Power user wish-list for Next-Gen CORSIKA

Hans Dembinski

Max-Planck-Institute für Kernphysik, Heidelberg

Goals for Next-Gen CORSIKA

- Extensible
- Easier to connect to other tools
- Easier to use

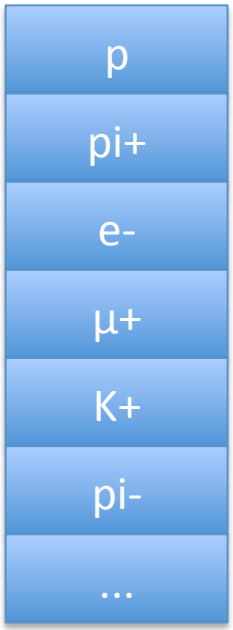
Python Frontend, C++ backend

- Python frontend
  - No need for separate configuration/steering language
  - Easy interfacing with other tools
- C++ backend
  - Does all heavy computations
  - Powerful and elegant bindings via pybind11 (no extra tools)
  - Allows static modifications of performance critical parts via template classes

```
from corsika import Particle, AtmospherePropagator
from corsika.writer import DATWriter
from corsika.units import km, eV, GeV

proton = Particle("proton",
                  position=(0, 0, 100 * km),
                  momentum=(0, 0, -1e18 * eV))

# set various options via keyword arguments and/or fields
atmosphere_propagator = AtmospherePropagator(atm="US standard")
atmosphere_propagator.he_model = "EPOS-LHC"
atmosphere_propagator.le_model = "FLUKA"
atmosphere_propagator.em_energy_cut_off = 0.1 * GeV
atmosphere_propagator.muon_energy_cut_off = 1 * GeV

# generate 100 vertical proton showers and write to a file
with DATWriter("DAT000001") as writer:
    for run in range(100):
        ground_particles, longitudinal_profiles = atmosphere_propagator(proton)
        # optionally do something with results
        writer(ground_particles, longitudinal_profiles)
```

Stack Manipulation

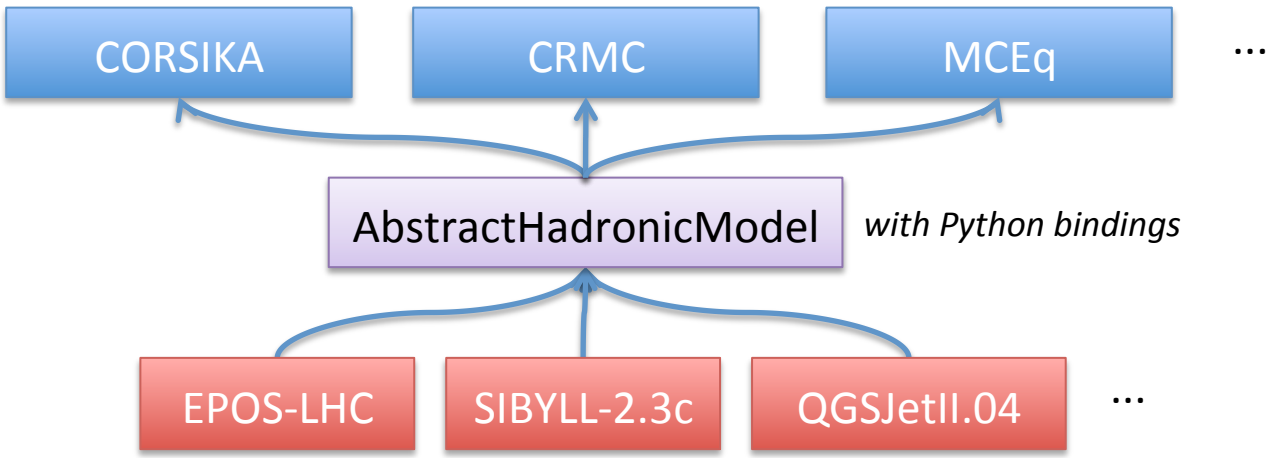
Particle Stack




```
#include <Particle>

template <typename ParticleT,
          typename ParticleStackT,
          typename AbstractHadronicModelT>
class AtmospherePropagator {
public:
  void main_loop() {
    while (!stack_.empty()) {
      const auto particle_in = stack_.pop();
      const auto particles_out =
propgate(particle_in);
        for (const auto& p : particles_out)
          stack_.push(p);
    }
  }

  std::vector<ParticleT> propagate(ParticleT p) {
      // uses model_ ...
  }

protected:
    ParticleStackT stack_;
    AbstractHadronicModelT model_;
};

template <typename ParticleT>
class DefaultParticleStack
{
public:
    DefaultParticleStack();
    ~DefaultParticleStack();
    ParticleT pop(); // may throw SimulationAbort
    void push(ParticleT p);
    bool empty() const;
};
```

- Generated particles are put on particle stack
- Next particle to propagate is taken from the stack
- Allow users to modify what “next particle from particle stack” is
- Allow stack to abort simulation
- IceCube’s use-case: Only simulate showers with high-energy muons
  - First propagate all hadrons down to some threshold energy
  - Abort shower early if no hadrons decay into high-energy muons

Public Hadronic Model Interface



- CORSIKA needs abstraction layer over hadronic interaction models
  - (In)Elastic cross-section
  - Generate (in)elastic event after collision
- Allow other tools to access this common interface
- Goal: Unify tools used in HE and CR physics
  - Make sure events generated for LHC and Auger are same
  - Anatoli @ ISVHECRI 2018: “Need to check out tooling...”

Modify hadronic interactions

```
#include <Particle>

template <typename ParticleT,
          typename ParticleStackT,
          typename AbstractHadronicModelT>
class AtmospherePropagator {
public:
  void main_loop() {
    while (!stack_.empty()) {
      const auto particle_in = stack_.pop();
      const auto particles_out =
propgate(particle_in);
        for (const auto& p : particles_out)
          stack_.push(p);
    }
  }

  std::vector<ParticleT> propagate(ParticleT p) {
      // uses model_ ...
  }

protected:
    ParticleStackT stack_;
    AbstractHadronicModelT model_;
};
```

- Allow users to modify model output in CORSIKA
  - Study impact of modified hadronic interactions on air shower observables (multiplicity, cross-section, ...)
- Allow users to replace AbstractHadronicModelT in C++ backend

Users may inherit from AbstractHadronicModel
to implement arbitrary modifications of model response

Multiple Weights per Particle

```
#include <Particle>

template <typename ParticleT,
          typename ParticleStackT,
          typename AbstractHadronicModelT>
class AtmospherePropagator {
public:
  void main_loop() {
    while (!stack_.empty()) {
      const auto particle_in = stack_.pop();
      const auto particles_out =
propgate(particle_in);
        for (const auto& p : particles_out)
          stack_.push(p);
    }
  }

  std::vector<ParticleT> propagate(ParticleT p) {
      // uses model_ ...
  }

protected:
    ParticleStackT stack_;
    AbstractHadronicModelT model_;
};
```

- Allow multiple weights per Particle
- Use cases:
  - Study effect of thinning
  - Study impact of multiple hadronic model modifications at once (e.g. pion multiplicity up and down)

```
template <unsigned NWeight>
class Particle {
public:
    // Particle Interface
    float& weight(unsigned i) {
        return weight_[i];
    }
protected:
    float weight_[NWeight];
};
```

CORSIKA on Github

- Use public git repository like Github, Gitlab, ...
  - Allow users to fork and send Pull Requests
    - Increase participation in development
  - Issue tracker
    - Keep track of bugs and feature requests
  - Continuous integration
    - Build checks and unit tests for commits and all pull requests
  - Open documentation
  - Visibility
  - Requires suitable Open Source License (e.g. BSD 3-clause)