The Gaussino core simulation software

Marco Clemencic, Gloria Corti, Michał Mazurek,
Adam Morris, Witold Pokorski

CERN, NCBJ

WLCG/HSF Workshop, DESY, 2024-05-14

Some history

Gauss: the LHCb simulation application [Old docs]

- Generates events using external packages (*e.g.* Pythia8, EvtGen, ...)
- Simulates interactions with the detector with Geant4
- First production version in 2004
- Based on the Gaudi software framework [gaudi/Gaudi]

A need to upgrade the software

- Need for code optimisation
- Reduce memory usage
- Adapt to changes in LHCb & HEP common software
- Support multi-threading
- Support fast simulation techniques

Gaussino

Main idea

- Extract the **experiment-independent functionality** from Gauss
- Developed in collaboration with CERN EP-SFT
- **Standalone application** with minimal functionality
- **Toolkit** for building experiment-specific applications

Features kept from Gauss

- Similar modularity
- Integrated gen and sim phases
- MC truth output
- Gaudi algorithms, tools etc
- High-level configuration in python

New features

- Multi-threaded event loop in Gaudi
- Multi-threaded Geant4
- Interface for custom simulation with Geant4
- Interface to new external libraries
  - *e.g.* DD4Hep (detector description)
  - *e.g.* Machine Learning libraries

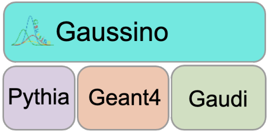
gaussino.docs.cern.ch

GitLab repository

From initial design to full-scale production

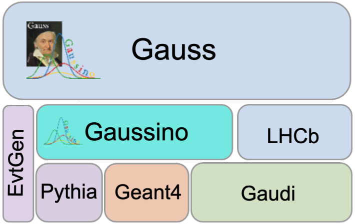
Standalone Gaussino (out-of-the-box) [Docs]

- Generate collisions with Pythia8 or shoot individual particles with ParticleGun
- Choose from a set of Geant4 physics lists
- Define simple detector geometry in the configuration or provide DD4hep or GDML files

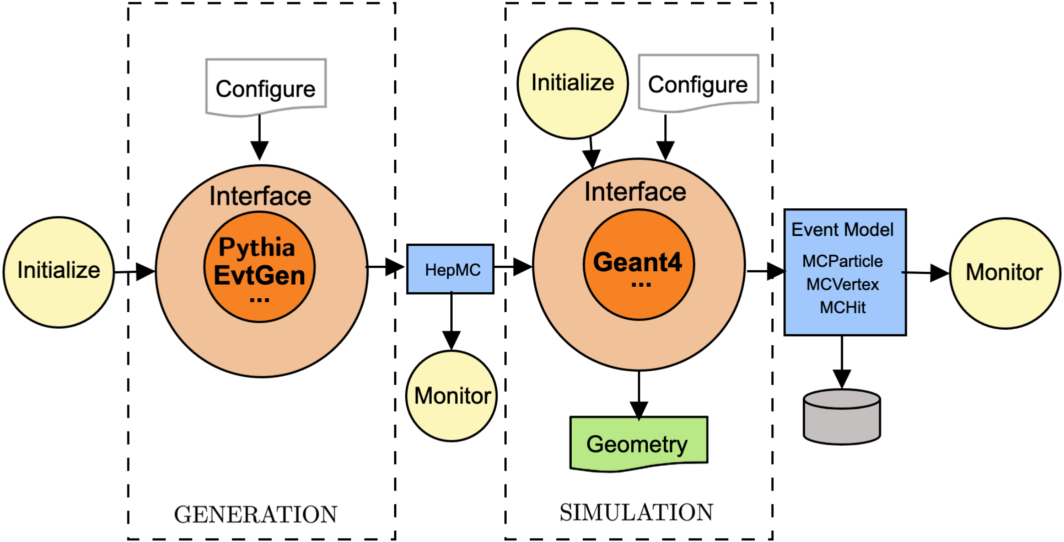


Gauss on Gaussino [Docs] [lhcb/Gauss]

- LHCb simulation application built on top of Gaussino, adding the experiment-specific parts
- Not the focus of this talk but a comprehensive example of using Gaussino as a toolkit

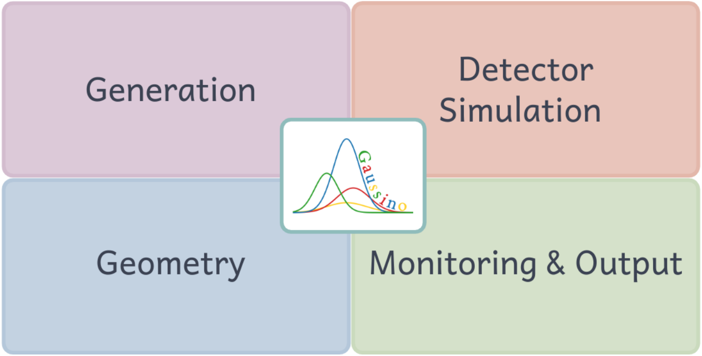


Generation and Simulation phases



Configuration

- Python configuration steering C++ classes
- Modular structure with 4 main configurables

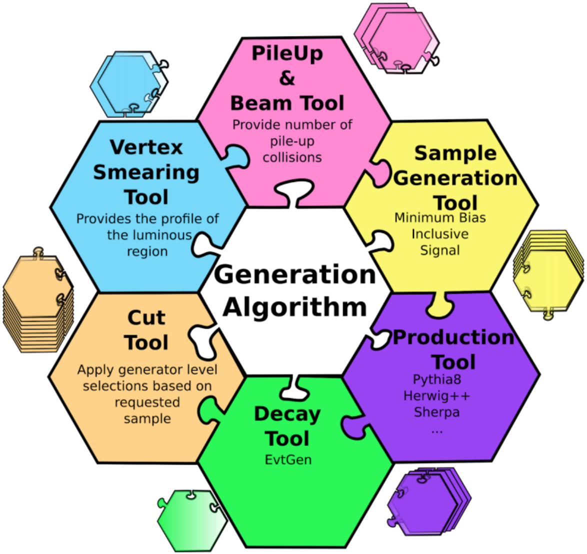


Generation phase

- Extracted from Gauss
- Highly modular
- Output in HepMC3 format

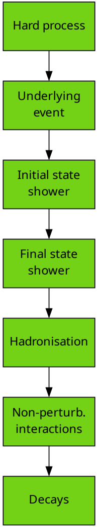
  [Talk by A. Verbytskyi]

- Interface for Pythia8 included



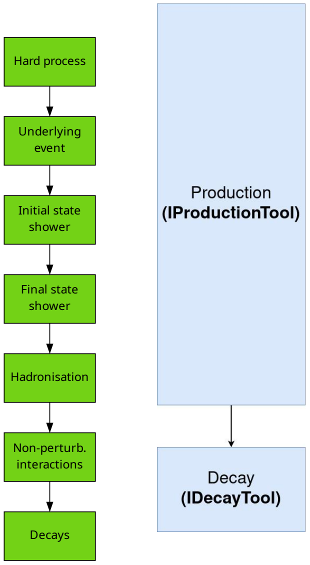
Generator interface developments

- ⚠ Ongoing developments to allow **fine-grained control over different stages of generation**
- More control in the python configuration with the new interfaces
- Design choice influenced by what generators provide as user hooks
- LHE as exchange format



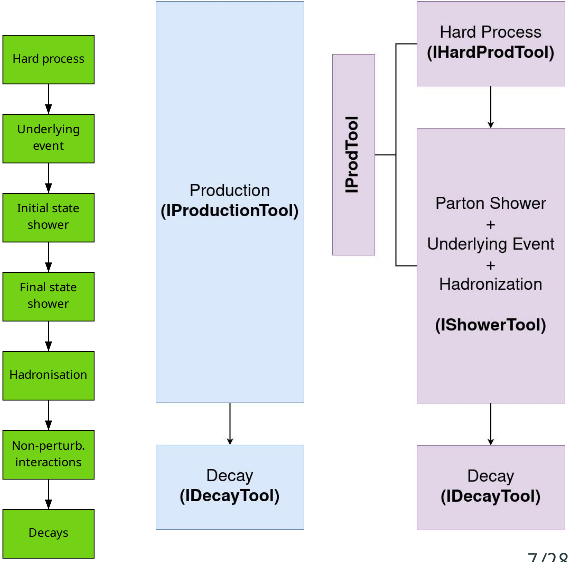
Generator interface developments

- ⚠ Ongoing developments to allow **fine-grained control over different stages of generation**
- More control in the python configuration with the new interfaces
- Design choice influenced by what generators provide as user hooks
- LHE as exchange format



Generator interface developments

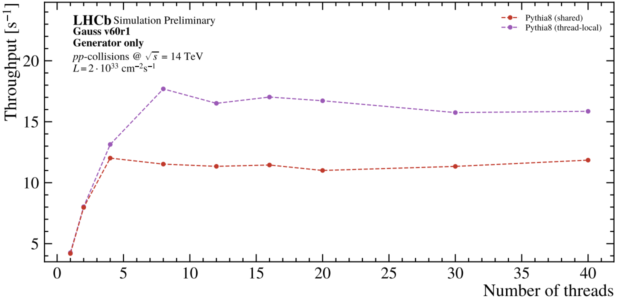
- Ongoing developments to allow **fine-grained control over different stages of generation**
- More control in the python configuration with the new interfaces
- Design choice influenced by what generators provide as user hooks
- LHE as exchange format



Multi-threading with single-threaded generators

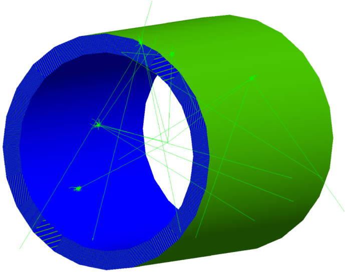
Two approaches:

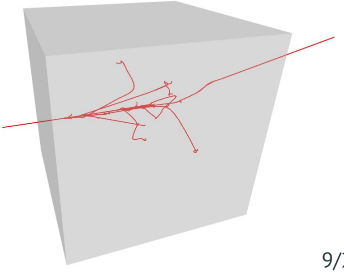
- Thread-local instance if supported by generator
- Shared instance



Pythia8 in LHCb Run 3 conditions

- Re-implementation with **improved modularity**
- Interaction with Geant4
- Infrastructure for:
  - custom simulation
  - re-use underlying event
  - ⚠ parametric (ultra-fast) sim+reco [CHEP 2023]
- Generic geometry service



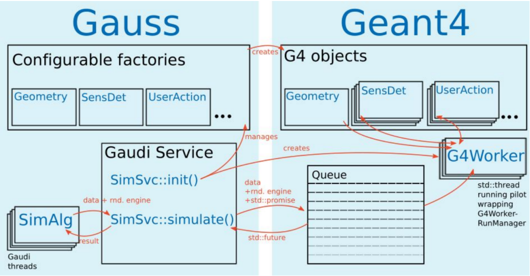


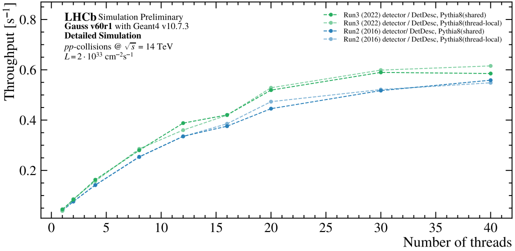
Interface with Geant4

- **Gaudi tools as factories for Geant4 objects**

Multi-threaded Geant4

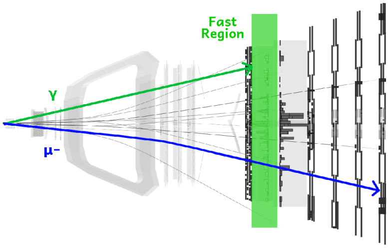
- Gaudi works with task-based multithreading
- Geant4 10 has event-based multithreading
  - $\rightarrow$ control Geant4 processes ourselves
  - **reimplementation of `G4EventManager` to allow Gaudi to take control**
- Geant4 11 introduces task-based MT
  - ⚠ Need to investigate how to incorporate this





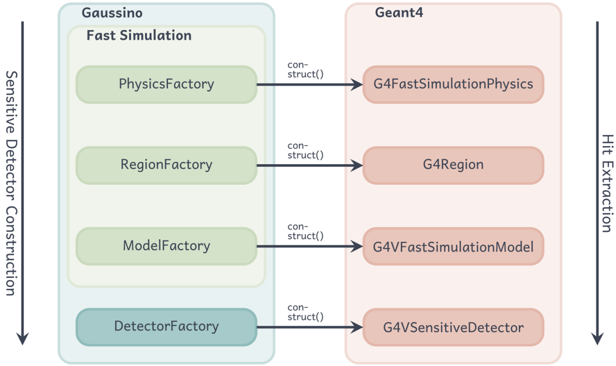
Geant4 10.7 in LHCb Run 3 conditions

Custom simulation



Delegate simulation of certain particles in a region of the detector to *e.g.*

- Point library for calorimeters
- Machine learning models (*e.g.* GANs)
- Particle transport on GPUs

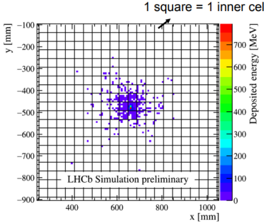


- Use Geant4's fast simulation hooks
- Full-detector transport via physics list

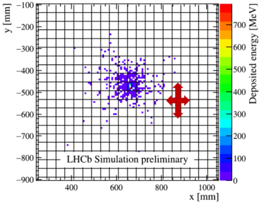
Custom simulation: Parametric simulation

Point libraries [ICHEP 2020]

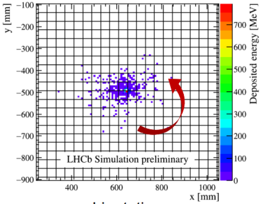
- Library of energy deposits extracted from detailed simulation
- Transform based on the properties of the impinging particle



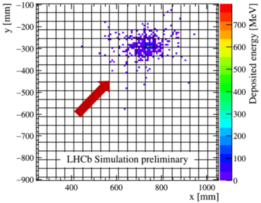
as taken from library



tiny stretching/translation for internode correction



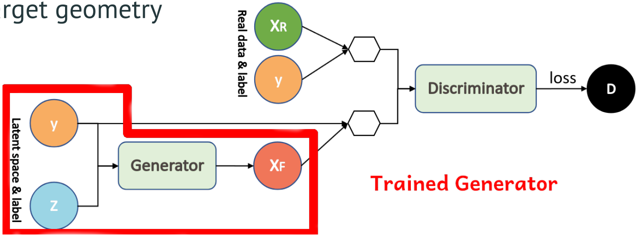
phi rotation



translation

Machine learning [CHEP 2023]

- Train generative models on output of Geant4
- Produce hits from those models during production
- Interfaces for ONNXRuntime & PyTorch backends
- Supports CaloChallenge workflow [Docs]
    - train on experiment-agnostic data
    - compare models objectively
    - retrain chosen model on target geometry



[Diagram credit: M. Bagheri]

Custom simulation: Accelerated particle transport

Offload certain physics processes to different computing architecture

Electromagnetic physics on GPUs

- AdePT[GitHub]: ongoing work to integrate in Gaussino
    - will provide guidelines for integrating the others
- Celeritas[GitHub]: preliminary studies

Optical photons on GPUs

- Mitsuba[GitHub]: preliminary studies, developed outside HEP

Event splitting & merging

- Mechanism for **selectively simulating parts of an event** and merging the output
- Proven track-record in LHCb for **greatly reducing computing time** spent in Geant4
- Currently has two implementations (ReDecay & SplitSim) but could be extended

ReDecay[EPJC (2018) 78:1009]

- Re-use same underlying event for many signal decays
- Particularly useful when the production mechanism is not studied
  - *e.g.* Beauty & Charm decays

SplitSim

- Simulate part of the event before applying a cut
- Efficient filtering on material interactions or particles decayed by Geant4
  - *e.g.* Converted photons
  - *e.g.* Rare K^0_S decays

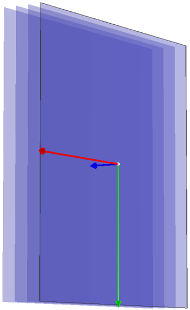
Geometry

Generic service to steer **passing of information to Geant4** from different backends:

- DD4Hep [Docs] [Talk by T. Madlener]
- Import & export of GDML files
- Custom service for "internal" volumes of simple shapes: `ExternalDetector`

Geometry: ExternalDetector

- Provides necessary tools to **embed volumes** and **mark as sensitive**
- Works **stand-alone**, or can be **mixed with other services**
- Wrapper classes around `G4VSolid`, easily extensible
- Factory classes to create `G4Materials` based on chemical properties or elements
- Can attach hit extraction and monitoring algorithms
- Supports Geant4 parallel worlds [Docs]



Tracker planes created with CuboidEmbedder (G4Box)

Visualisation

Visualisation in Geant4 is a crucial part of **verifying the geometry after conversion**

Dedicated steering in Gaussino due to Gaudi & Geant4 multi-threading interplay:

- Visualisation has its own thread
- Information exchange at the right time

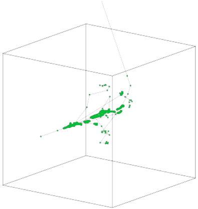
Two options implemented:

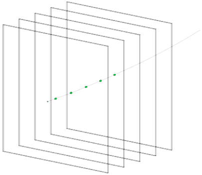
- Native Geant4 visualisation drivers
- Phoenix event display

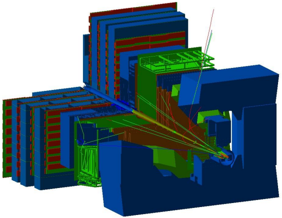
Visualisation

Geant4 visualisation drivers

- Available **at run time**
- Volume overlap checks possible
- **Geant4 data only**
- Drivers: ASCIITree, OpenGL, DAWN, HepRep



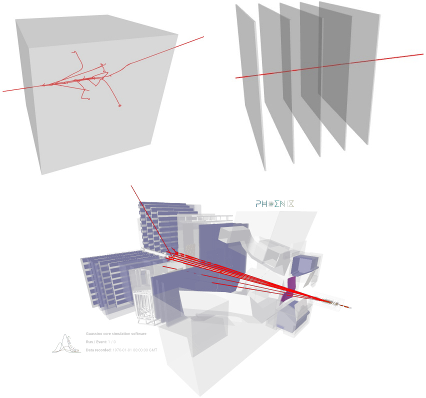




Visualisation

Phoenix event display [Talk by E. Moyse]

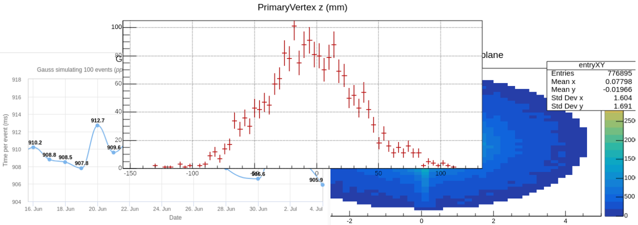
- Available as **external tool**
- Supports a variety of geometry and event formats
  - GDML must be converted to *e.g.* glTF
  - Internal JSON format for event data if not using a supported one
- Possible to **compare simulated and reconstructed data**



Monitoring & Output

Various persistent output formats possible with pre-defined contents:

- Built-in event model
- Consistent MC truth: combined info from generator and Geant4
    - Choice of details to keep
- Histograms
- Counters
- Custom n-tuples



Configuration

One or more python scripts that manipulate the 4 configurable objects:

- `Gaussino`
- `GaussinoGeneration`
- `GaussinoSimulation`
- `GaussinoGeometry`

Execution

Invoke with:

```
gaudirun.py options.py [options2.py] [...]
```

Gaussino as a standalone application

Basic configuration

```python
# General imports
from Configurables import Gaussino, GaussinoGeneration, GaussinoSimulation, GaussinoGeometry
from GaudiKernel import SystemOfUnits as units
# Specify number of events
Gaussino().EvtMax = 1000
# Run number informs random seed
Gaussino().RunNumber = 1234
# Set even numbers (useful when splitting into multiple jobs)
Gaussino().FirstEventNumber = 3001
# Specify which phases to run
Gaussino().Phases = ["Generator", "Simulation"]
```

Multi-threading

```python
Gaussino().EnableHive = True,
Gaussino().ThreadPoolSize = 4
Gaussino().EventSlots = 4
```

Gaussino as a standalone application

ParticleGun

```python
# Configure ParticleGun to produce 1 GeV photons
from Configurables import ParticleGun, FixedMomentum, \
FlatNParticles

GaussinoGeneration().ParticleGun = True
pgun = ParticleGun("ParticleGun")
pgun.addTool(FixedMomentum, name="FixedMomentum")
pgun.ParticleGunTool = "FixedMomentum"
pgun.FixedMomentum.px = 0.0 * units.GeV
pgun.FixedMomentum.py = 0.0 * units.GeV
pgun.FixedMomentum.pz = 1.0 * units.GeV
pgun.FixedMomentum.PdgCodes = [22]
pgun.addTool(FlatNParticles, name="FlatNParticles")
pgun.NumberOfParticlesTool = "FlatNParticles"
pgun.FlatNParticles.MinNParticles = 1
pgun.FlatNParticles.MaxNParticles = 1
```

Pythia8

```python
# Configure Pythia8 to produce proton-proton collisions
GaussinoGeneration(
    # Choose the Gaudi tools to configure the generator
    # NB: tool-specific options can be passed here
    SampleGenerationTool = "MinimumBias",
    ProductionTool = "Pythia8ProductionMT",
    PileUpTool = "FixedLuminosityWithSvc",
    # Set the beam configuration (approx. 2023 LHCb)
    B1Particle = "p",
    B2Particle = "p",
    BeamMomentum = 6.8 * units.TeV,
    BeamEmittance = 0.01845 * units.mm,
    BeamBetaStar = 2.0 * units.m,
    BunchRMS = 63.36 * units.mm,
    Luminosity = 1.75e29 / (units.cm2 * units.s),
    TotalCrossSection = 102.5 * units.millibarn,
    RevolutionFrequency = 11.245 * units.kilohertz,
    # Crossing angle and interaction point
    InteractionPosition = [0.0] * 3,
    BeamHCrossingAngle = -0.145 * units.mrad,
    BeamVCrossingAngle = +0.200 * units.mrad,
    BeamLineAngles = [0.0, 0.0],
)
```

Particle transport & detector simulation

```
# Geant4 physics lists
GaussinoSimulation().PhysicsConstructors += [
    "G4EmStandardPhysics",
    "G4HadronPhysicsFTFP_BERT",
]
```

```
# Custom simulation
GaussinoSimulation().CustomSimulation = "MeshModelSimulation"
customsim = CustomSimulation("MeshModelSimulation")
customsim.Model = { # Which custom simulation model to use
    "MeshModel": {
        "Type": "Gaussino__G4Par04__MeshModelFactory",
    }
}
customsim.Region = { # Which region to delegate
    "MeshModel": {
        "SensitiveDetectorName": "CollectorSDet",
    }
}
customsim.Physics = { # Which particles to delegate
    "ParticlePIDs": [22, 11, -11],
}
```

Gaussino as a standalone application

Detector geometry

```python
from Configurables import ExternalDetectorEmbedder
from ExternalDetector.Materials import LEAD

# Create the geometry service
GaussinoGeometry().ExternalDetectorEmbedder = "ExternalDetectorEmbedder"
external = ExternalDetectorEmbedder("ExternalDetectorEmbedder")
# Materials
external.Materials = {
    "G4_AIR": {"Type": "MaterialFromNIST"},
    "Pb": LEAD,
}
# Define the surrounding world
external.World = {
    "WorldMaterial": "G4_AIR",
    "Type": "ExternalWorldCreator",
}
```

e.g. a 1 m^3 cube of lead at z=10 m surrounded by air

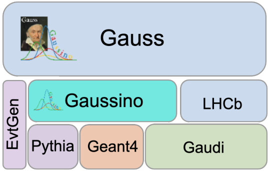
```python
# Define a simple shape
external.Shapes = {
    "CubeOfLead": {
        "Type": "Cuboid",
        "MaterialName": "Pb",
        "xPos": 0 * units.m,
        "yPos": 0 * units.m,
        "zPos": 10.0 * units.m,
        "xSize": 1.0 * units.m,
        "ySize": 1.0 * units.m,
        "zSize": 1.0 * units.m,
    },
}
# Mark sensitive
external.Sensitive = {
    "CubeOfLead": {
        "Type": "MCCollectorSensDet",
        "PrintStats": True,
    },
}
```

Gaussino as a toolkit

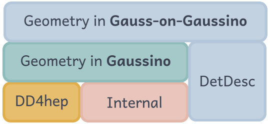
Gaussino as the basis for a fully-fledged experiment-specific simulation application

Gauss-on-Gaussino [Docs] [lhcb/Gauss]

- LHCb simulation application
- Adds **experiment-specific components and configuration**
  - LHCb event model
  - Customised generators & EvtGen
  - DD4hep & DetDesc (legacy) descriptions of LHCb throughout the years
  - Subdetector-specific monitoring
- **Ongoing production tests on the grid**
- Aim to use for **all LHCb simulations** in the future including pre-Upgrade I (Runs 1 and 2)
- Already used in Upgrade II (Run 5) studies



Dependency structure



Geometry dependencies

Conclusion

Gaussino provides:

- an **experiment-independent core simulation framework**
  - ✓ easy configuration in python
  - ✓ modular generator and detector simulation phases
  - ✓ support for custom simulation
  - ✓ internal geometry service
- a test-bed for **detector developments**
- a test-bed for **new simulation techniques**
- a toolkit for full-scale experiment-specific simulation software

gaussino.docs.cern.ch

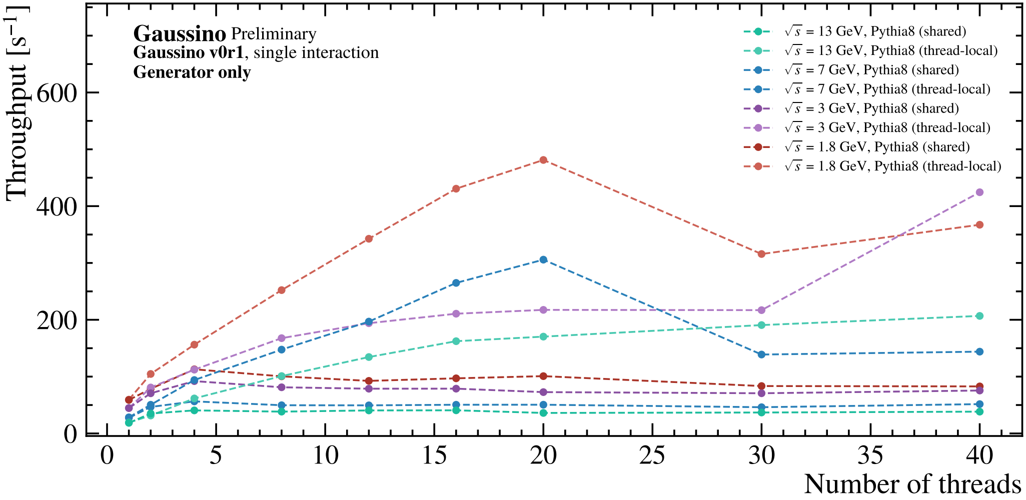
GitLab repository

Appendix

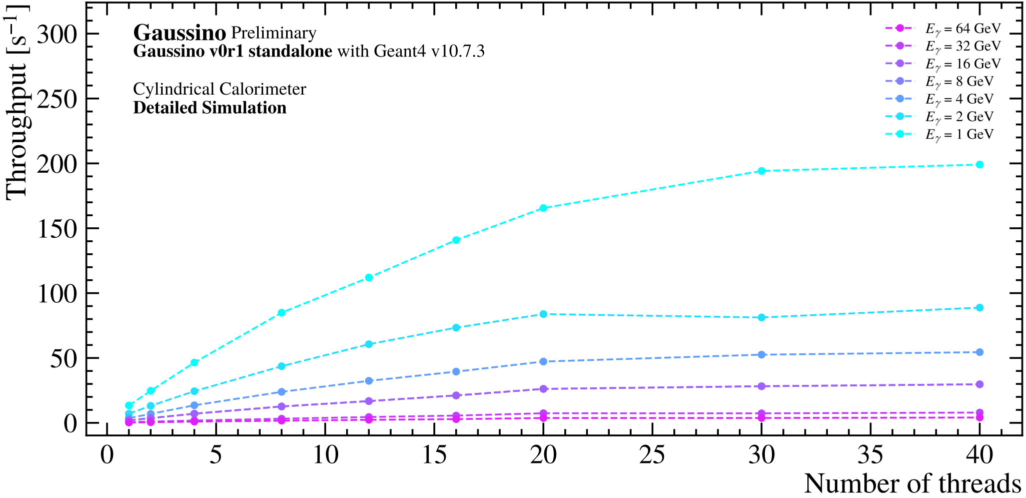
Random numbers

- Ensure **reproducibility**
- Seed initialised with
  - run number
  - event number
  - algorithm instance name
- Create random engines on the stack

Generator performance



Simulation performance



Generic ML interface

ML model serving

- Use Gaudi **services**
- Handle loading of the model
- Accessible throughout the whole execution
- Set general properties

ML model evaluation

- Use Gaudi **tools and algorithms**
- Pass the random generator seed to ensure reproducibility
- Fixed or automatic types for inputs & outputs