Generalization and Reusability an LHCb Viewpoint

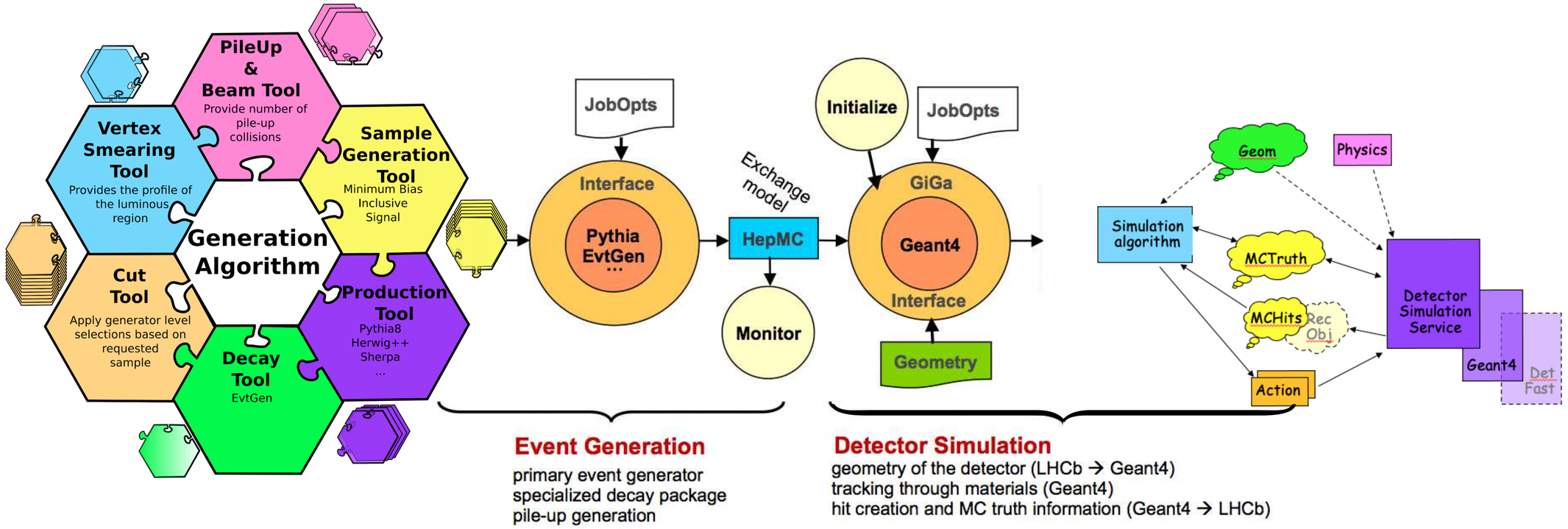
Adam Davis
on behalf of the LHCb Simulation Project
23 November 2021

The University of Manchester

LHCb Simulation - an overview

J Phys.: Conf. Ser 331 032047
NSS2010 - CD Conference Record, N42-284;
LHCb-PROC-2010-056

- The LHCb Simulation software, Gauss, is by design modular



- Allows us to efficiently switch between generator and fast simulation option by simple options

- The framework is *general* enough to be able to seamlessly switch without the user knowledge of inner workings

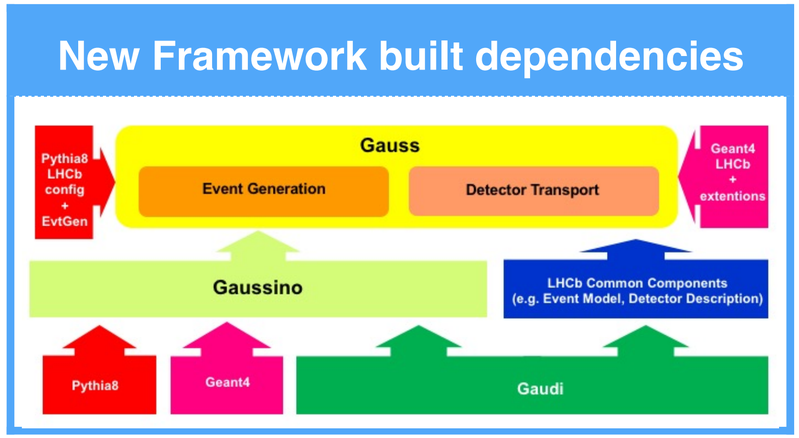
Generalization and Reusability

- This structure has been crucial for the simulation needs of LHCb (think about generating $B_c^+ \to J/\psi\tau\nu$, $D^0 \to K^+K^-$ and $H \to c\bar{c}$, and pp as well as Heavy Ion/Fixed target (SMOG) collisions, all in the same framework)

- Use in multithreaded environments requires further generalization → Gaussino (see Michal's Talk)

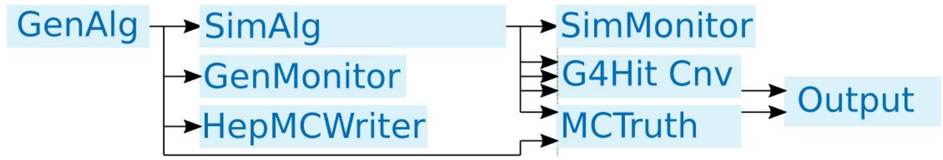
- Do you want event level or particle level parallelization? → This choice will affect your design choice and will limit you in the future if you've not already planned for it

- Generators and Simulation description will have different ideas about what is the best parallelism for them



DOI: 10.1109/NSS/MIC42101.2019.9060074

```
/*output data*/ operator()(const /*input data*/) const
```



Reusability in Fast Simulation

- Looking ahead, we need to be able to choose what fast simulation we want to run
  - I want ReDecay (Eur. Phys. J. C 78 (2018) 1009) with BcVegPy only selecting photons which are reconstructed using a fast calorimeter simulation, but the rest of the event to be simulated as normal.
- Lamarr (fully parametrized simulation, input is generator level particles, output is LHCb reconstructed quantities) was born out of using Delphes which could not fulfill such a need
- We want to also be able to use the *same* framework for Run 1 simulations and for Run 3 and beyond
- Need to already foresee the design choices now

Towards the future

- How are we going to ensure reusability of current software when dealing with new and emerging processors (GPU, IPU, somethingPU)?
- What about FPGA? Are we even going to go there?
- There are options, say OneAPI, which allows for backend agnostic writing, but is this a long term solution?
- In either case, how do you make a framework which is agnostic to your choice?
  - In 10 years, can I produce simulation we made with gcc49-slc6 and Geant4 10.4?
- These are all choices we need to ensure we can make today for tomorrow.