Towards a LUXE EDM

Federico Meloni (DESY)

LUXE technical meeting
12/05/2021

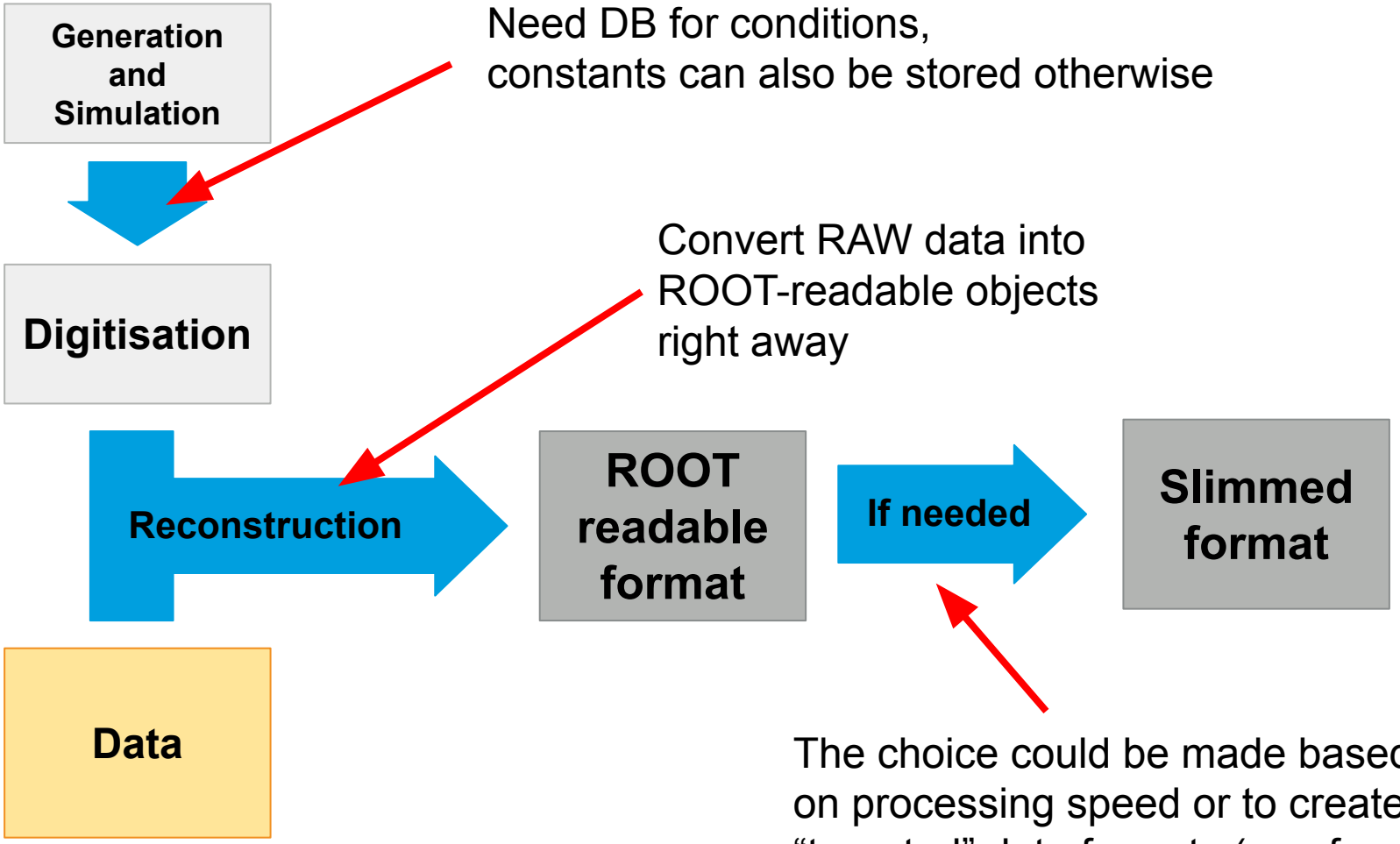
The Event Data Model

A collection of classes, interfaces and concrete types, and their relationship that, together, provide a representation of an event and eases its manipulation by the reconstruction and analysis developers.

The Event Data Model (EDM):

- creates a **commonality** across the detector **subsystems**.
- allows the use of **common software** between online and offline environments.
- defines the **structure of the data at various stages** and allows elements of the data flow processing tasks to access the data without resorting to the use of other resources, e.g. databases.
- defines **additional metadata** that is added to the detector data allowing processing tasks to quickly identify the type and origin of each event.

What about LUXE?



What about LUXE?

Adopt many ideas also present in ATLAS' xAODs (see talk).

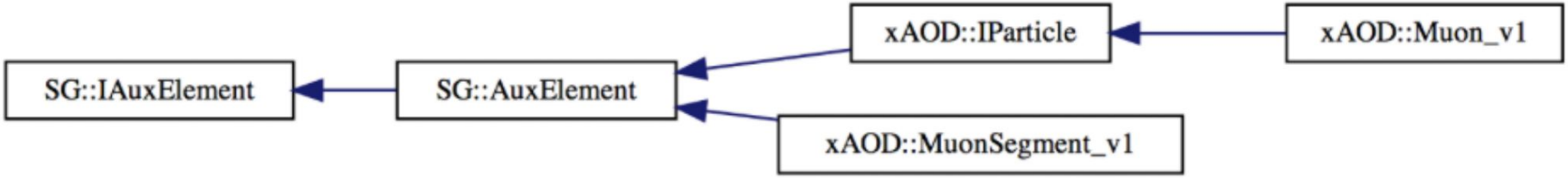
- **ROOT readable**, with two possible options:
  - Loading of EDM classes for advanced use (links, etc)
  - Fully flat ROOT-only is in principle also possible but quickly becomes very messy (interfaces, functions, ...)

- Implementation of classes as **interface + auxiliary store**
  - Keep access to variables standardised
  - Avoid complex object oriented EDM classes
  - Simple class inheritance structure

- Structure **information in containers**
  - New containers can be added at runtime.
  - New variables can be added to an object at runtime.

Reminder: xAOD class inheritance

The class that all xAOD interface classes inherit from is SG::AuxElement

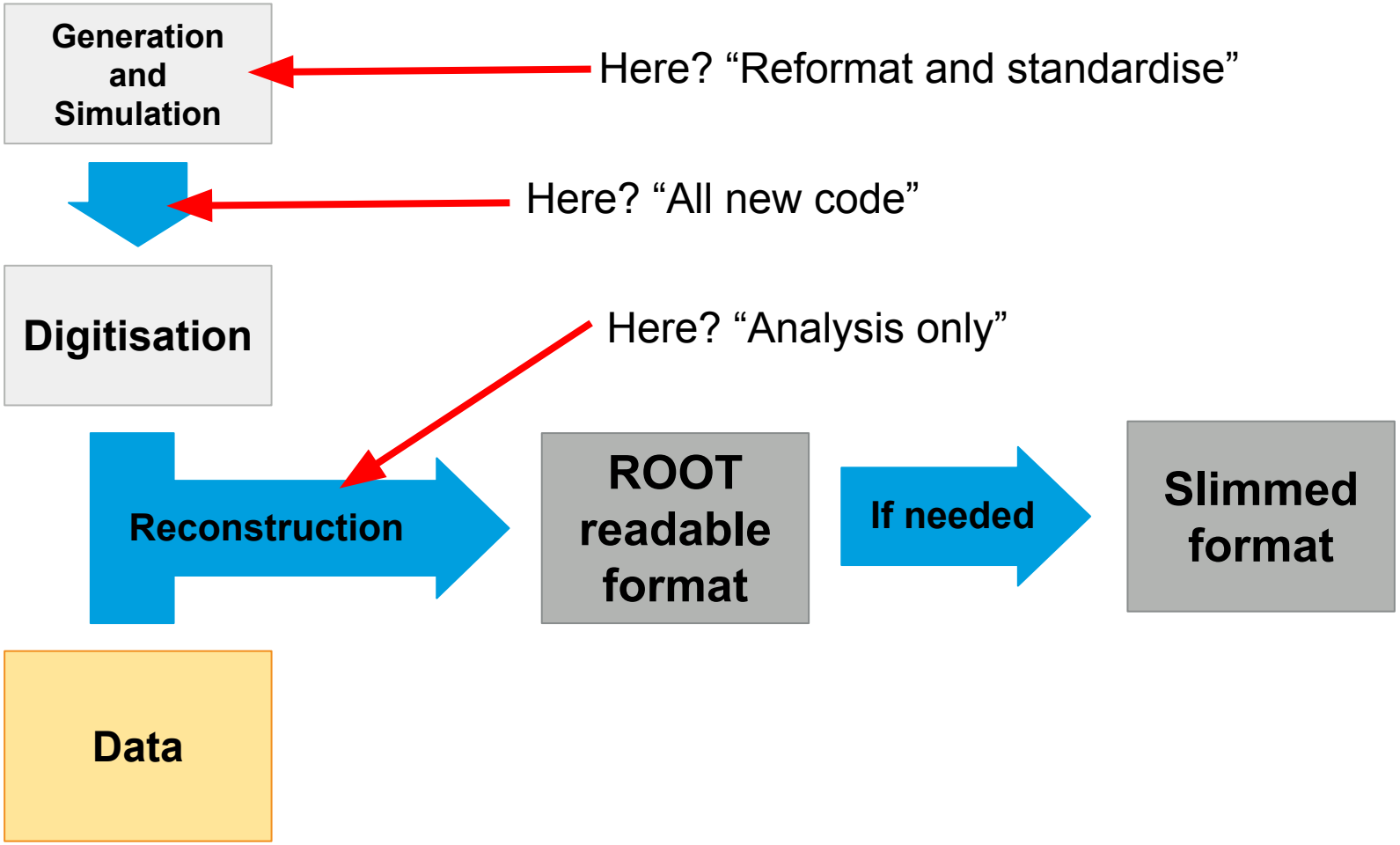


It's this class that provides the core infrastructure for the xAOD EDM implementation.

- Provides **standardised access** to all variables stored in the object's auxiliary store
- Allows the user to add **new variables to an object at runtime**

Setting up an equivalent is the natural first step in the implementation of LUXE's EDM.

One EDM for everything?



What classes do we need?

- Event information (per bunch crossing?)
- TruthParticles
- TruthVertices
- Tracker hits
- Calorimeter hits
- Scintillation hits
- Cherenkov hits
- Tracker clusters
- Calorimeter clusters
- Tracks
- Vertices?
- Reconstructed particles

Build on top of key4hep/ILCsoft instead?

Thoughts

- Save some work: it exists (certainly ILCsoft works too, key4hep it's not clear to me) from start to finish
  - We have a custom generator format and dedicated sim package
- No direct ROOT readability, but ntuplers and python-based ~intuitive libraries exist
- Cherenkov and scintillators will need to be implemented
- Interface with conditions DB could be harder / need a lot of forks