HepMC mini-Workshop
21st January 2014

LHCb HepMC usage and requirements

HepMC as persistent generator event record

Use of HepMC in the LHCb simulation framework

EvtGen & HepMC

Related requirements + wishes

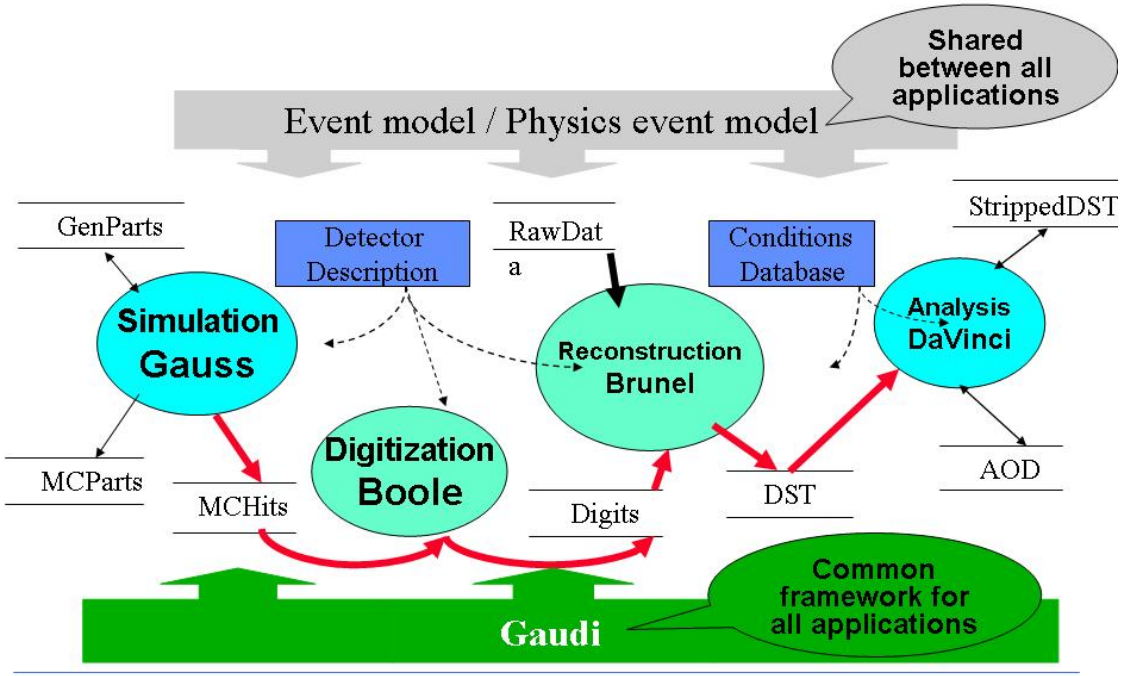
Gloria Corti, CERN

on behalf of the LHCb Simulation Core Developers

Pre-amble

- All LHCb event processing applications are based on LHCb common software, in particular a common LHCb Event model
  - What affects the event model may have consequences on any application accessing the particular classes
  - Event model classes that have been saved in (MC) samples used for physics analysis must be readable in the future (i.e. Data Preservation)

Data produced is used/studied directly or in further processing (digitization, reconstruction,…)

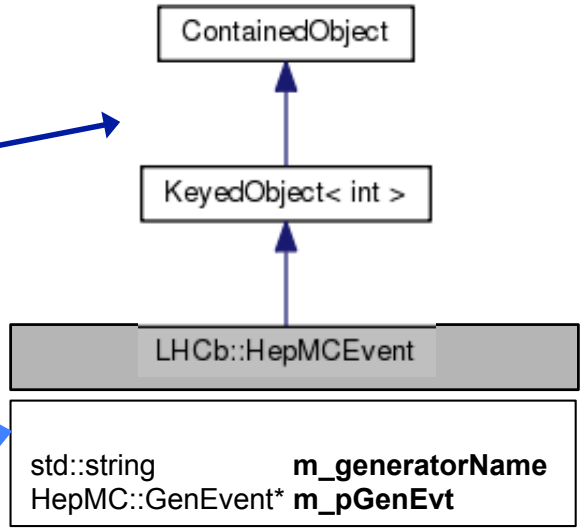


HepMC as event generator record

- Generator level events are stored in HepMC format within an LHCb wrapper that provides general LHCb data object functionality

Access to Transient Event Model and persistency (i.e. from memory and from files)

Handling of vector of objects



One LHCb::HepMCEvent per pileup (in time collisions)

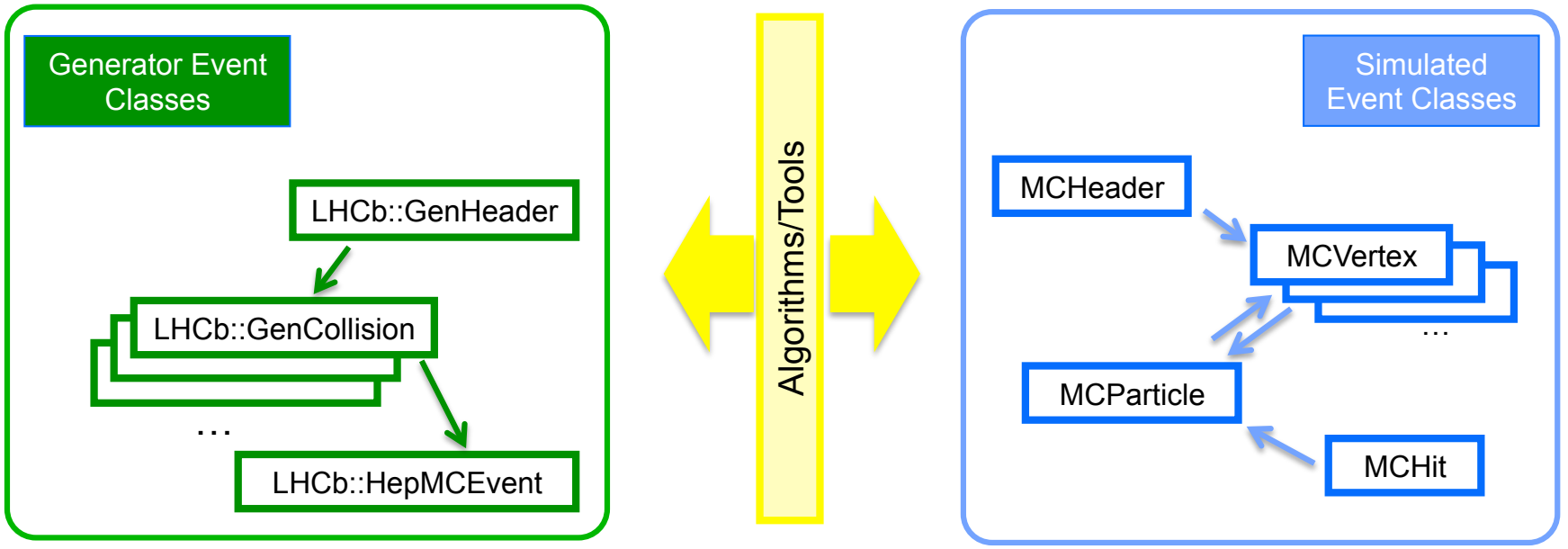
Link to one HepMC::GenEvent from other LHCb event data classes as a whole

- The persistency is taken care via Gaudi and ROOT dictionaries
  - Issue when moved from HepMC 1.26.02 to HepMC 2.0.
    - ROOT developed a Class Streamer making use of a HepMC 2.0 dictionary capable to read HepMC 1 events
  - Dictionary also essential for python-based analysis

HepMC LHCb specificities

- Particles have status code (HepMC::GenParticle::status()) which have special meanings.
  - 1 = stable in production generator (p from Primary Vertex, ...)
  - 2 = decayed/fragmentated by production generator (quark, ...)
  - 3 = (Pythia) documentation particle (string, ...)
  - 777, 888 = decayed by EvtGen (all unstable particles)
  - 889 = signal particle
  - 999 = stable in EvtGen (p from B decays, ...)
- Units are LHCb units (MeV, mm, and ns).
- HepMC::GenEvent::signal_process_vertex() is the decay vertex of the signal particle.
- General Information at the collision level kept in an LHCb class, GenCollision
  - Mandelstam and Bjorken-x variables, Process type and flag to indicate collision w signal
  - Some overlap with more recent HepMC::PDFInfo
  - Would like to store a “universal” process ID common to “all” generators

LHCb MC Event Model



- History of particles traveling trough detector in dedicated LHCb event data classes **MCParticles and MCVertices** and their relationship, a.k.a. **MC Truth**
- MCParticles/MCVertices contain also the generator level information (decay trees of all hadrons), and only a limited part of the hard process information (heavy quarks, W&Z but not strings, …).

HepMC in persistent files

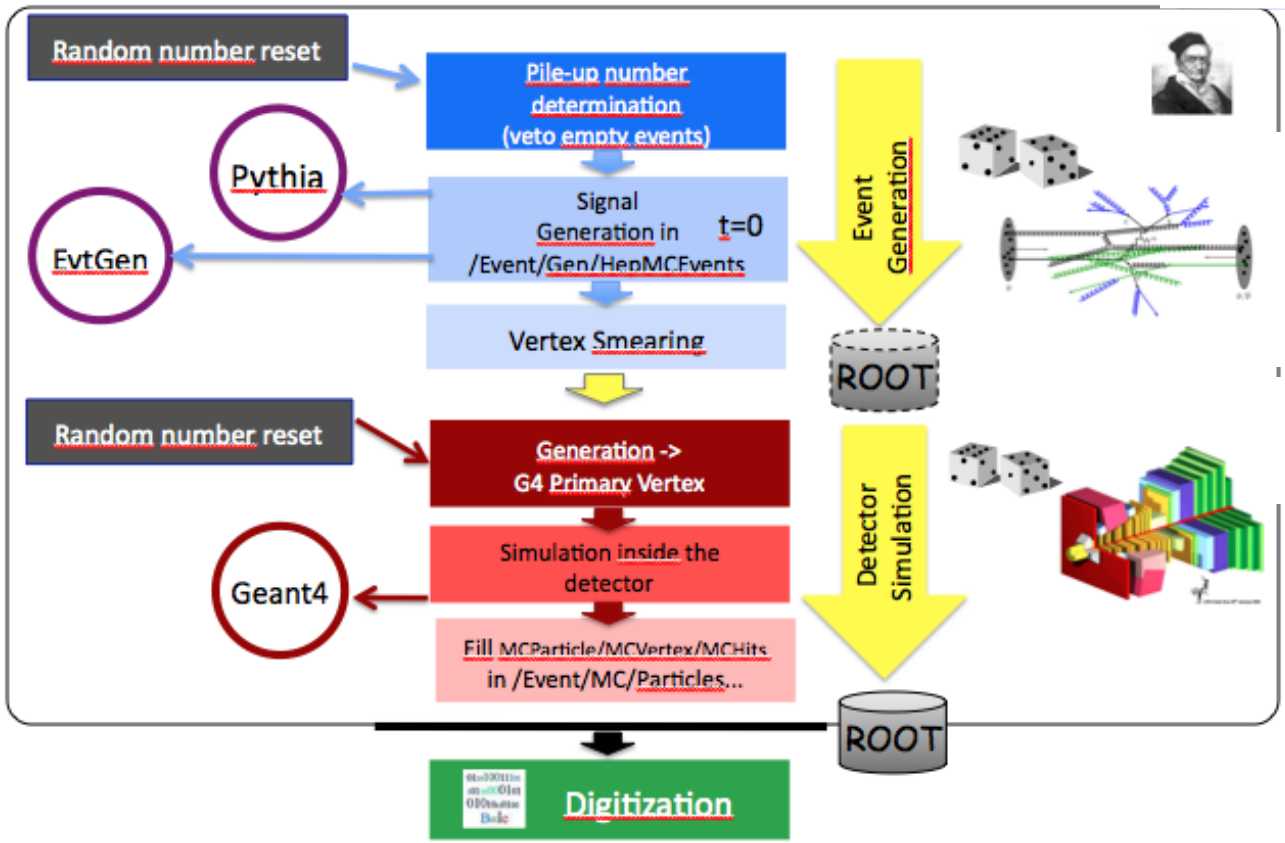
- HepMC was kept in all MC DST files with MC Truth to provide information for analysis of simulated events. But very expensive in term of disk space vs the rest of the LHCb (packed!) event model

	zipped	unzipped
Total	398	1040
HepMC	61	184
MCParticles	46	101
MCVertices	57	107

- Starting from productions made in 2013, HepMC is only kept in generator level productions
  - Heavy quark information needed for flavour tagged analysis and excited b and c hadrons copied to MC Truth
  - will likely be dropped for some of the central productions of this type too
- Users in some cases access directly via ROOT files with only generator level information for their private studies

Simulation framework and HepMC

- HepMC used not only to save the events on output but also as input to transfer the event to Geant4 for the detector simulation
  - The processing of particle is done based on the status code in HepMC

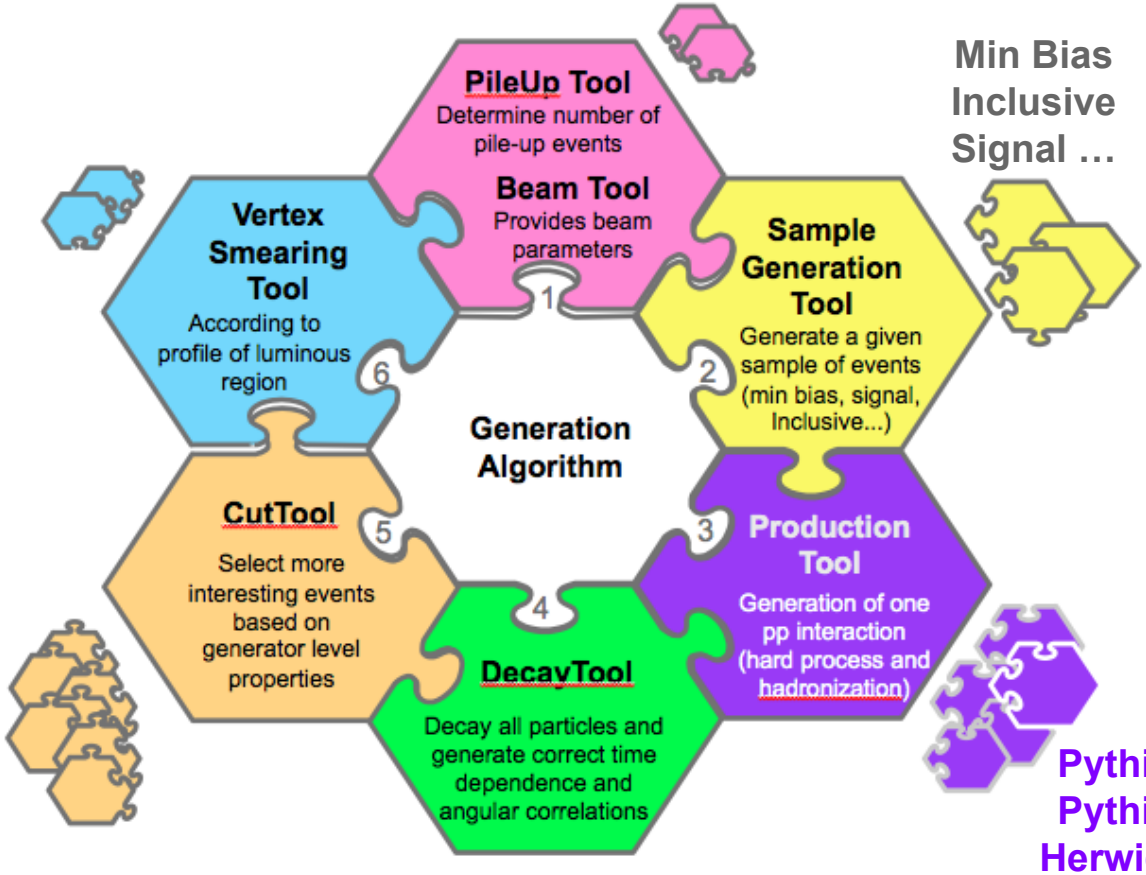


Two INDEPENDENT phases normally run in sequence as in production

but generator phase can and is run by itself

Generator phase and HepMC

- In LHCb the generator phase in handled by different pieces of code each with a well defined action



HepMC is used as exchange format between these separate objects, in particular the Production (e.g. by Pythia) and the Decay (by EvtGen)

Generators and HepMC

- The functionality of Production and Decay tools is implemented using external and LHCb custom generator libraries
- Generators are fixed for massive productions in a given version of the LHCb simulation application, Gauss

  → ALL generators have to use the same version of HepMC

  - Pythia6 (via HepMCfio HEPEVT wrapper library)
  - Pythia8 (via its hepmcinterface library)
  - EvtGen (see next slide)
  - Photos++ and Taula++ used through EvtGen
  - Herwig++ (via ThePeg converter)
  - Hijing (via HepMCfio HEPEVT wrapper library)
- We also use a ReadHepMCAsciFile production tool to try out other generators before integrating them in our framework
  - And a WriteHepMC to dump them
- And RIVET for tuning (integrated in Gauss)

HepMC use in EvtGen

from John Back, Warwick EvtGen group

- EvtGen uses its own EvtParticle class to represent a particle
  - an “event” is just a list of these objects
- HepMC only used as an (optional) output format once EvtGen has finished generating the decay
  - Returns HepMC::GenEvent in the frame of the “mother” or primary/base particle
  - Each HepMC::GenVertex = a particle decay
  - EvtParticle = HepMC::GenParticle(4-mtm, PDG id, status = stable/ decayed, position ≡ lifetime)
  - Nested vertices follow the complete particle decay tree

HepMC in LHCb detector simulation

- Geant4 does not have a tree structure to keep history
- Introduced use of HepMC internally to Geant4 to provide such a tree structure
  - Use LHCb-specialization of Geant4 classes (G4EventAction & G4TrackAction)
  - Use of G4Track 'id' as barcode id
  - Introduce split of particles, i.e. additional vertices to attach secondary particles produced 'in flight' like Bremstrahlung photons.
  - Use the barcode of the production vertex to store the type of the originating process, i.e. hadronic interaction, pair production, etc.

Summary and wishes

- HepMC is extensively used in LHCb.
  - Using HepMC 2.06.05 in production for 2011 and 2012 analysis and 2.06.08 for development
- The HepMC version has to be kept synchronized between all generators used by the LHCb simulation application, Gauss
  - It may influence moving to a new version of a given generator, or dropping it for a while in case of incompatibility.
- Modification of the HepMC record will affect mostly Gauss, but may also affect analysis code
  - Timing of change may be an issue
- MC samples written with old versions of HepMC have to be readable with newer versions
  - Compatibility to be provided centrally
- If new functionality then it would be nice to have general event information – some already there