Introduction: Fast Simulation

- Current VMC developments -

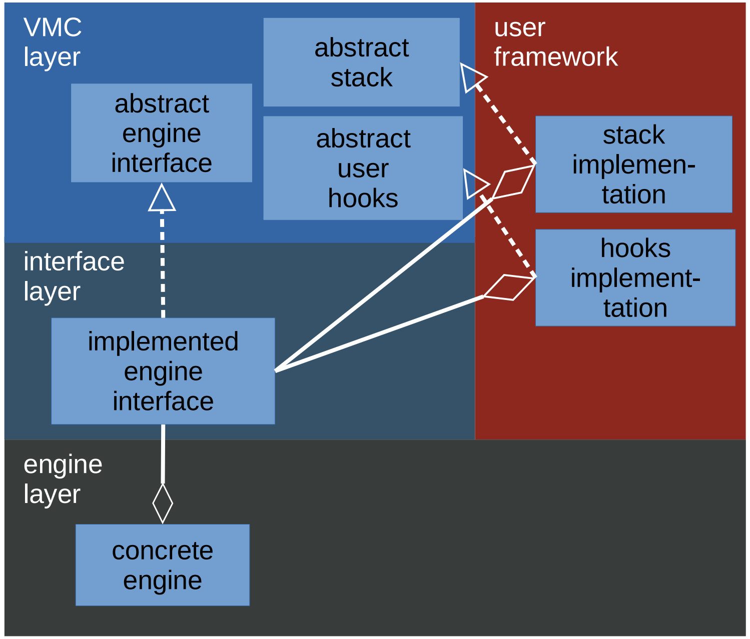
Benedikt Volkel

CERN; Physikalisches Institut, Ruprecht-Karls-Universität Heidelberg

ALICE Software and Computing Week, 03/04/2019

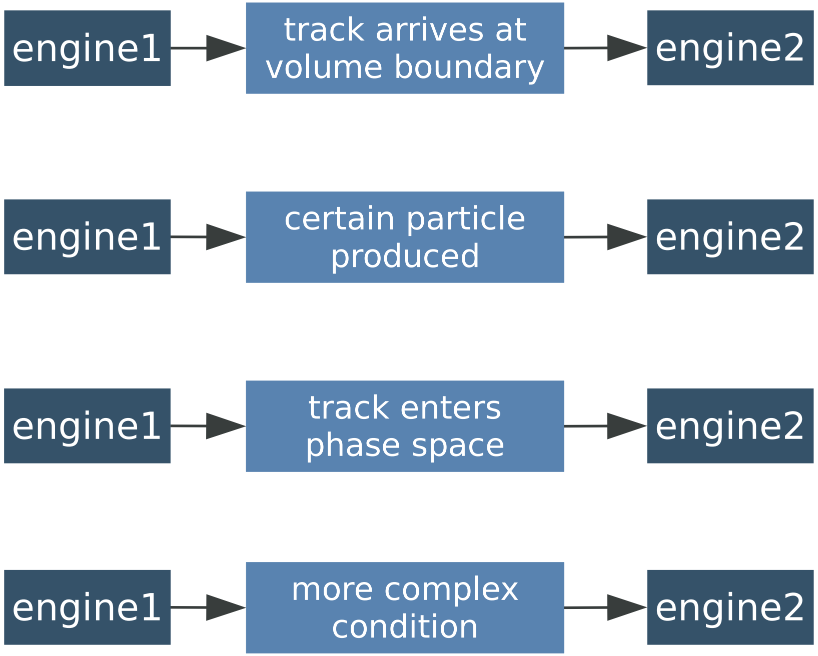
Mixing full and fast simulation in VMC

- ALICE uses detector simulation engines via Virtual Monte Carlo (VMC) interface
- on this level only exactly one engine could be used per event due to singleton structure
- wanted to have the ability of mixing engines depending on user conditions
  → ability of mixing e.g. GEANT3 with GEANT4
  → able to develop fast simulation on VMC level
- extended VMC merged in ROOT
  [https://github.com/root-project/root/pull/3513]



Simplified picture of the extensions

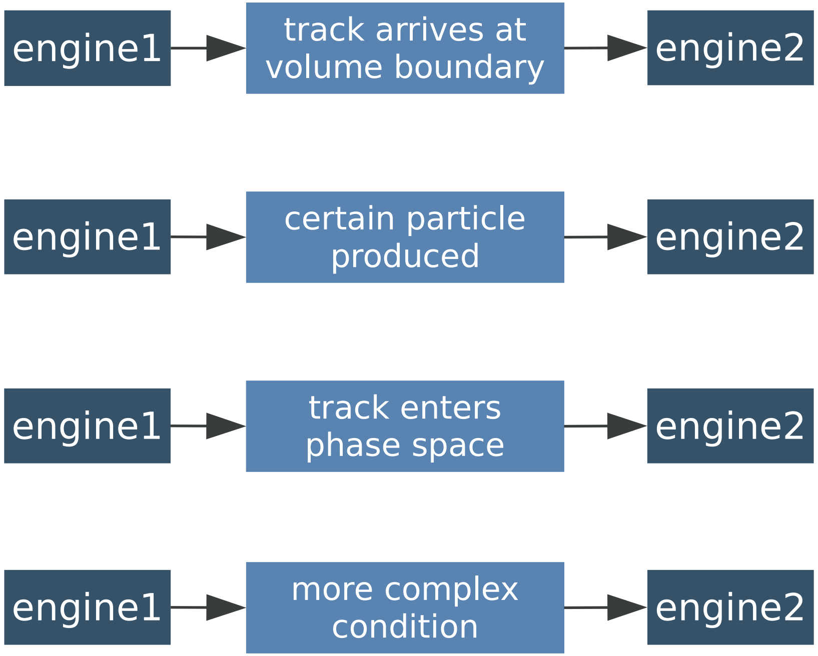
goal: share simulation
assigning specific volumes, particle types, phase space to different engines



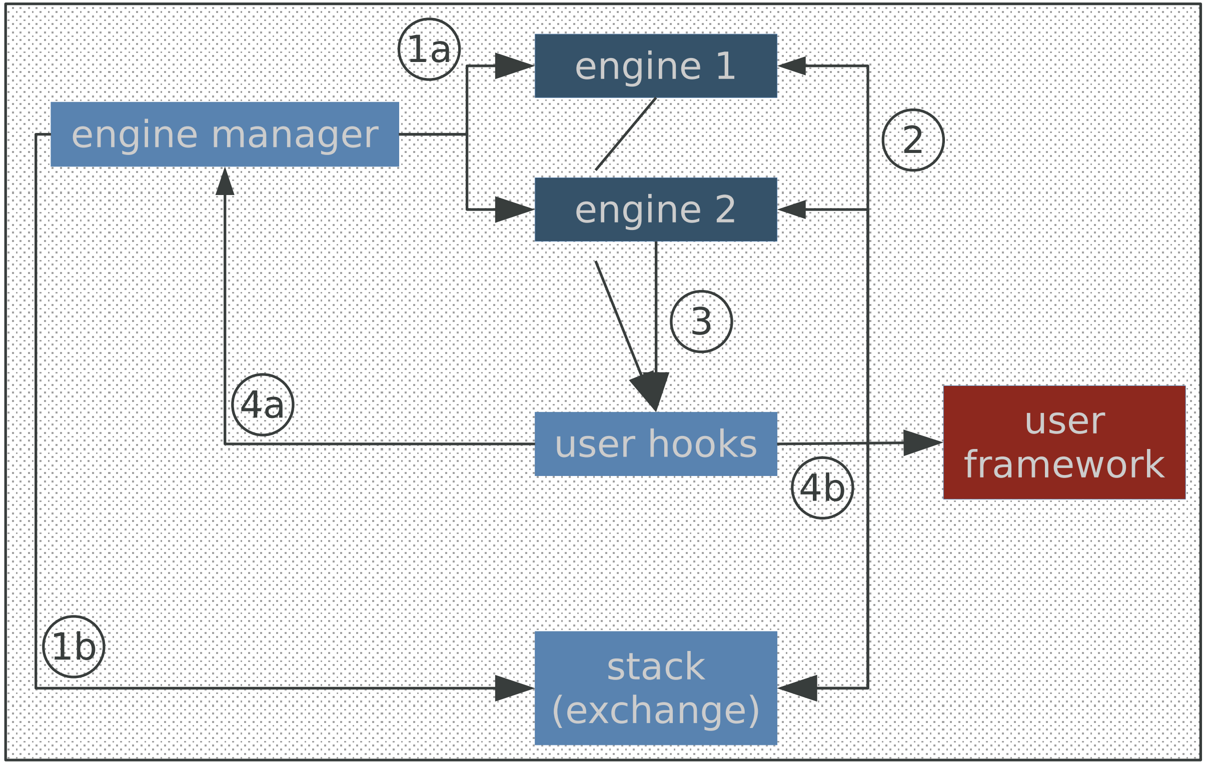
dispatch/track transfer between engines
based on user conditions

Simplified picture of the extensions

goal: share simulation
assigning specific volumes, particle types, phase space to different engines



dispatch/track transfer between engines
based on user conditions



Choosing among multiple engines

Geant3_VMC

```
#include <TVirtualMC.h>

class TGeant3
: public TVirtualMC
{
  // implementations
};
```

Geant4_VMC

```
#include <TVirtualMC.h>

class TGeant4
: public TVirtualMC
{
  // implementations
};
```

VMCFastSim (virtual)

```
#include <TVirtualMC.h>

template <class T>
class VMCFastSim
: public TVirtualMC
{
    //...
    virtual void process() = 0;
    //...
};
```

MyFastSim

```
#include <VCMFastSim/FastSim.h>

class MyFastSim
: public FastSim<MyFastSim>
{
  public:
    //...
    void process() override
    {
      // put implementation here
    }
};
```

FastSim1

FastSim2

FastSim3

Choosing among multiple engines

Geant3_VMC

```
#include <TVirtualMC.h>

class TGeant3
: public TVirtualMC
{
  // implementations
};
```

Geant4_VMC

```
#include <TVirtualMC.h>

class TGeant4
: public TVirtualMC
{
  // implementations
};
```

VMCFastSim (virtual)

```
#include <TVirtualMC.h>

template <class T>
class VMCFastSim
: public TVirtualMC
{
    //...
    virtual void process() = 0;
    //...
};
```

user framework

```
// Get the TMCManager
auto manager = TMCManager::Instance();

// create user stack and notify manager
auto stack = new MyStack();
manager->SetUserStack(stack);

// create engines
//------------------------------------
auto geant3 = new TGeant3();
auto geant4 = new TGeant4();
auto fastSim = new MyFastSim();
//------------------------------------
```

```
// specify condistions how to change
// and transfer tracks among engines

// init...
manager->Init();
// ...and run 42 events
manager→Run(42);
```

MyFastSim

```
#include <VCMFastSim/FastSim.h>

class MyFastSim
: public FastSim<MyFastSim>
{
  public:
    //...
    void process() override
    {
      // put implementation here
    }
};
```

FastSim1

FastSim2

FastSim3

Fast simulation in GEANT4

- GEANT4 offers inherently a solution of plugging in fast simulation into full simulation
  → can be used via GEANT4(_VMC)
  [see also talk by Dmytro Kesan on Friday on FairRoot implementation]
  [code at https://github.com/FairRootGroup/FairRoot/tree/master/base/sim/fastsim]
- fast simulation connected to GEANT4 regions and particle definition (and additional user conditions)
  → automatic dispatch
- offers various functionality
  - user is free regarding the fast sim implementation
  - consistency of transport ensured by GEANT4
  - "path finder" available to automatically transport (incident) particles/tracks to region's boundary (also considering external fields)
  - not directly related but could be useful: cross section biasing depending on regions, correct re-weighting of tracks
    [there was a request/idea by Friederike Bock]
- Use native GEANT4 fast simulation implementations (via GEANT4_VMC)?!

Conclusion and outlook

- extensions of VMC code were merged
  [see https://github.com/root-project/root/pull/3513]
- GEANT3_VMC and GEANT4_VMC extensions to be merged in coming days
- preserved backward-compatibility, no performance overhead in single run
- can freely combine full and fast simulation
  - e.g. if useful, GEANT3 and GEANT4 (and Fluka) can be mixed
  - can use GEANT3 (and Fluka) with fast simulation
- open questions and comments
  - Any requirements of PWGs on VMC? Which functionality are they interested in?
  - In case there will be the decision on moving to GEANT4 as the default engine, could we directly move forward and implement fast simulation using what is offered?
  - Using 2 physics lists when using GEANT4 was considered
    → that is not (yet) supported by GEANT4 itself
    → this cannot be solved immediately using VMC
    (GEANT4 uses a G4RunManager singleton accepting one physics list)

ALICE

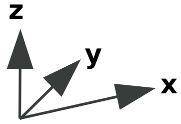
UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

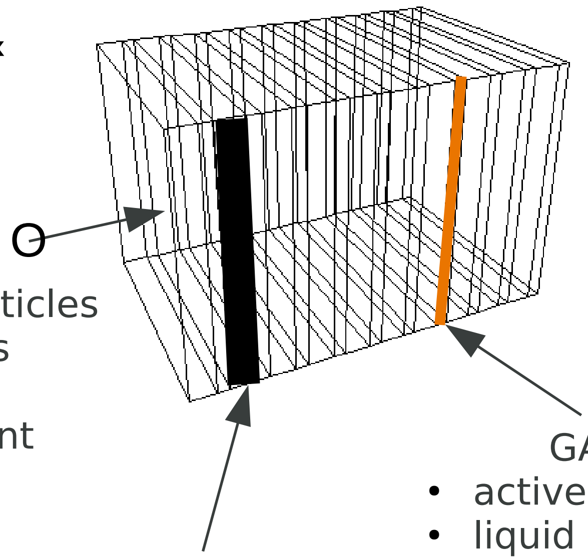
Thank you for your attention

Backup

Testing the code

toy sampling calorimeter for tests



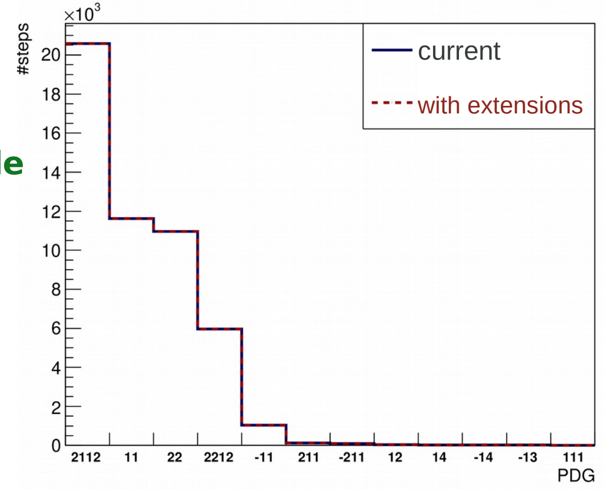


shoot particles
- protons
- 1 GeV
- 1 / event

GAPX
- active material
- liquid argon

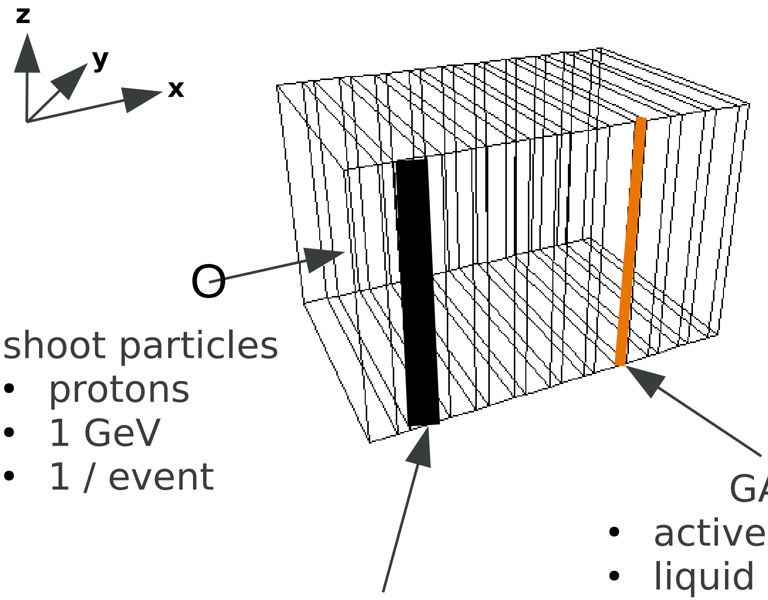
ABSO
- passive material
- lead

backward-compatible in terms of physics
[has also been checked for version compatibility]



Testing the code

toy sampling calorimeter for tests

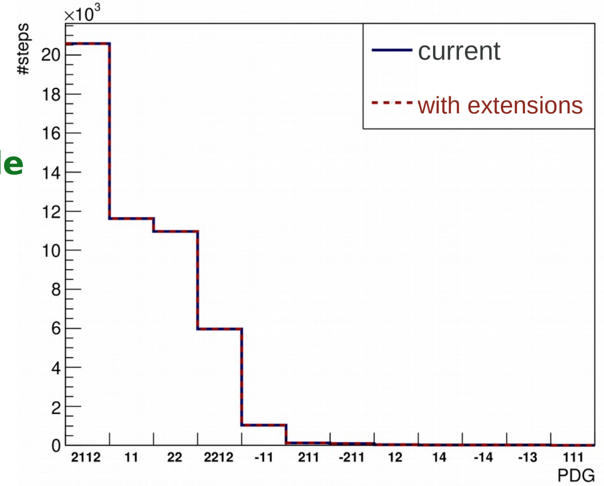


shoot particles
- protons
- 1 GeV
- 1 / event

GAPX
- active material
- liquid argon

ABSO
- passive material
- lead

backward-compatible in terms of physics
[has also been checked for version compatibility]



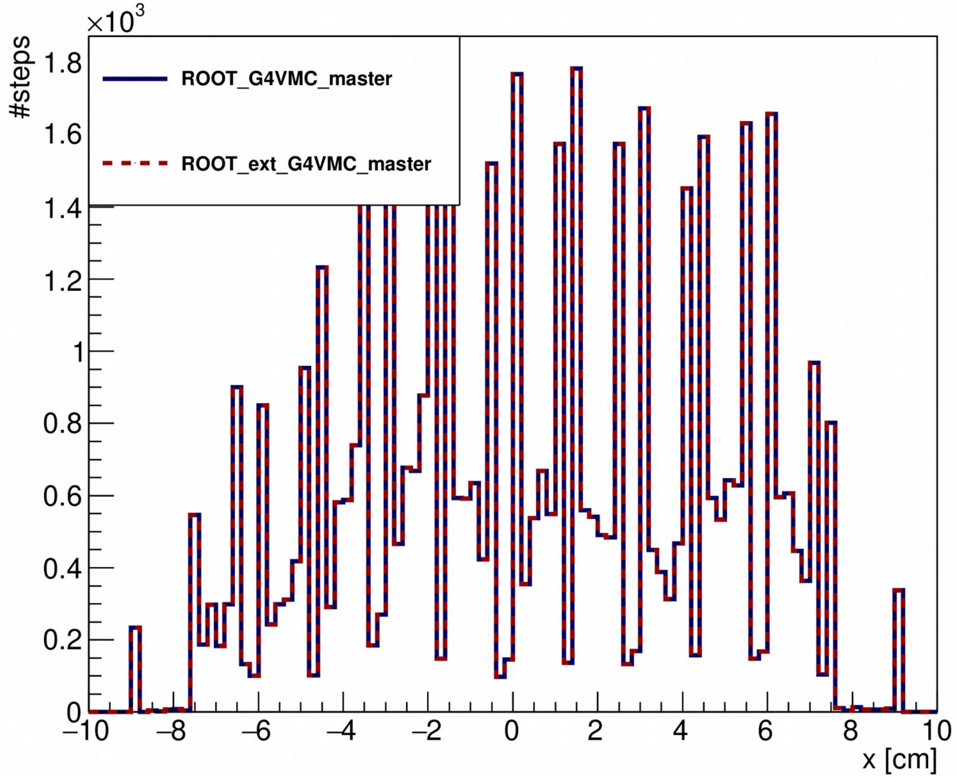
able to mix full sim engines to simulate different parts of the geometry

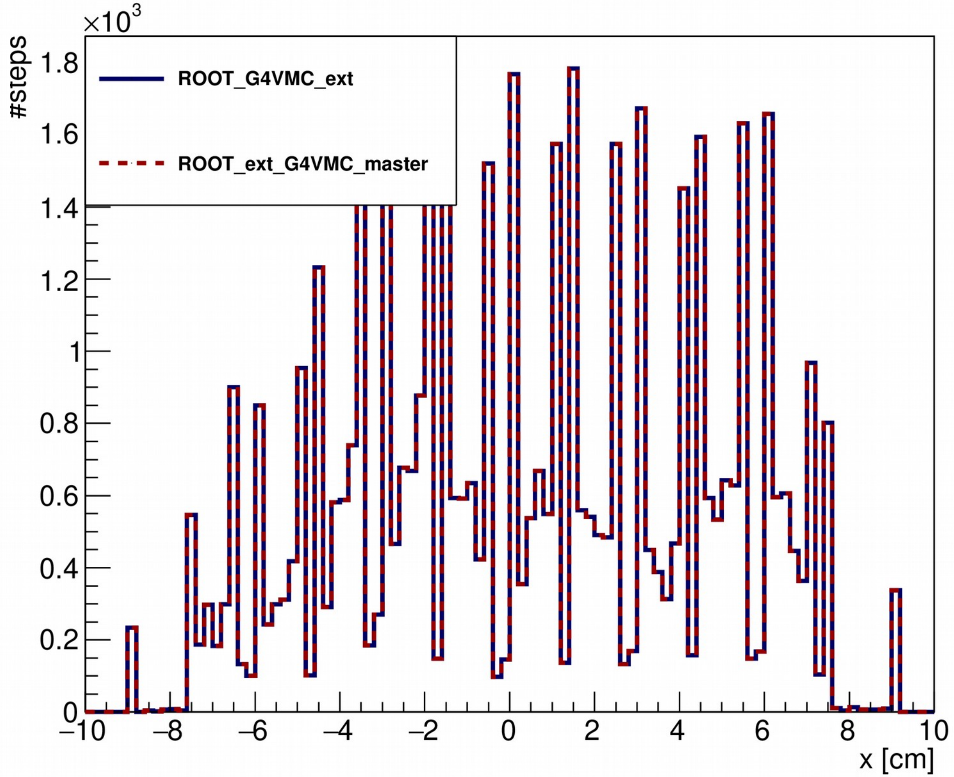
- GEANT3: ABSO
- GEANT4: GAPX

engine	ABSO [rel. #steps]	GAPX [rel. #steps]
GEANT3	~90 %	~10 %
GEANT4	~25 %	~75 %

try to avoid steps of 0 length at volume boundaries before a track is transferred from one engine to another to save computation time

Backward compatibility

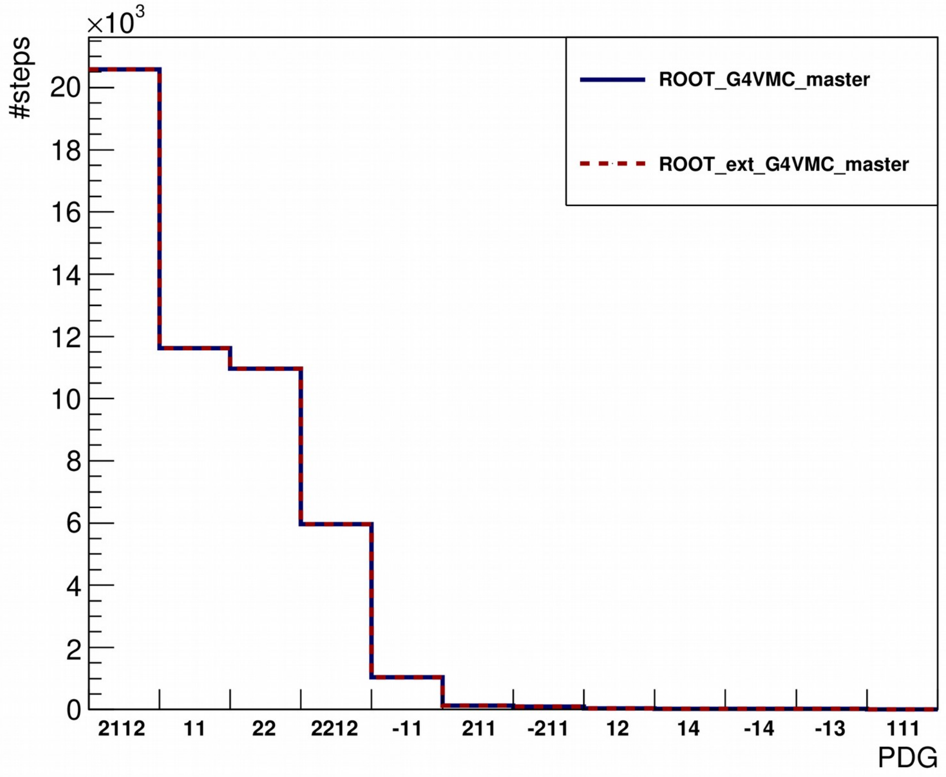


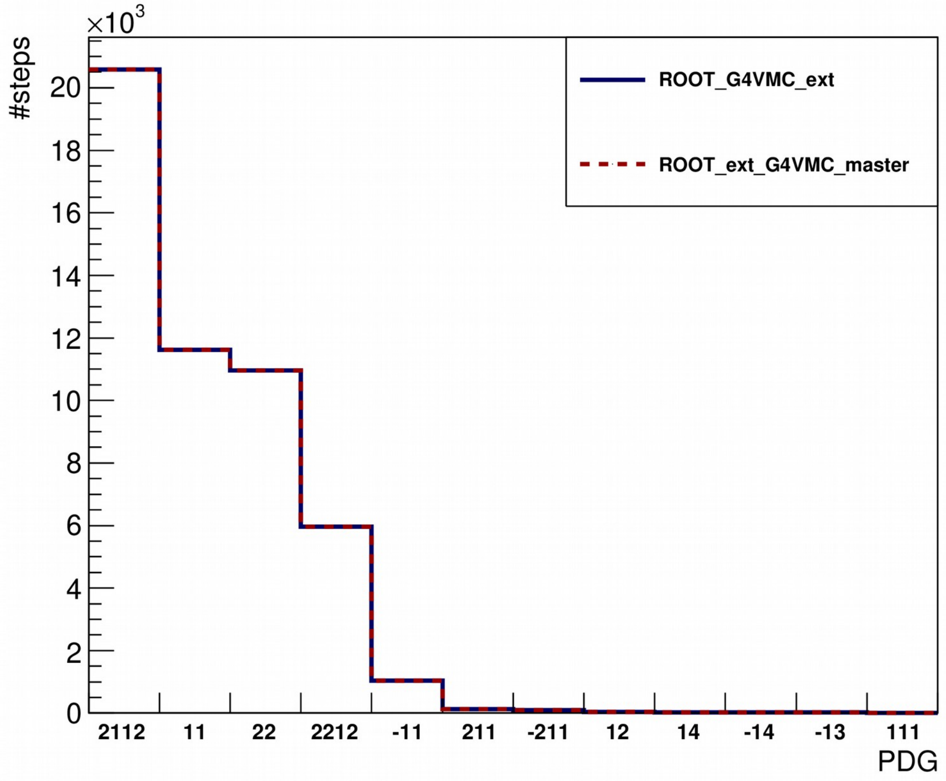


steps in x are perfectly overlaying

[same for other coordinates and observables]

Backward compatibility

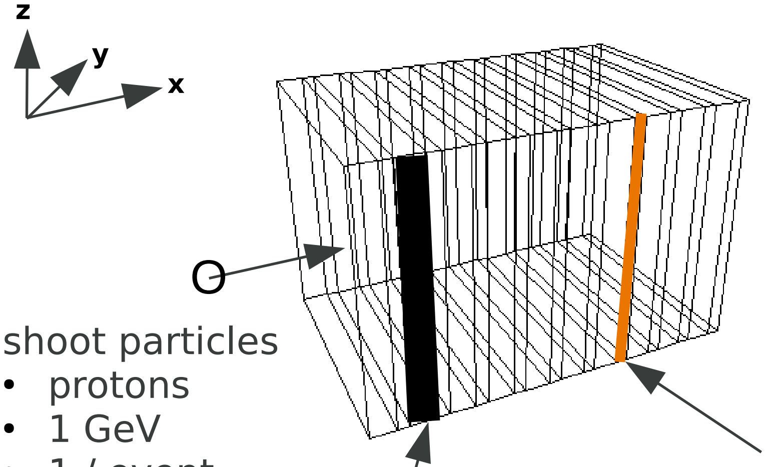




steps made per PDG are perfectly overlaying

Prototype of a fast calorimeter simulation (**WIP**)

toy sampling calorimeter for tests



shoot particles

- protons
- 1 GeV
- 1 / event

GAPX

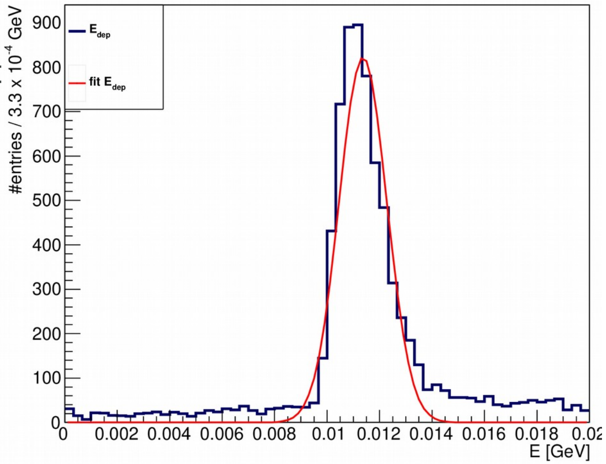
- active material
- liquid argon

ABSO

- passive material
- lead

now: only interested in total energy deposit in sensitive layers

1. derive distribution of total energy deposit from GEANT4
2. fit normal distribution [**proof of principle!**]
3. pass fitted distribution to FastSim
4. FastSim samples total energy deposits from fitted distribution



GEANT4_VMC → VMCFastSim → sum of hits

dispatch when reaching calorimeter

sample and provide energy deposit (hits)

Prototype of a fast calorimeter simulation (**WIP**)

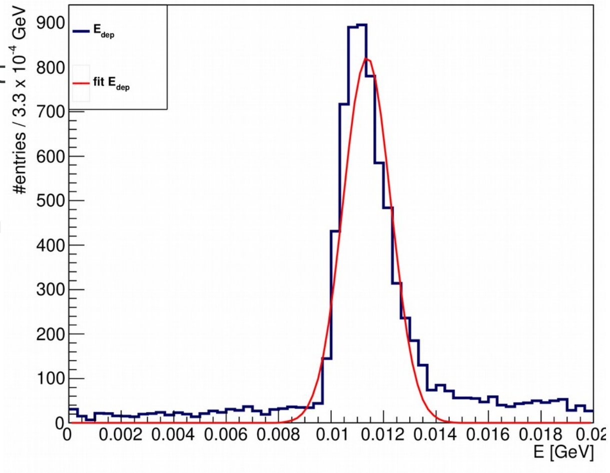
ALICE

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

quite some work ahead

- derive realistic parametrization
- parametrization for different energies and particles
- provide energy per sensitive layer
- set incident particle to the boundary of the calorimeter to be transported further (including potential magnetic field interactions)
- ...

1. derive distribution of total energy deposit from GEANT4
2. fit normal distribution [**proof of principle!**]
3. pass fitted distribution to FastSim
4. FastSim samples total energy deposits from fitted distribution



dispatch when reaching calorimeter

sample and provide energy deposit (hits)

Prototype of a fast calorimeter simulation (**WIP**)

ALICE UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

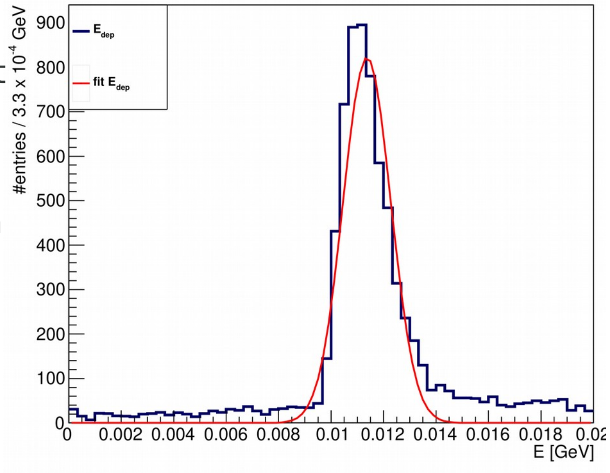
quite some work ahead

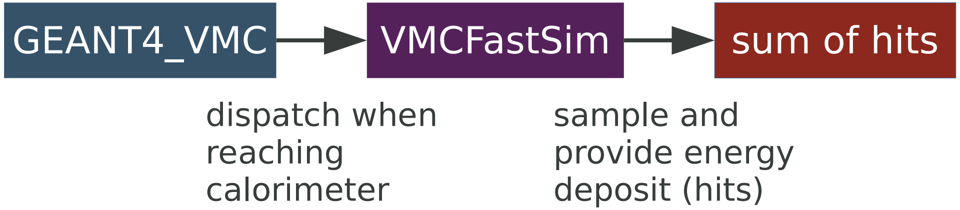
- derive realistic parametrization
- parametrization for different energies and particles
- provide energy per sensitive layer
- set incident particle to the boundary of the calorimeter to be transported further (including potential magnetic field interactions)
- ...

... however

- has been shown that new fast simulation classes can be derived and used
- have functioning workflow
  → can transfer tracks among full and fast simulation
- dummy fast simulation is ~35 times faster
- has been shown that we can directly produce hits instead of steps
  → flexibility of generic fast simulation class

1. derive distribution of total energy deposit from GEANT4
2. fit normal distribution [**proof of principle!**]
3. pass fitted distribution to FastSim
4. FastSim samples total energy deposits from fitted distribution

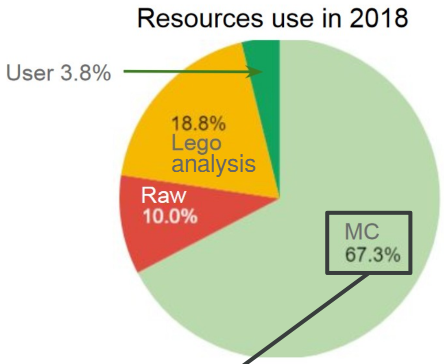




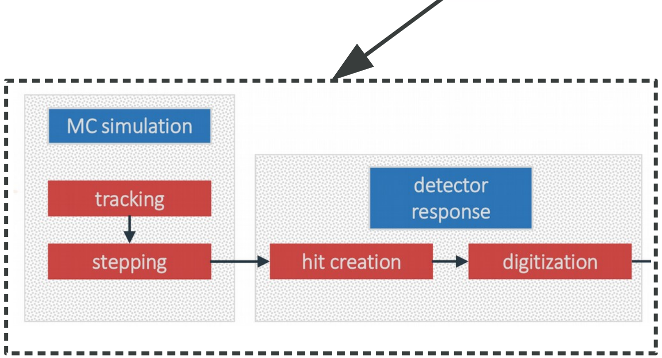
Setting the scene, MC simulation in ALICE

- 2/3 of computing resources are dedicated to MC simulation, all full simulation
- expected up to 100 more data in Runs 3 and 4

  → similar factor required in simulation



cannot cover this with current usage of full simulation



directions:

- general workflow/framework optimization
- full simulation optimization
- reduce/review need for (full) simulation
- embedding techniques

  [approaches presented and discussed by Sandro]
- fast simulation approaches