Using multiple engines in the Virtual Monte Carlo package

Andreas Morsch[a], Benedikt Volkel[a,b], Ivana Hřivnáčová[c], Jan Fiete Grosse-Oetringhaus[a], Sandro Wenzel[a]

[a]CERN,[b]Ruprecht-Karls-Universitaet Heidelberg,
[c]Institut de Physique Nucléaire (IPNO), Université Paris-Sud, CNRS-IN2P3

CHEP 2019, Adelaide, 05.11.2019

(Track 2 – Offline Computing)

SPONSORED BY THE

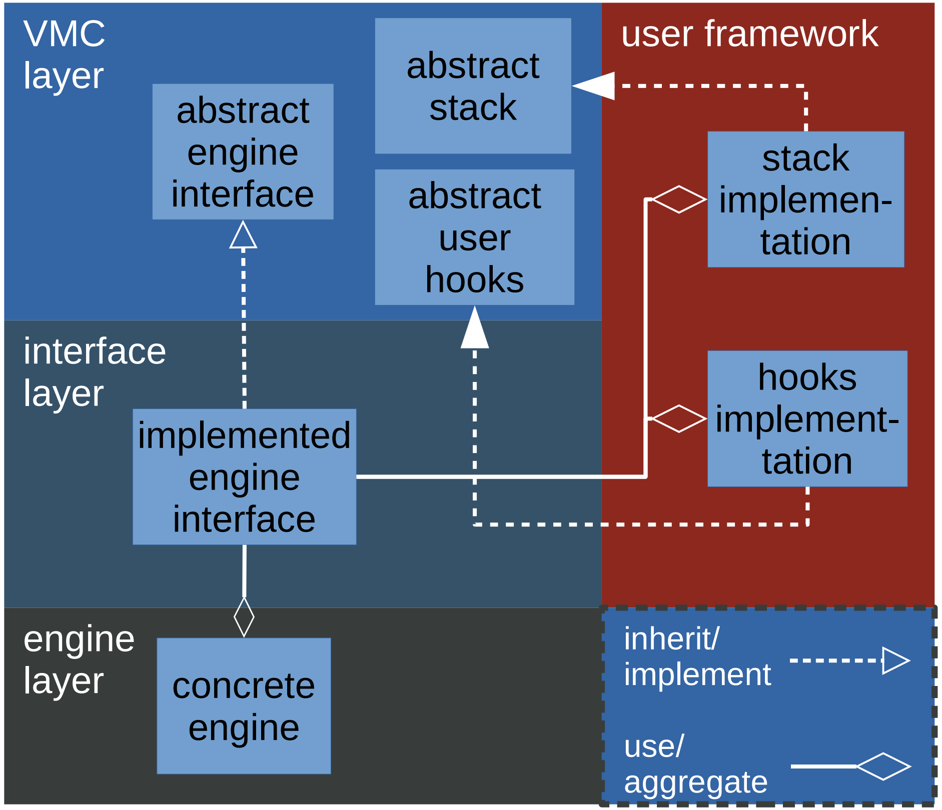
Federal Ministry of Education and Research

CHAPTER I

Virtual **M**onte **C**arlo – how it used to be

VMC how it used to be

- abstract / unified interface to run detector simulation with different engines
  [such as GEANT3, GEANT4]
- one set of user hooks serves for any engine
  [e.g. stepping, begin / end of event, wrapped in one class derived from `TVirtualMCApplication`]
- one user stack implementation serves for any engine
  [class derived from `TVirtualMCStack`]
- 3 main interfaces, via
  1) `TVirtualMC` (e.g. via static `TVirtualMC::Instance()`)
  2) any method of the MCApplication
  3) user stack

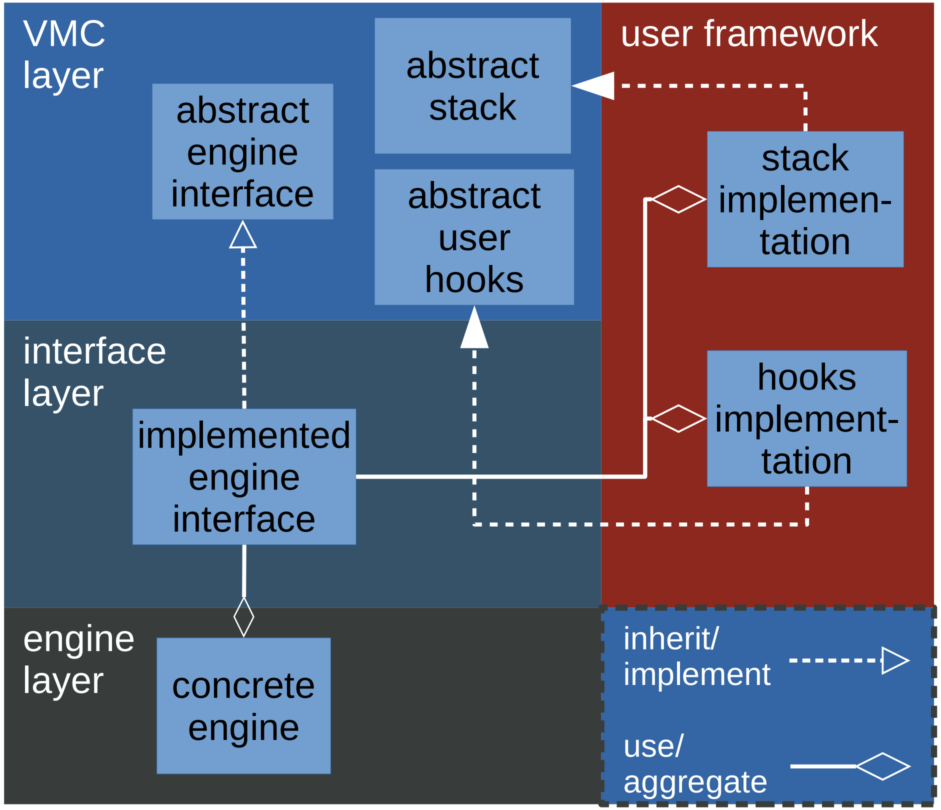


schematic of dependencies and interplay between VMC, user framework and engine backend

VMC how it used to be

- abstract / unified interface to run detector simulation with different engines
  [such as GEANT3, GEANT4]
- one set of user hooks serves for any engine
  [e.g. stepping, begin / end of event, wrapped in one class derived from `TVirtualMCApplication`]
- one user stack implementation serves for any engine
  [class derived from `TVirtualMCStack`]
- 3 main interfaces, via
  1) `TVirtualMC` (e.g. via static `TVirtualMC::Instance()`)
  2) any method of the MCApplication
  3) user stack

limitation of running only a single engine



schematic of dependencies and interplay between VMC, user framework and engine backend

Development goals

- overcome **limitation of running only one** simulation engine
- **allow partitioning** events among multiple different engines
  - e.g. use detailed GEANT4 simulation where necessary and use GEANT3 when less accuracy is already enough but full simulation is still desired
- **more freedom** for the user to inject his / her own VMC implementation
  - custom fast simulation to work with GEANT3 and GEANT4 on VMC level
  - complex / re-usable tasks neither suited for belonging to the stack nor to the application
- enable and test **interplay** of different engines

CHAPTER II

running multiple engines

Mixing multiple engines

vanilla sampling calorimeter to
demonstrate mixing of engines

sensitive layer

passive layer

n particles of specific type
and energy (here: electrons)

Mixing multiple engines

vanilla sampling calorimeter to demonstrate mixing of engines

sensitive layer
passive layer

n particles of specific type and energy (here: electrons)

simulation scenarios

sensitive (GAPX)	passive (ABSO)
GEANT3	
GEANT4	
GEANT4	**GEANT3**

- in mixed scenario
  - keep detailed GEANT4 simulation of sensitive layers
  - use GEANT3 for passive layers

Mixing multiple engines

vanilla sampling calorimeter to demonstrate mixing of engines

sensitive layer — passive layer

n particles of specific type and energy (here: electrons)

TMCManager

```
...
void TransferTrack(Int_t targetEngineId)
...
```

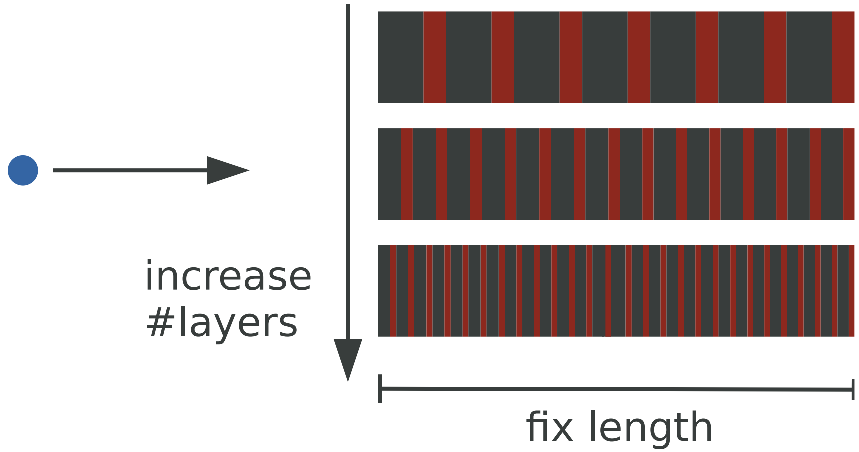
simulation scenarios

sensitive (GAPX)	passive (ABSO)
GEANT3	
GEANT4	
GEANT4	**GEANT3**

- in mixed scenario
  - keep detailed GEANT4 simulation of sensitive layers
  - use GEANT3 for passive layers

Mixing multiple engines (continued)

vanilla sampling calorimeter to demonstrate mixing of engines

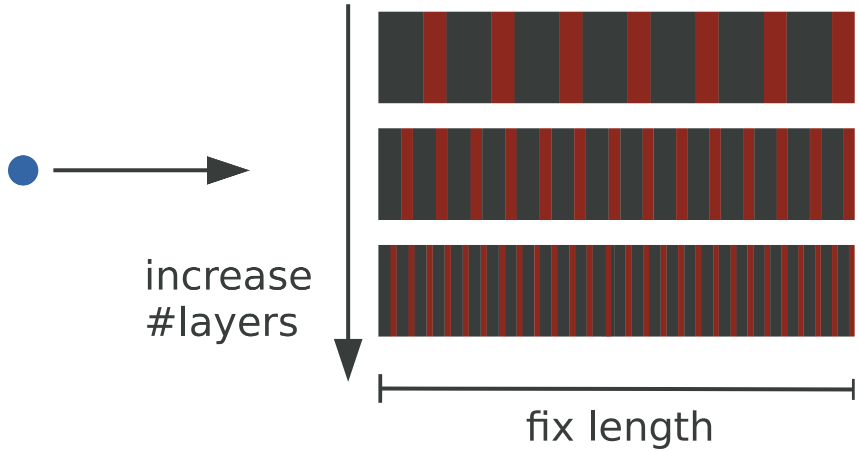


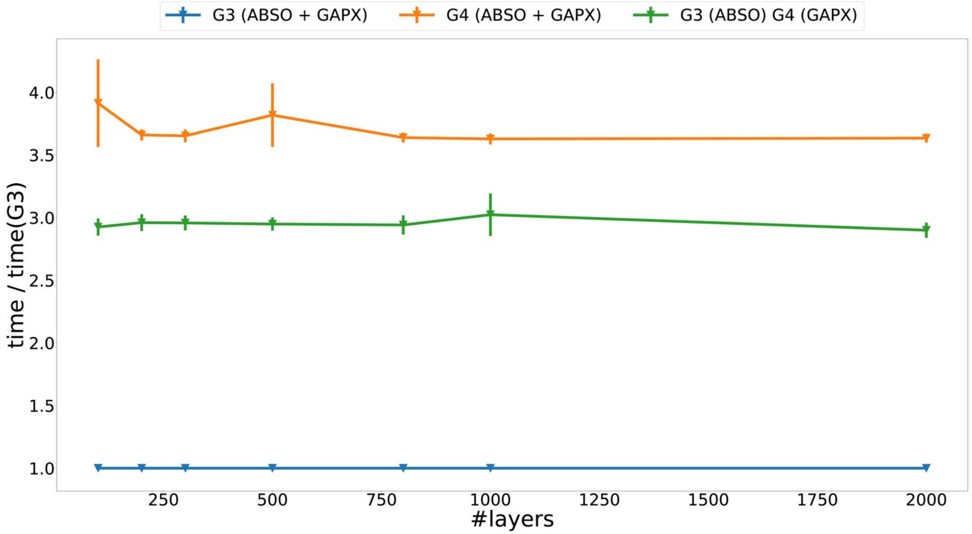
TMCManager

```
...
void TransferTrack(Int_t targetEngineId)
...
```

Mixing multiple engines (continued)

vanilla sampling calorimeter to demonstrate mixing of engines





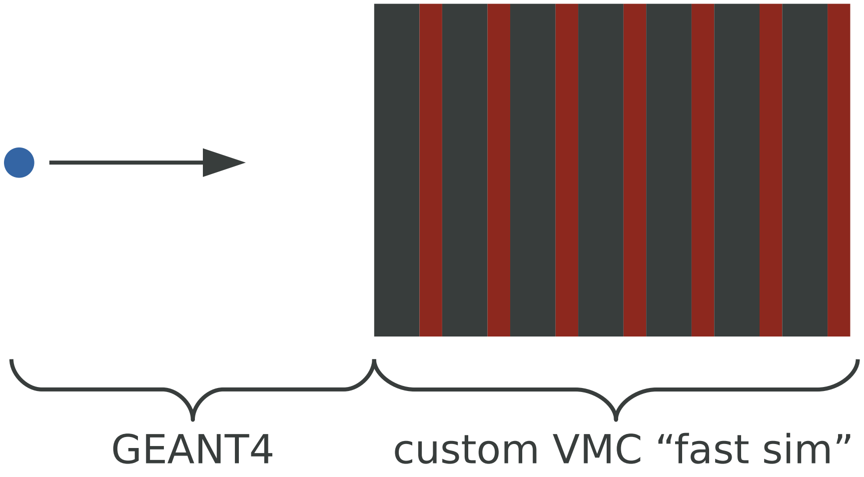
- time elapsed **relative to G3**
- simulation more slowly using **GEANT4 only**
- **speed-up is possible by mixing engines**
- **no scaling overhead with number of track transfers**

TMCManager

```
...
void TransferTrack(Int_t targetEngineId)
...
```

A custom VMC “fast simulation”

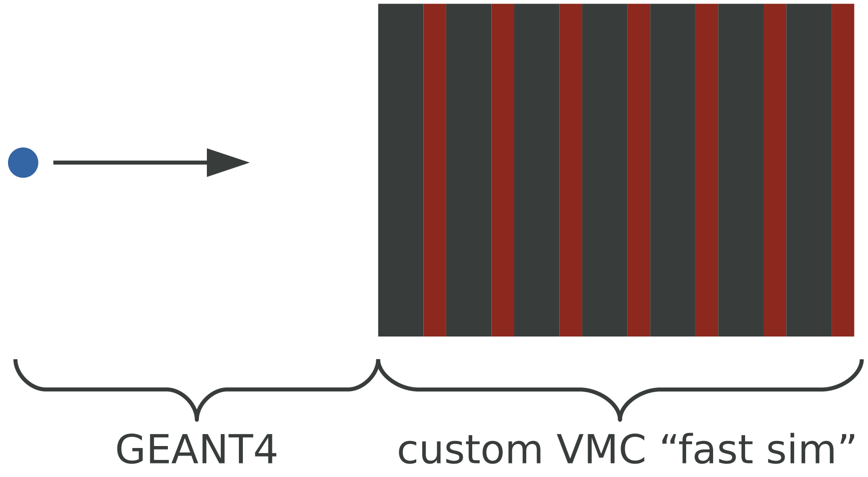
vanilla sampling calorimeter to demonstrate mixing of engines



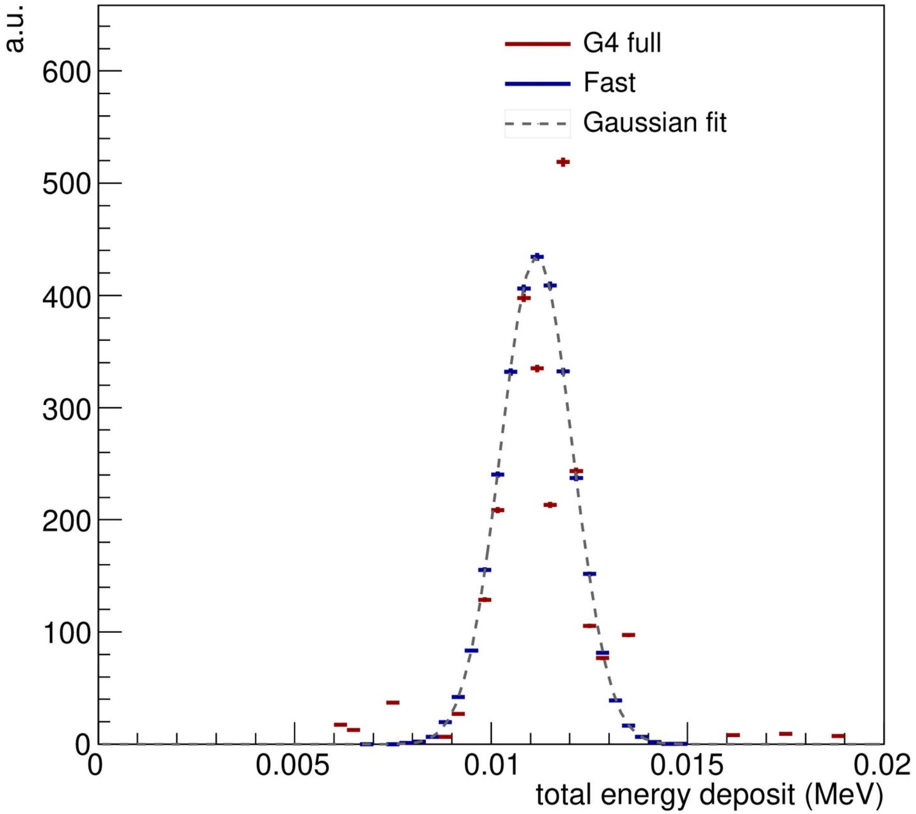
- again a mixed scenario

A custom VMC "fast simulation"

vanilla sampling calorimeter to demonstrate mixing of engines



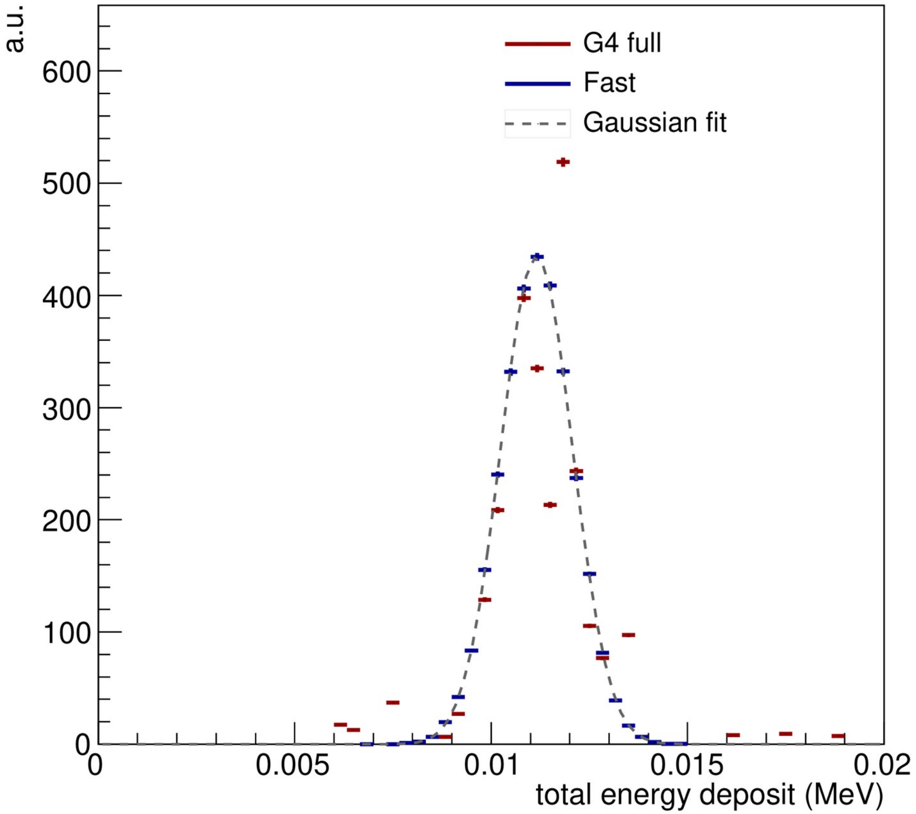
- again a mixed scenario
- "fast sim" draws **total energy deposit** from fitted distribution



disclaimer: proof-of-concept

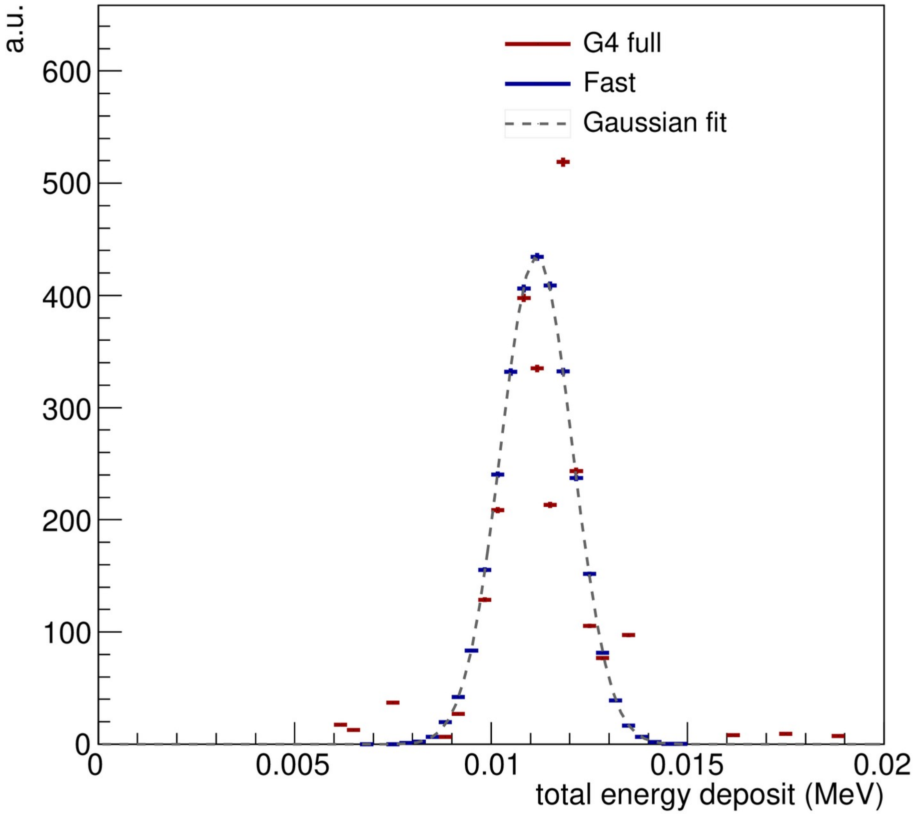
A custom VMC “fast simulation”

- provide `VMCFastSim` class
  - only 2 methods to be implemented by the user
    1) `VMCFastSim::Process()`
    2) `VMCFastSim::Stop()`
- use `VMCFastSim` to implement a “`FastShower`” class
- code at
  - https://github.com/benedikt-voelkel/VMCFastSim
  - https://github.com/benedikt-voelkel/FastShower



A custom VMC “fast simulation”

- provide `VMCFastSim` class
  - only 2 methods to be implemented by the user
    1) `VMCFastSim::Process()`
    2) `VMCFastSim::Stop()`
- use `VMCFastSim` to implement a “`FastShower`” class
- code at
  - https://github.com/benedikt-voelkel/VMCFastSim
  - https://github.com/benedikt-voelkel/FastShower



actual fast simulation might be done in a few lines

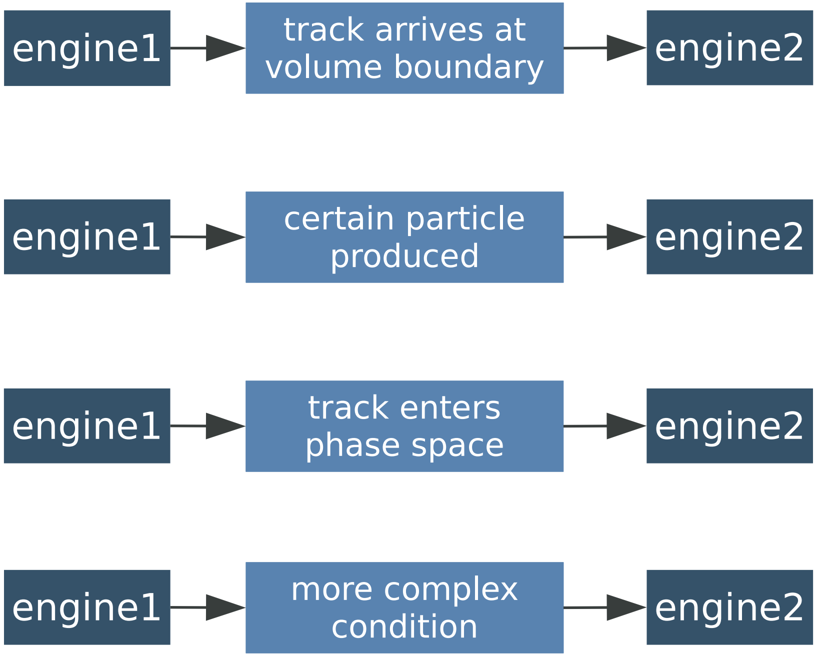
```
bool FastShower::Process() {
  if(GetCurrentParticle()->GetPDGCode() == 2212) {
    mStoreHit(mDistribution(mGenerator));
  }
  // ...
}
```

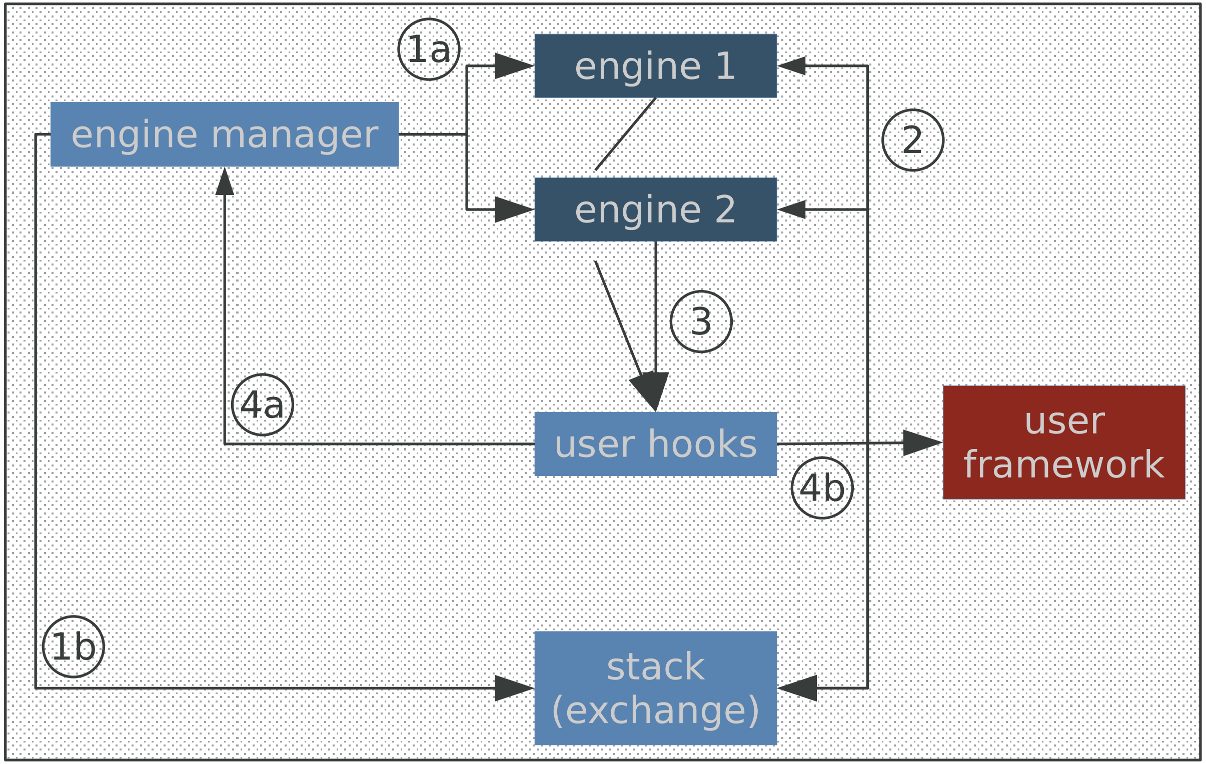
CHAPTER III

technical details – below the hood

Sketching the implementation

partition simulation among multiple different engines





New classes and extensions

ALICE UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

TMCManager

```
void ForwardTrack(Int_t toBeDone, Int_t trackId,
                  Int_t parentId,
                  TParticle* particle)
void TransferTrack(Int_t targetEngineId)
...
```

- singleton object
- needs to be explicitly requested by the user during construction of the `UserApplication`
  [keep runtime overhead as small as possible]
- VMCs are
  - owned by the manager
  - automatically registered when instantiated
- handles
  - communication between engines
  - pausing and resuming engines
  - transferring particles / tracks between engines

New classes and extensions

TMCManager

```
void ForwardTrack(Int_t toBeDone, Int_t trackId,
                  Int_t parentId,
                  TParticle* particle)
void TransferTrack(Int_t targetEngineId)
...
```

TMCManagerStack

A concrete implementation of TVirtualMCStack *providing the interfaces accordingly for the usage and communication with the* TMCManager.

TVirtualMCApplication

```
void RequestManager()
```

```
TMCManager* fMCManager
```

- singleton object
- needs to be explicitly requested by the user during construction of the `UserApplication`
  [keep runtime overhead as small as possible]
- VMCs are
  - owned by the manager
  - automatically registered when instantiated
- handles
  - communication between engines
  - pausing and resuming engines
  - transferring particles / tracks between engines

Conclusion

- VMC package **enhanced** to allow usage of multiple engines and to **overcome previous limitations**
  - mix full simulation engines, e.g. GEANT3 and GEANT4
  - inject custom user VMC, e.g. some kind of fast simulation
- user is **free to decide how to partition** simulation between engines [geometry, particle type, phase space etc.]
- former run-mode (single engine) **fully preserved**
- **no runtime overhead** observed when moving tracks between engines
- **implementation details wrapped** into `TMCManager` and `TMCManagerStack`
- **example available in GEANT4_VMC package, E03c**

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Thanks for your attention

BACKUP

Deployment overview (thanks to I. Hřivnáčová)

- crucial enhancements have been explained (more can be found in the BACKUP)
- example using multiple engines implemented along with GEANT4_VMC: **E03c**
  - a `diff` (e.g. to E03a) nicely shows that just a few modifications in the user code are necessary
- VMC now distributed via its own repository
- ROOT supports building with or without built-in VMC
  [ROOT version >= 6.18.00]
- releases
  - VMC, tag 1.0
    https://github.com/vmc-project/vmc
  - GEANT3_VMC, tag 3.0
    https://github.com/vmc-project/geant3
  - GEANT4_VMC, tag 5.0
    https://github.com/vmc-project/geant4_vmc
- new VMC documentation can be found at https://vmc-project.github.io

see also poster contribution 322 by I. Hřivnáčová

New classes and extensions (implementation examples)

TMCManager

```
void SetUserStack(TVirtualMCStack* userStack)
void ForwardTrack(Int_t toBeDone, Int_t trackId,
                  Int_t parentId,
                  TParticle* particle)
void TransferTrack(Int_t targetEngineId)
template <typename F> Apply(F f)
template <typename F> Init(F f)
void Run(Int_t nEvents)
void ConnectEnginePointer(TVirtualMC*& mc)
TVirtualMC* GetCurrentEngine()
```

- singleton object
- needs to be explicitly requested by the user during construction of the `UserApplication`
  [keep runtime overhead as small as possible]
- VMCs are
  - owned by the manager
  - automatically registered when instantiated
- handles
  - communication between engines
  - pausing and resuming engines
  - transferring particles / tracks between engines

New classes and extensions (implementation examples)

TMCManager
`void SetUserStack(TVirtualMCStack* userStack)`
`void ForwardTrack(Int_t toBeDone, Int_t trackId, Int_t parentId, TParticle* particle)`
`void TransferTrack(Int_t targetEngineId)`
`template <typename F> Apply(F f)`
`template <typename F> Init(F f)`
`void Run(Int_t nEvents)`
`void ConnectEnginePointer(TVirtualMC*& mc)`
`TVirtualMC* GetCurrentEngine()`

- user is still owner of constructed `TParticle` objects and numbering
- should be called in `UserStack::PushTrack(...)`
- additional last argument might be the target engine ID

```
void Ex03MCStack::PushTrack(Int_t toBeDone, Int_t parent, ..., Int_t& ntr, ...) {
  // TParticle construction yielding “particle”
  // define track ID
  ntr = GetNtrack() - 1;
  if(auto mgr = TMCManager::Instance()) {
    mgr->ForwardTrack(toBeDone, ntr, parent, particle);
  }
  // further implementation
}
```

New classes and extensions (implementation examples)

ALICE UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

TMCManager

```
void SetUserStack(TVirtualMCStack* userStack)
void ForwardTrack(Int_t toBeDone, Int_t trackId,
                  Int_t parentId,
                  TParticle* particle)
void TransferTrack(Int_t targetEngineId)
template <typename F> Apply(F f)
template <typename F> Init(F f)
void Run(Int_t nEvents)
void ConnectEnginePointer(TVirtualMC*& mc)
TVirtualMC* GetCurrentEngine()
```

- call e.g. in `UserApplication::Stepping()`
- interrupts transport and transfers particle to target engine stack [preserves momentum and geometry information]
- decide based on geometry, particle phase space / type etc.

```
void Ex03MCApplication::Stepping() {
  // ...
  Int_t targetId = -1;
  if(fMC->GetId() == 0 && strcmp(fMC->GetCurrentVol(), “ABSO”) == 0) {
    targetId = 1;
  } else if(fMC->GetId() == 1 && strcmp(fMC->GetCurrentVol(), “GAPX”) == 0) {
    targetId = 0;
  }
  // ...
  fMCManager->TransferTrack(targetId);
}
```

New classes and extensions (implementation examples)

TMCManager

```
void SetUserStack(TVirtualMCStack* userStack)
void ForwardTrack(Int_t toBeDone, Int_t trackId,
                  Int_t parentId,
                  TParticle* particle)
void TransferTrack(Int_t targetEngineId)
template <typename F> Apply(F f)
template <typename F> Init(F f)
void Run(Int_t nEvents)
void ConnectEnginePointer(TVirtualMC*& mc)
TVirtualMC* GetCurrentEngine()
```

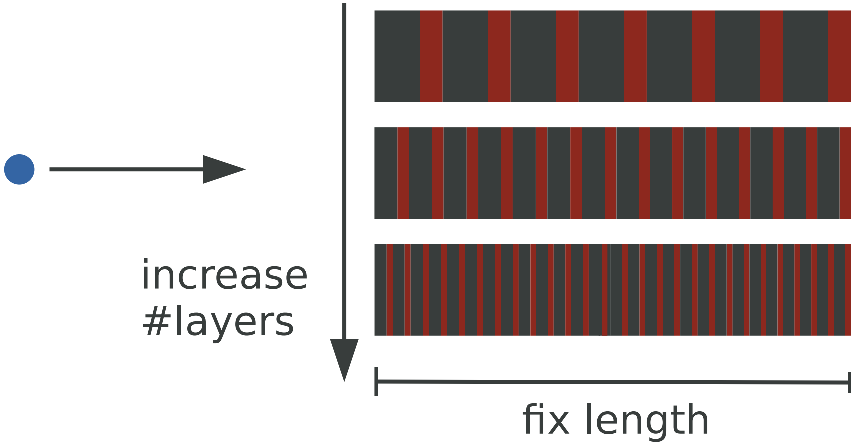
- the type `F` is assumed to implement `()` taking a `TVirtualMC` as an argument
- `f` is applied to all registered engines
- passed pointer will be kept up-to-date

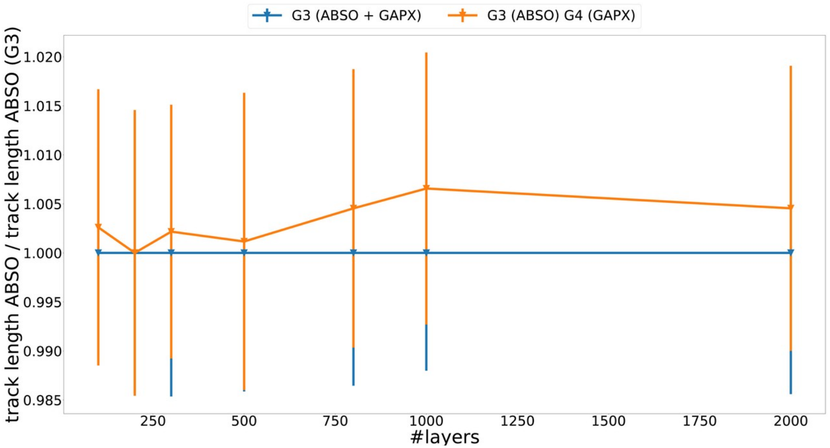
```
void Ex03MCApplication::InitMC(
 std::initializer_list<const char*> setupMacros) {
  // ...
  fMCManager->Init([this](TVirtualMC* mc) {
    mc->SetRootGeometry();
    mc->SetMagField(fMagField);
    mc->Init();
    mc->BuildPhysics();
  });
  // ...
}
```

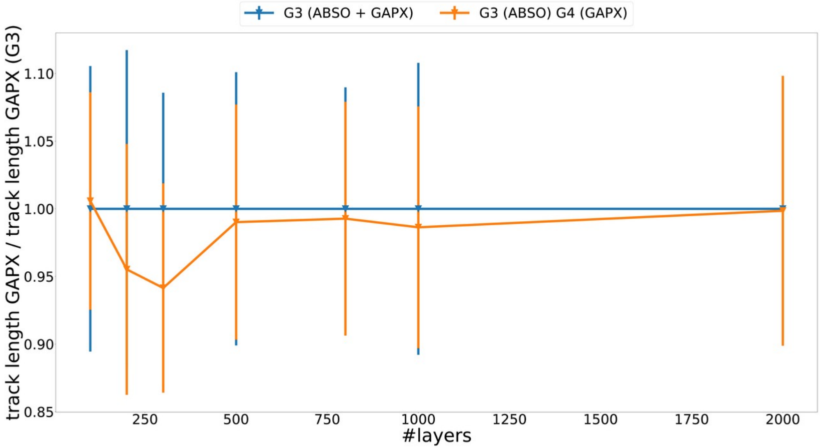
```
Ex03DetectorConstruction::Ex03DetectorConstruction() {
  // ...
  if(auto mgr = TMCManager::Instance()) {
    mgr->ConnectEnginePointer(fMC);
  }
  // ...
}
```

Mixing multiple engines (continued)

vanilla sampling calorimeter to demonstrate mixing of engines







- track length in ABSO (top) **relative to G3**
- track length in GAPX (top) **relative to G3**
- no cut optimisation done per engine yet, however, simulated track lengths of same order of magnitude