionalities

Conclusions

Deliverable:
C++ library (in ESC gitlab) to produce hits file in ROOT format to be used by any Geant4-based system

Time-scale:
End of 2018

Manpower:
SLAC personnel on a best effort basis ; +Chris and Alexander for MCTruth integration)

This is clearly a oversimplified and not-optimized approach, but the objective is to have an simple, easy-to-maintain proof-of-principle

Project satisfies:

Primary goals:

1. no dependency, must be integrated in existing frameworks/applications, with minimal disruptions
2. exchange of information between existing frameworks/applications
3. allow to reason about more realistic system for future EIC framework
4. provide simulation independent hits for Tracking algorithm studies

Not a goal:

1. develop a new complete, extendable, performant component of a framework