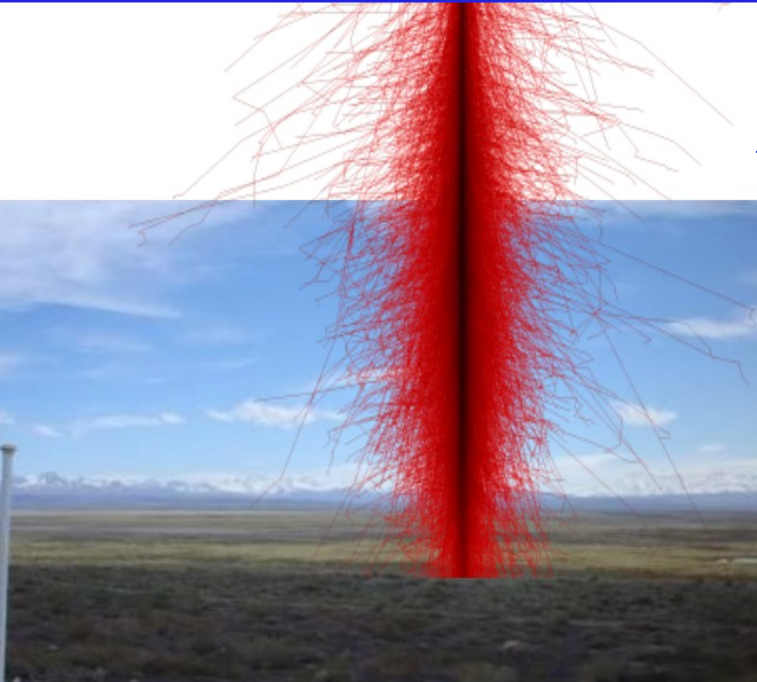
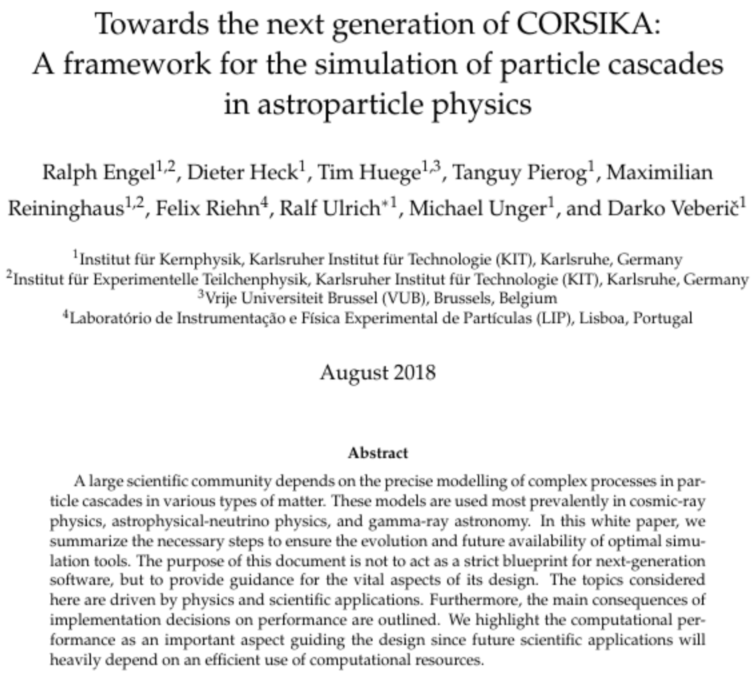
CORSIKA Upgrade
Simulating particle cascades for astroparticle physics



Ralf Ulrich
on behalve of the authors of arXiv:1808.08226 (in particular Maximilan Reininghaus)
and the particpants of the *Next-generation CORSIKA Workshop*

CORSIKA upgrade

- Cornerstone for the scientific work of many experimental collaborations
- Excellent understanding of particle cascades is important for almost all aspects in astroparticle physics
- There are existing limitations that must be overcome
- Need a new and modern framework that allows our field to tackle physics questions over the next ~3 decades
  - New large-scale detectors, new fundamental physics



Towards the next generation of CORSIKA: A framework for the simulation of particle cascades in astroparticle physics

Ralph Engel[1,2], Dieter Heck[1], Tim Huege[1,3], Tanguy Pierog[1], Maximilian Reininghaus[1,2], Felix Riehn[4], Ralf Ulrich[*1], Michael Unger[1], and Darko Veberič[1]

[1]Institut für Kernphysik, Karlsruher Institut für Technologie (KIT), Karlsruhe, Germany
[2]Institut für Experimentelle Teilchenphysik, Karlsruher Institut für Technologie (KIT), Karlsruhe, Germany
[3]Vrije Universiteit Brussel (VUB), Brussels, Belgium
[4]Laboratório de Instrumentação e Física Experimental de Partículas (LIP), Lisboa, Portugal

August 2018

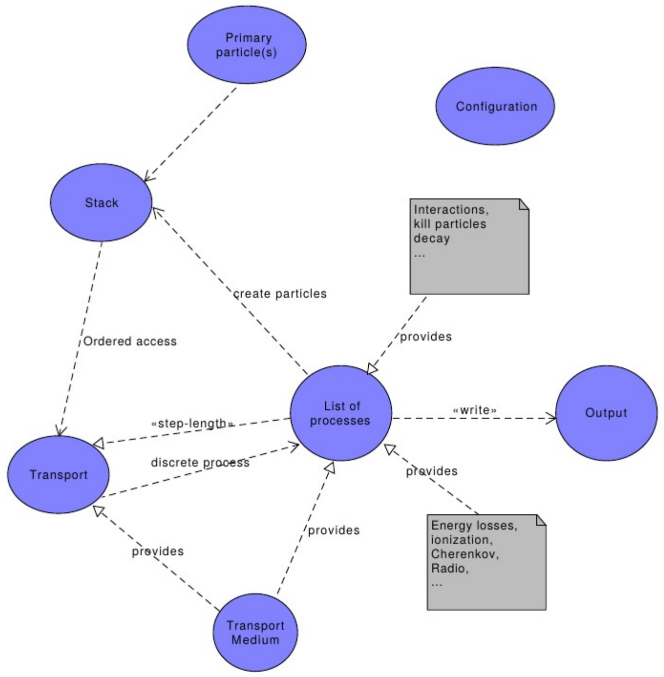
Abstract

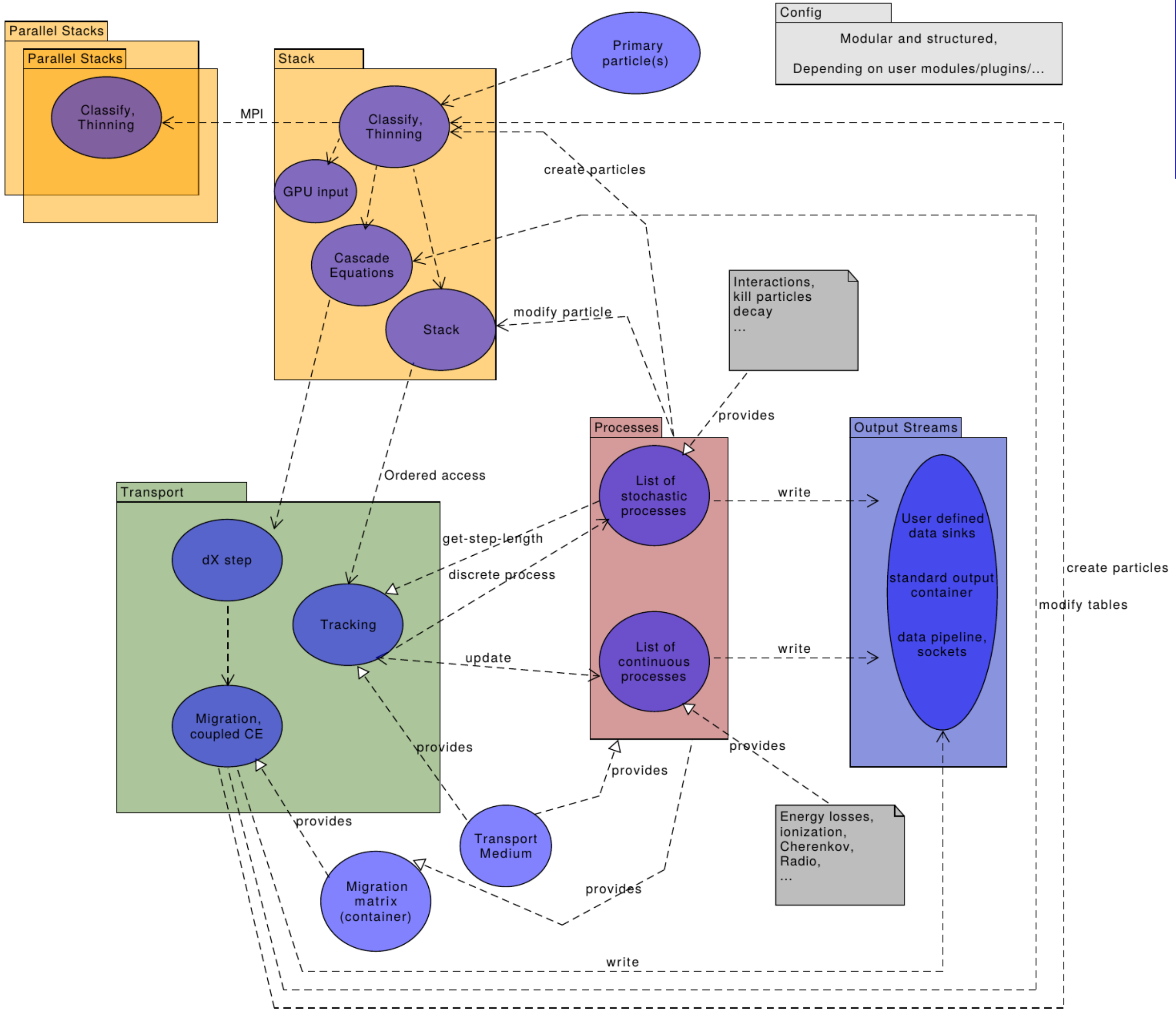
A large scientific community depends on the precise modelling of complex processes in particle cascades in various types of matter. These models are used most prevalently in cosmic-ray physics, astrophysical-neutrino physics, and gamma-ray astronomy. In this white paper, we summarize the necessary steps to ensure the evolution and future availability of optimal simulation tools. The purpose of this document is not to act as a strict blueprint for next-generation software, but to provide guidance for the vital aspects of its design. The topics considered here are driven by physics and scientific applications. Furthermore, the main consequences of implementation decisions on performance are outlined. We highlight the computational performance as an important aspect guiding the design since future scientific applications will heavily depend on an efficient use of computational resources.

arxiv.org/abs/1808.08226

Next generation of CORSIKA

- Framework for simulating particle cascade processes
  - modular, flexibel
  - precise, fast
- Fundamental integration of
  - Parallelization
  - GPUs
  - Modularity and flexibility
- Highest quality air shower simulations and complex data analyses




Parallel Stacks
Parallel Stacks
Classify, Thinning
MPI
Stack
Classify, Thinning
GPU input
Cascade Equations
Stack
Primary particle(s)
Config
Modular and structured,
Depending on user modules/plugins/...
create particles
modify particle
Interactions, kill particles decay ...
provides
Ordered access
Processes
List of stochastic processes
List of continuous processes
Output Streams
User defined data sinks
standard output container
data pipeline, sockets
write
write
Transport
dX step
Tracking
Migration, coupled CE
get-step-length
discrete process
update
provides
provides
provides
provides
provides
Transport Medium
Migration matrix (container)
Energy losses, ionization, Cherenkov, Radio, ...
write
create particles
modify tables

Milestones, planning

Requirement for CORSIKA-upgrade: better physics performance than CORSIKA

- **Milestone 0**: July 2018, Workshop at KIT, and white paper
- **Milestone 1**: end of september 2018
  Framework definition, working environment/infrastructure, first documentation
- Milestone 2: end of 2018
  First cascade calculations, w/ simple atmosphere
- Milestone 3: February 2019
  SIBYLL2.3 and UrQMD included and a useful atmosphere model
- Milestone 4:~Summer 2019
  Include E.M. interactions
- Milestone 5: CORSIKA 8.1.0, ~2020
  First full physics (demonstrator) release

Brief introduction to some concepts

- In physics we often think+work within well defined reference frames. We want to map this fact into code and enforce it!
- Help physicist to produce correct algorithms.
- Code as close as possible to natural physics representation.

OK	not OK
567_GeV + 1_TeV	constants::c + 1_m
point1.GetX(showerFrame)	point1.GetX()
particle::GetMass(Sib2Cors(PID::Electron))	particle::GetMass(5)

→ **does simply not compile**

Example, main cascade loop

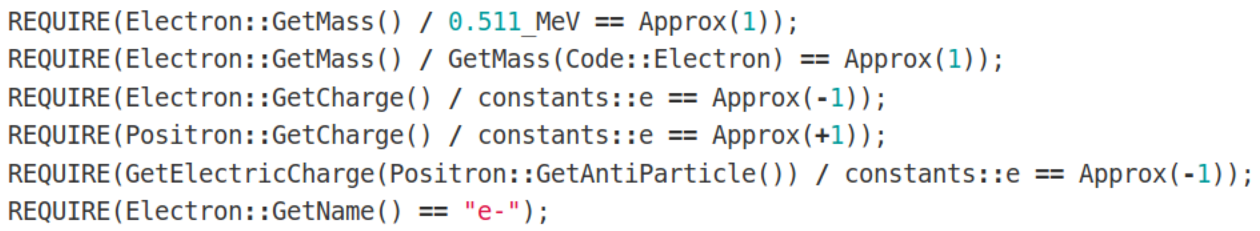
```
void Run() {
  while (!fStack.IsEmpty()) {
    Particle& p = *fStack.GetNextParticle();
    Step(p);
  }
}

void Step(Particle& particle) {
  double nextStep = fProcesseList.MinStepLength(particle);
  fProcesseList.DoContinuous(particle, fStack);
  fProcesseList.DoDiscrete(particle, fStack);
}
```

```
ProcessReport p0(false);
HeitleModel p1;
const auto sequence = p0 + p1;
corsika::stack::super_stupid::SuperStupidStack stack;
corsika::cascade::Cascade EAS(sequence, stack);
stack.NewParticle().SetEnergy(10_TeV);
EAS.Run();
```

Example, particles on stack(s)

```
using namespace corsika::units;
using namespace corsika::stack;

void fill(corsika::stack::super_stupid::SuperStupidStack& s) {
  for (int i = 0; i < 11; ++i) {
    auto p = s.NewParticle();
    p.SetId(corsika::particles::Code::Electron);
    p.SetEnergy(1.5_GeV * i);
  }
}

void read(corsika::stack::super_stupid::SuperStupidStack& s) {
  cout << "found Stack with " << s.GetSize() << " particles. " << endl;
  EnergyType Etot;
  for (auto p : s) {
    Etot += p.GetEnergy();
    cout << "particle: " << p.GetId() << " with " << p.GetEnergy() / 1_GeV << " GeV"
         << endl;
  }
  cout << "Etot=" << Etot << " = " << Etot / 1_GeV << " GeV" << endl;
}
```

Example, particle properties



```
REQUIRE(Electron::GetMass() / 0.511_MeV == Approx(1));
REQUIRE(Electron::GetMass() / GetMass(Code::Electron) == Approx(1));
REQUIRE(Electron::GetCharge() / constants::e == Approx(-1));
REQUIRE(Positron::GetCharge() / constants::e == Approx(+1));
REQUIRE(GetElectricCharge(Positron::GetAntiParticle()) / constants::e == Approx(-1));
REQUIRE(Electron::GetName() == "e-");
```

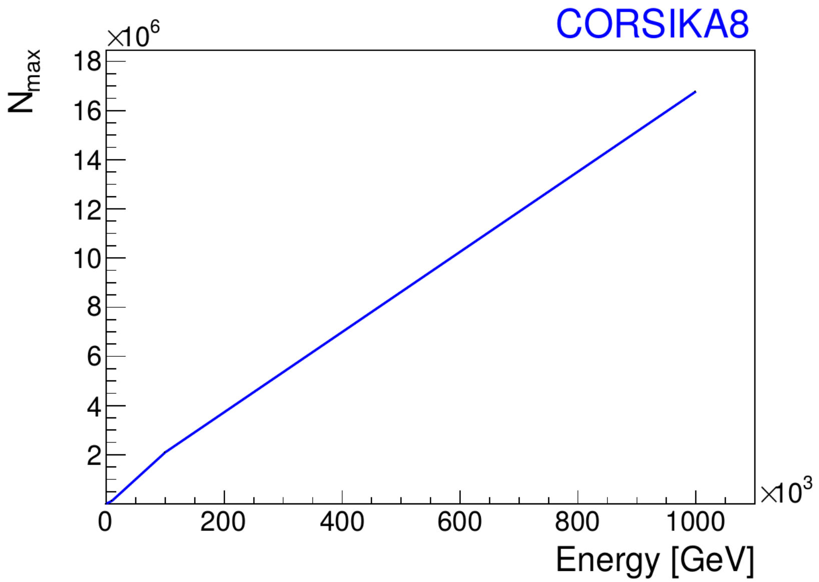
- Particle properties are automatically generated from Pythia8 ParticleData.xml file.

Examples, geometry

```
// define the root coordinate system
CoordinateSystem root;

// another CS defined by a translation relative to the root CS
CoordinateSystem cs2 = root.translate({0_m, 0_m, 1_m});

// rotations are possible, too; parameters are axis vector and angle
CoordinateSystem cs3 = root.rotate({1_m, 0_m, 0_m}, 90 * degree_angle);

// now let's define some geometrical objects:
Point const p1(root, {0_m, 0_m, 0_m}); // the origin of the root CS
Point const p2(cs2, {0_m, 0_m, 0_m});  // the origin of cs2

Vector<length_d> const diff =
    p2 -
    p1; // the distance between the points, basically the translation vector given above
auto const norm = diff.squaredNorm(); // squared length with the right dimension

// print the components of the vector as given in the different CS
std::cout << "p2-p1 components in root: " << diff.GetComponents(root) << std::endl;
std::cout << "p2-p1 components in cs2: " << diff.GetComponents(cs2)
          << std::endl; // by definition invariant under translations
std::cout << "p2-p1 components in cs3: " << diff.GetComponents(cs3)
          << std::endl; // but not under rotations
std::cout << "p2-p1 norm^2: " << norm << std::endl;

Sphere s(p1, 10_m); // define a sphere around a point with a radius
std::cout << "p1 inside s:  " << s.isInside(p2) << std::endl;
```

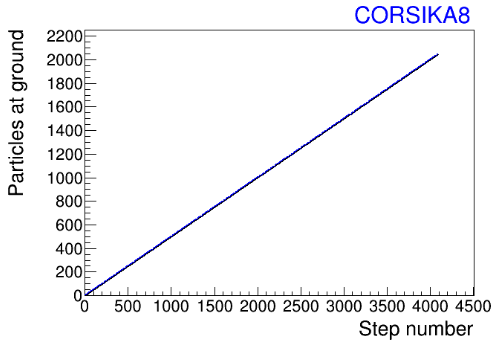
Heitler model (equal energy splitting)

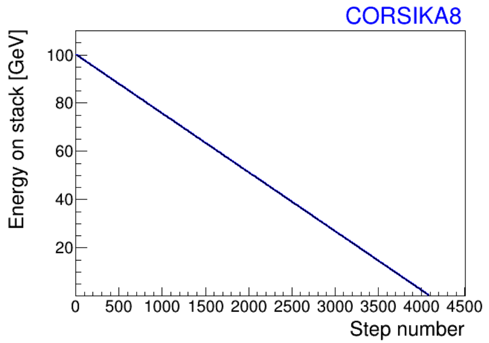
```
class ProcessSplit : public corsika::process::BaseProcess<ProcessSplit> {
public:
  template <typename Particle, typename Stack>
  void DoDiscrete(Particle& p, Stack& s) const {
    EnergyType E = p.GetEnergy();
    if (E < 1_GeV) {
      p.Delete();
      fCount++;
    } else {
      p.SetEnergy(E / 2);
      s.NewParticle().SetEnergy(E / 2);
    }
  }
```

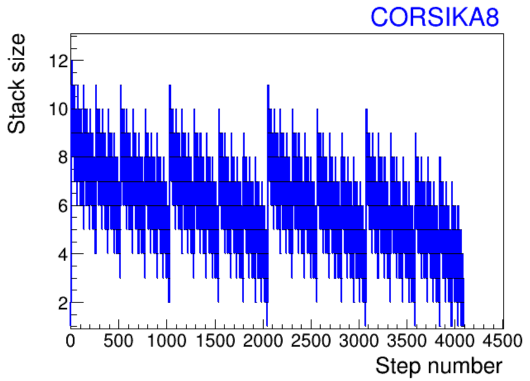


Diagnostics of the cascade process, E_0=100GeV

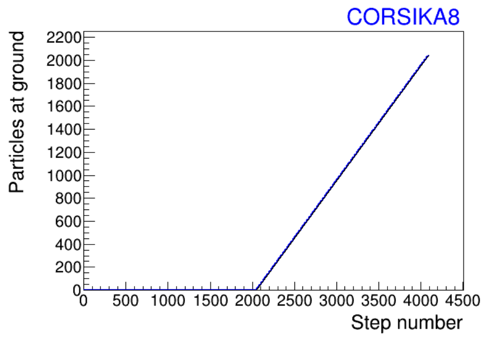
Use lowest energy particle for next step in cascade:

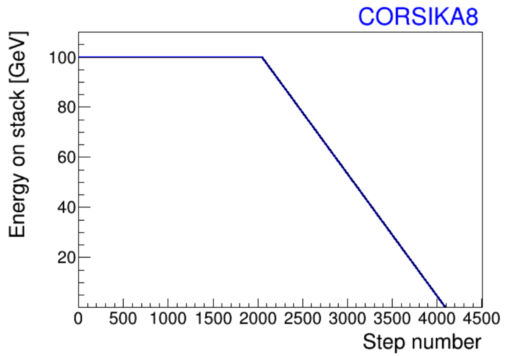


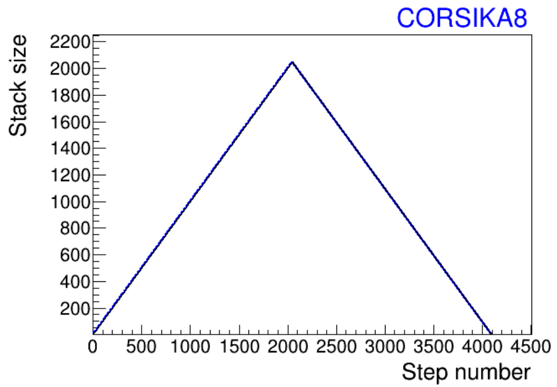




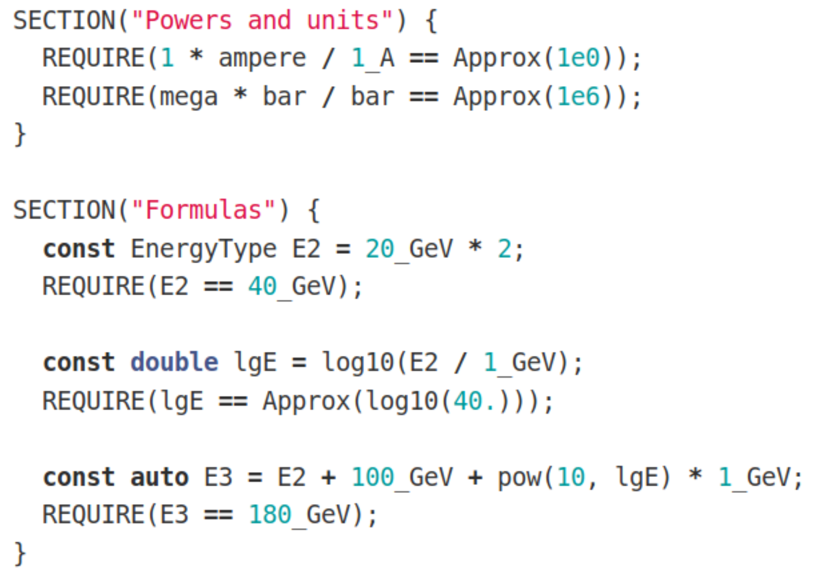
Use highest energy particle for next step in cascade:







Example, pysical units



```
SECTION("Powers and units") {
  REQUIRE(1 * ampere / 1_A == Approx(1e0));
  REQUIRE(mega * bar / bar == Approx(1e6));
}

SECTION("Formulas") {
  const EnergyType E2 = 20_GeV * 2;
  REQUIRE(E2 == 40_GeV);

  const double lgE = log10(E2 / 1_GeV);
  REQUIRE(lgE == Approx(log10(40.)));

  const auto E3 = E2 + 100_GeV + pow(10, lgE) * 1_GeV;
  REQUIRE(E3 == 180_GeV);
}
```

- Thus, this fails at compile time already:

  auto alpha = 90. * EeV / meter;

  EnergyType E1 = 10_GeV + alpha;

Discussion right now:
units → units::si and units::hep

Atmosphere

- Environment::GetVolumeId(point)
- Environment::GetVolumeBoundary(trajectory)
- Environment::GetTargetParticle(point)
- Environment::GetDensity(point)
- Environment::GetIntegratedDensity(trajectory)
- Environment::GetRefractiveIndex(point)
- Environment::GetTemperature(point)
- Environment::GetHumidity(point)
- Environment::GetMagneticField(point)
- Environment::GetElectricField(point)

Dependencies (right now: eigen3)

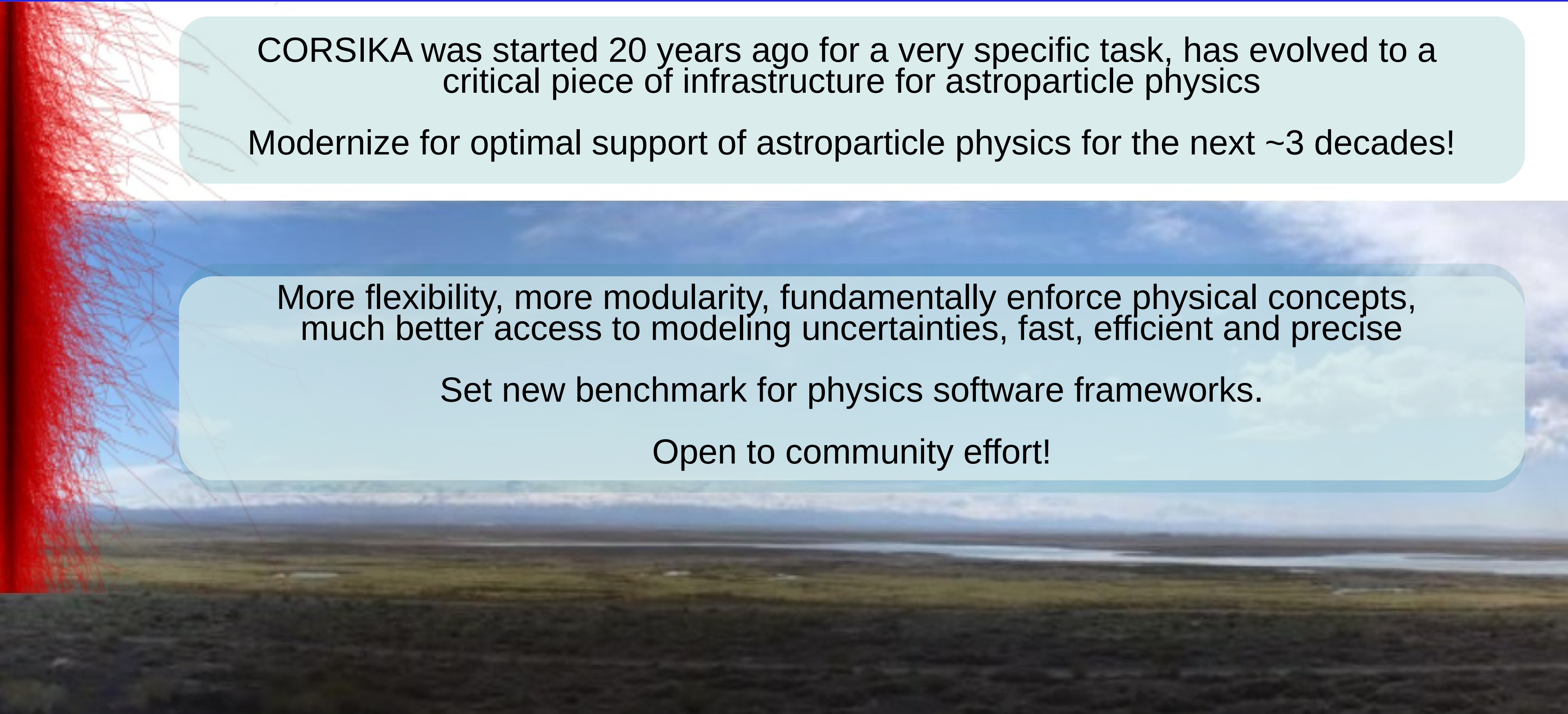
- C++17 compiler.
- CMake build system.
- git [for development].
- doxygen [for development].
- presumably boost for yaml and xml, histograms, file system access, command-line options, light-weight configuration parsers (property tree), random numbers, etc.
- HDF5 and/or ROOT for data storage [at least one of both required]
- PyBind11 for bindings to Python.
- HepMC as generic interface, also for exotics [optional].
- In order to generate random numbers, we will use standardized interfaces and established methods. For testing purposes, the possibility to exchange the random-number engine should be relatively easy. No homegrown generators and only well established, checked, and vetted methods for generating random numbers should be used, likelily provided by boost as well.
- Eigen3 for linear algebra.
- catch2 for unit tests.
- PhysUnits for units.

Impact on community

- **There is opportunity to actively contribute**
  - Shape parts of the project for the future, and for specific applications
  - Get in contact:
    - Write to me, connect to corsika-devel@lists.kit.edu, and to gitlab.ikp.kit.edu

- **Some goals and standards**
  - Make it really hard/impossible to produce wrong physics and results
  - Make complete use of available optimization and high-performance concepts
  - High standards on code, combined with excellent documentation
  - Extensive use of testing, automation and unit testing
  - Direct access to high-level validation
  - Very low-level enforcement of physical concepts on the level of code compilation

Summary



CORSIKA was started 20 years ago for a very specific task, has evolved to a critical piece of infrastructure for astroparticle physics

Modernize for optimal support of astroparticle physics for the next ~3 decades!

More flexibility, more modularity, fundamentally enforce physical concepts, much better access to modeling uncertainties, fast, efficient and precise

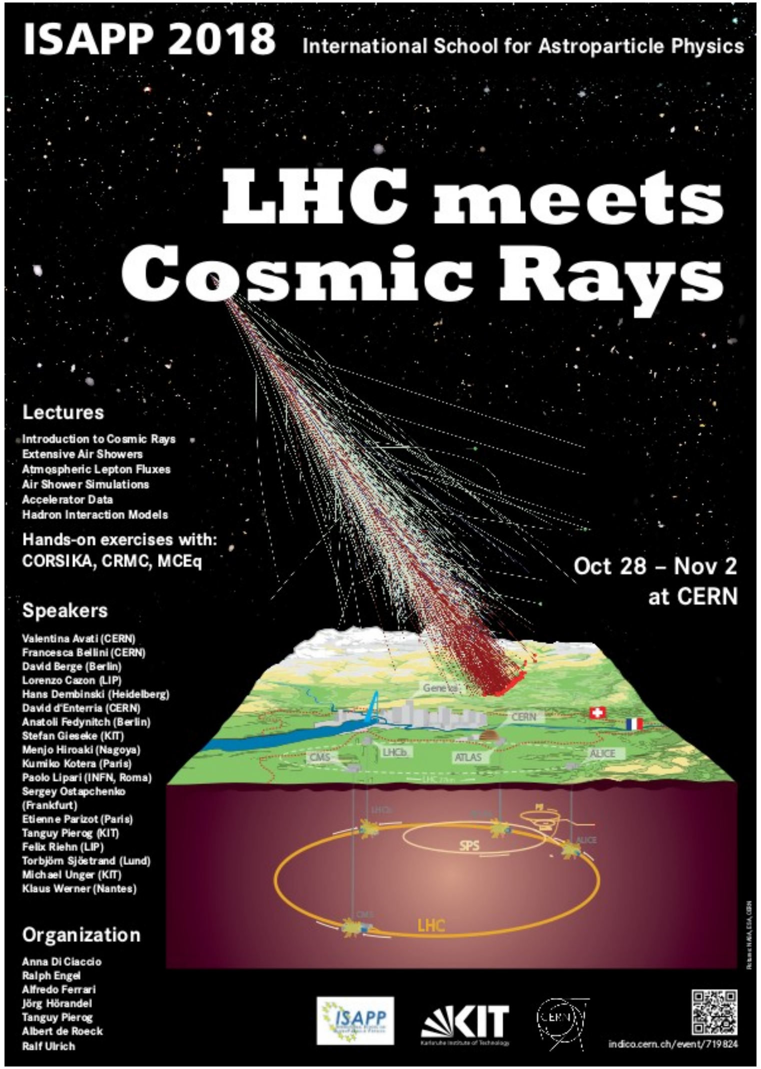
Set new benchmark for physics software frameworks.

Open to community effort!

CORSIKA and ISAPP school 2018 at CERN

Comprehensive school about air shower modeling and related physics.

https://indico.cern.ch/event/719824/



ISAPP 2018 International School for Astroparticle Physics

LHC meets Cosmic Rays

Lectures

Introduction to Cosmic Rays
Extensive Air Showers
Atmospheric Lepton Fluxes
Air Shower Simulations
Accelerator Data
Hadron Interaction Models

Hands-on exercises with:
CORSIKA, CRMC, MCEq

Oct 28 – Nov 2
at CERN

Speakers

Valentina Avati (CERN)
Francesca Bellini (CERN)
David Berge (Berlin)
Lorenzo Cazon (LIP)
Hans Dembinski (Heidelberg)
David d'Enterria (CERN)
Anatoli Fedynitch (Berlin)
Stefan Gieseke (KIT)
Menjo Hiroaki (Nagoya)
Kumiko Kotera (Paris)
Paolo Lipari (INFN, Roma)
Sergey Ostapchenko (Frankfurt)
Etienne Parizot (Paris)
Tanguy Pierog (KIT)
Felix Riehn (LIP)
Torbjörn Sjöstrand (Lund)
Michael Unger (KIT)
Klaus Werner (Nantes)

Organization

Anna Di Ciaccio
Ralph Engel
Alfredo Ferrari
Jörg Hörandel
Tanguy Pierog
Albert de Roeck
Ralf Ulrich

indico.cern.ch/event/719824