Cascade Equations ~~and accelerated methods~~ and recent developments in MCEq

Anatoli Fedynitch
ICRR, University of Tokyo, Japan

CORSIKA 8 virtual workshop
June 25th, 2020

ICRR
Institute for Cosmic Ray Research
University of Tokyo

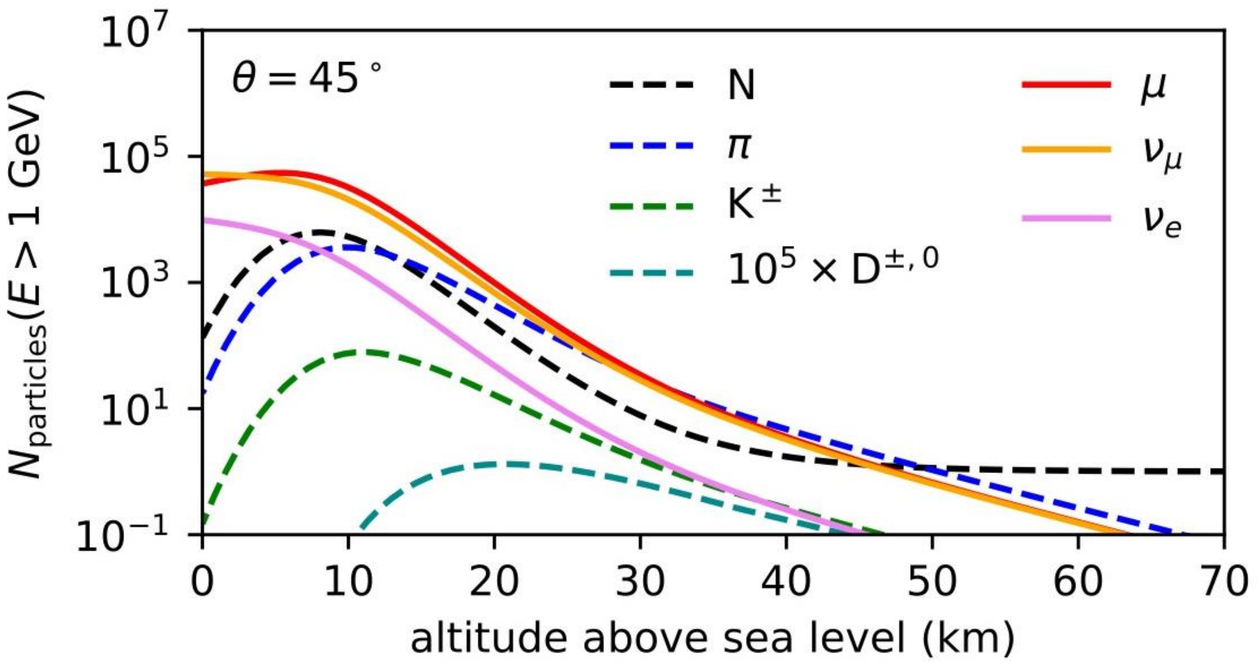
What is MCEq?

1. Open source numerical iterative cascade equation solver

2. Cascade equations = транспорт equations solved by CORSIKA with Monte Carlo method

3. Mainly used in atmospheric lepton and neutrino telescope community

4. Potentially interesting for
   - Cascade eqn. solver in CORSIKA8
   - Air shower & cosmic ray “theory”
   - Beyond standard model/Pheno
   - Astrophysics



Transport equations (hadronic cascade equations) in 1D

System of coupled non-linear PDE for each particle species *h* :

$$\frac{\mathrm{d}\Phi_h(E,X)}{\mathrm{d}X} = -\frac{\Phi_h(E,X)}{\lambda_{\mathrm{int},h}(E)} - \frac{\Phi_h(E,X)}{\lambda_{\mathrm{dec},h}(E,X)} - \frac{\partial}{\partial E}\left(\mu(E)\Phi_h(E,X)\right) + \sum_k \int_E^\infty \mathrm{d}E_k \frac{\mathrm{d}N_{k(E_k)\to h(E)}}{\mathrm{d}E} \frac{\Phi_k(E_k,X)}{\lambda_{\mathrm{int},k}(E_k)} + \sum_k \int_E^\infty \mathrm{d}E_k \frac{\mathrm{d}N^{\mathrm{dec}}_{k(E_k)\to h(E)}}{\mathrm{d}E} \frac{\Phi_k(E_k,X)}{\lambda_{\mathrm{dec},k}(E_k,X}$$

cosmic ray physics

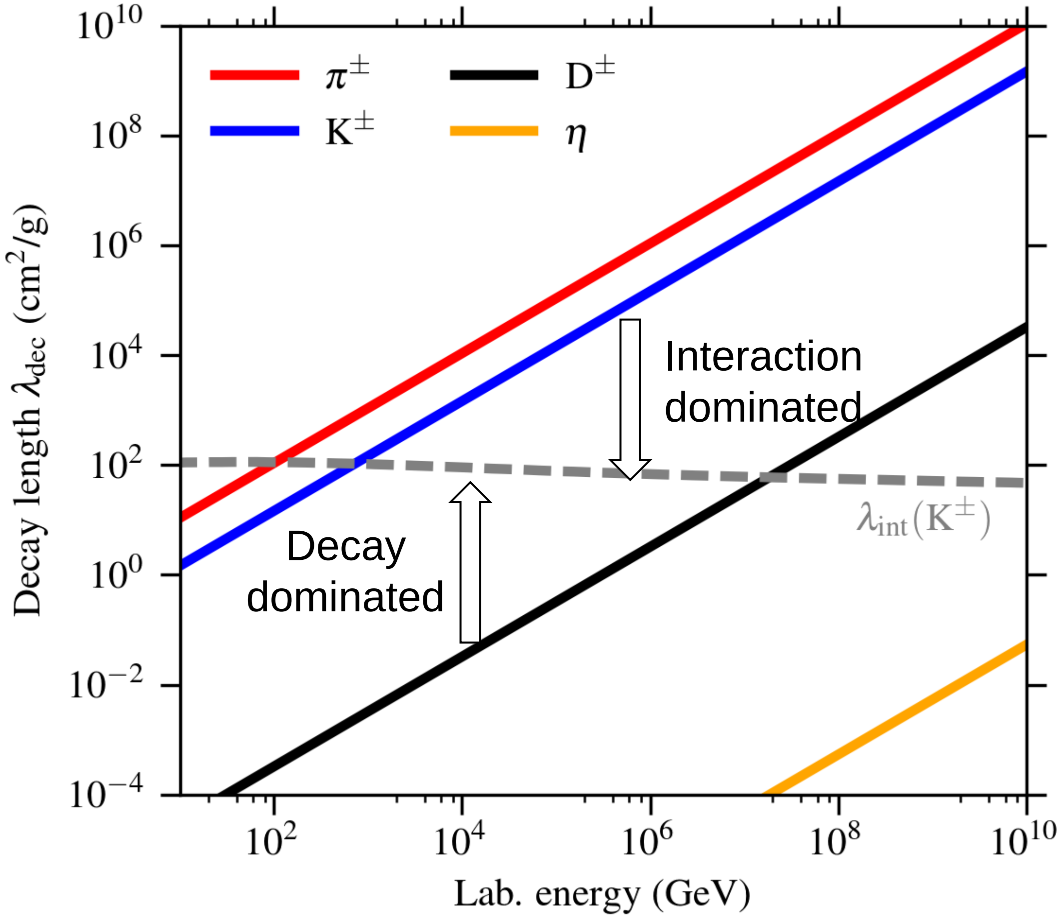
Interactions with air

Decays

atmospheric physics

Continuous losses

particle physics



$$X(h_0) = \int_0^{h_0} \mathrm{d}\ell\ \rho_{\mathrm{air}}(\ell)$$

MCEq: Matrix Cascade Equations

A. Fedynitch, R. Engel, T. K. Gaisser, F. Riehn and S. Todor
PoS ICRC 2015, 1129 (2015), EPJ Web Conf. 99, 08001 (2015)
and EPJ Web Conf. 116, 11010 (2016)

$$\frac{\mathrm{d}\Phi_h(E,X)}{\mathrm{d}X} = -\frac{\Phi_h(E,X)}{\lambda_{\mathrm{int},h}(E)} - \frac{\Phi_h(E,X)}{\lambda_{\mathrm{dec},h}(E,X)} - \frac{\partial}{\partial E}(\mu(E)\Phi_h(E,X)) + \sum_\ell \int_E^\infty \mathrm{d}E_\ell \frac{\mathrm{d}N_{\ell(E_\ell)\to h(E)}}{\mathrm{d}E}\frac{\Phi_\ell(E_\ell,X)}{\lambda_{\mathrm{int},l}(E_\ell)} + \sum_\ell \int_E^\infty \mathrm{d}E_\ell \frac{\mathrm{d}N^{\mathrm{dec}}_{\ell(E_\ell)\to h(E)}}{\mathrm{d}E}\frac{\Phi_\ell(E_\ell,X)}{\lambda_{\mathrm{dec},l}(E_\ell,X)}$$

$$\frac{\mathrm{d}\Phi^h_{E_i}}{\mathrm{d}X} = -\frac{\Phi^h_{E_i}}{\lambda^h_{\mathrm{int},E_i}} - \frac{\Phi^h_{E_i}}{\lambda^h_{\mathrm{dec},E_i}(X)} - \vec{\nabla}_i(\mu^h_{E_i}\Phi^h_{E_i}) + \sum_{E_k\geq E_i}^{E_N}\sum_\ell \frac{c_{\ell(E_k)\to h(E_i)}}{\lambda^\ell_{\mathrm{int},E_k}}\Phi^\ell_{E_k} + \sum_{E_k\geq E_i}^{E_N}\sum_\ell \frac{\mathrm{d}_{\ell(E_k)\to h(E_i)}}{\lambda^\ell_{\mathrm{dec},E_k}(X)}\Phi^\ell_{E_k}$$

State (or flux) vector

$$\vec{\Phi} = \left(\vec{\Phi}^{\mathrm{p}} \quad \vec{\Phi}^{\mathrm{n}} \quad \vec{\Phi}^{\pi^+} \quad \ldots \quad \vec{\Phi}^{\bar{\nu}_\mu} \quad \ldots\right)^T$$

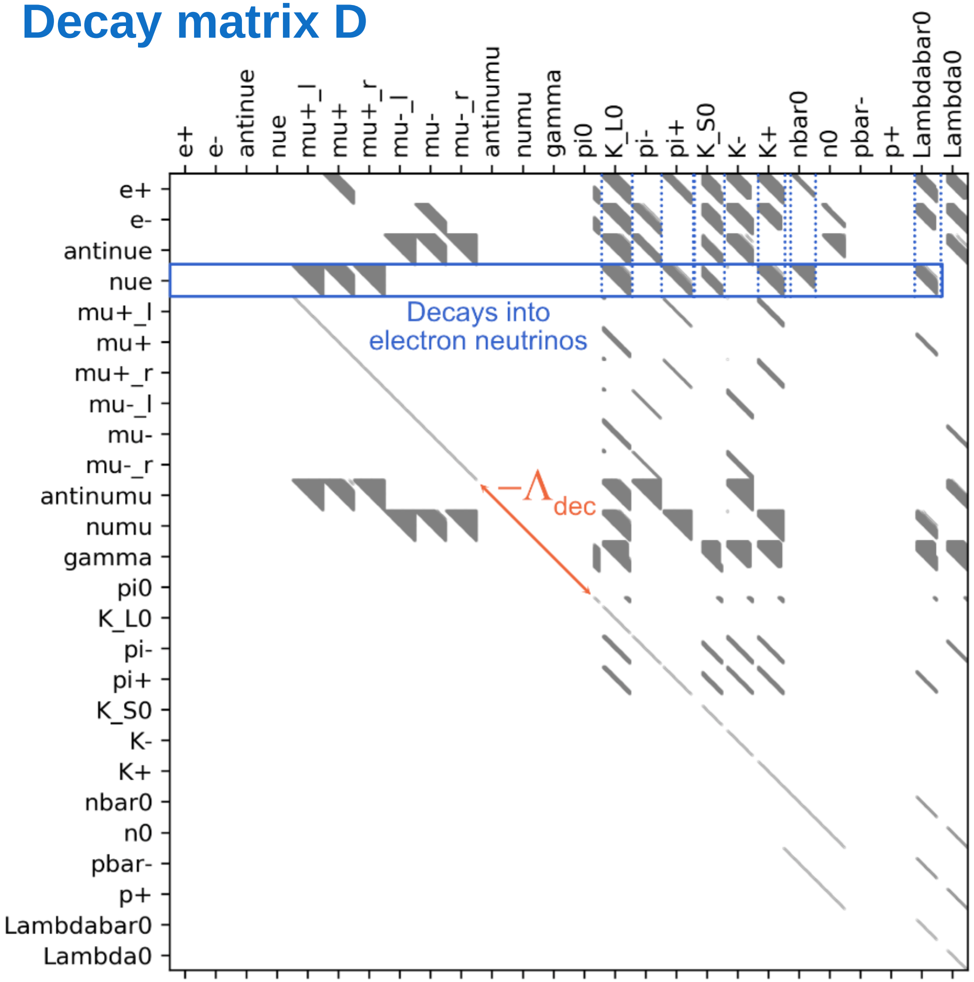
$$\vec{\Phi}^{\mathrm{p}} = \left(\Phi^{\mathrm{p}}_{E_0} \quad \Phi^{\mathrm{p}}_{E_1} \quad \cdots \quad \Phi^{\mathrm{p}}_{E_N}\right)^T$$

“Matrix form”

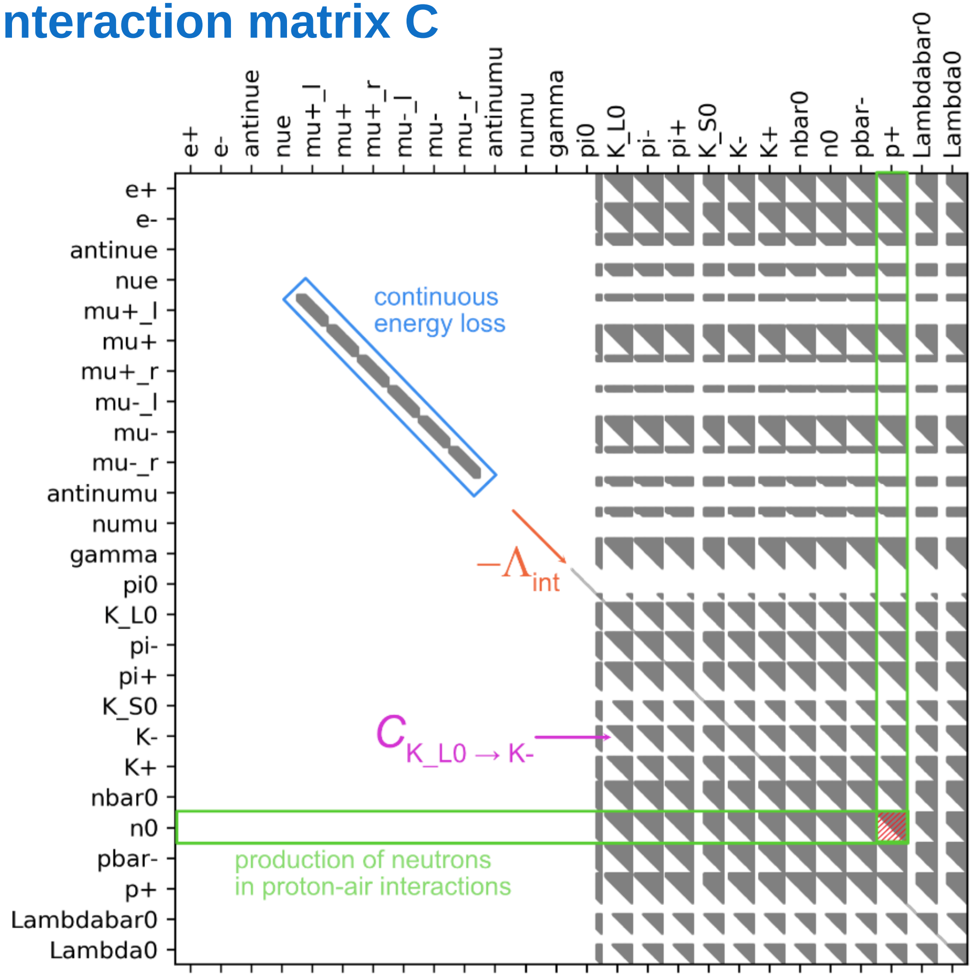
$$\frac{\mathrm{d}}{\mathrm{d}X}\vec{\Phi} = -\vec{\nabla}_E(\mathrm{diag}(\vec{\mu})\vec{\Phi}) + (-\mathbf{1}+\mathbf{C})\mathbf{\Lambda}_{\mathrm{int}}\vec{\Phi} + \frac{1}{\rho(X)}(-\mathbf{1}+\mathbf{D})\mathbf{\Lambda}_{\mathrm{dec}}\vec{\Phi}$$

Sparse matrix structure

Decay matrix D

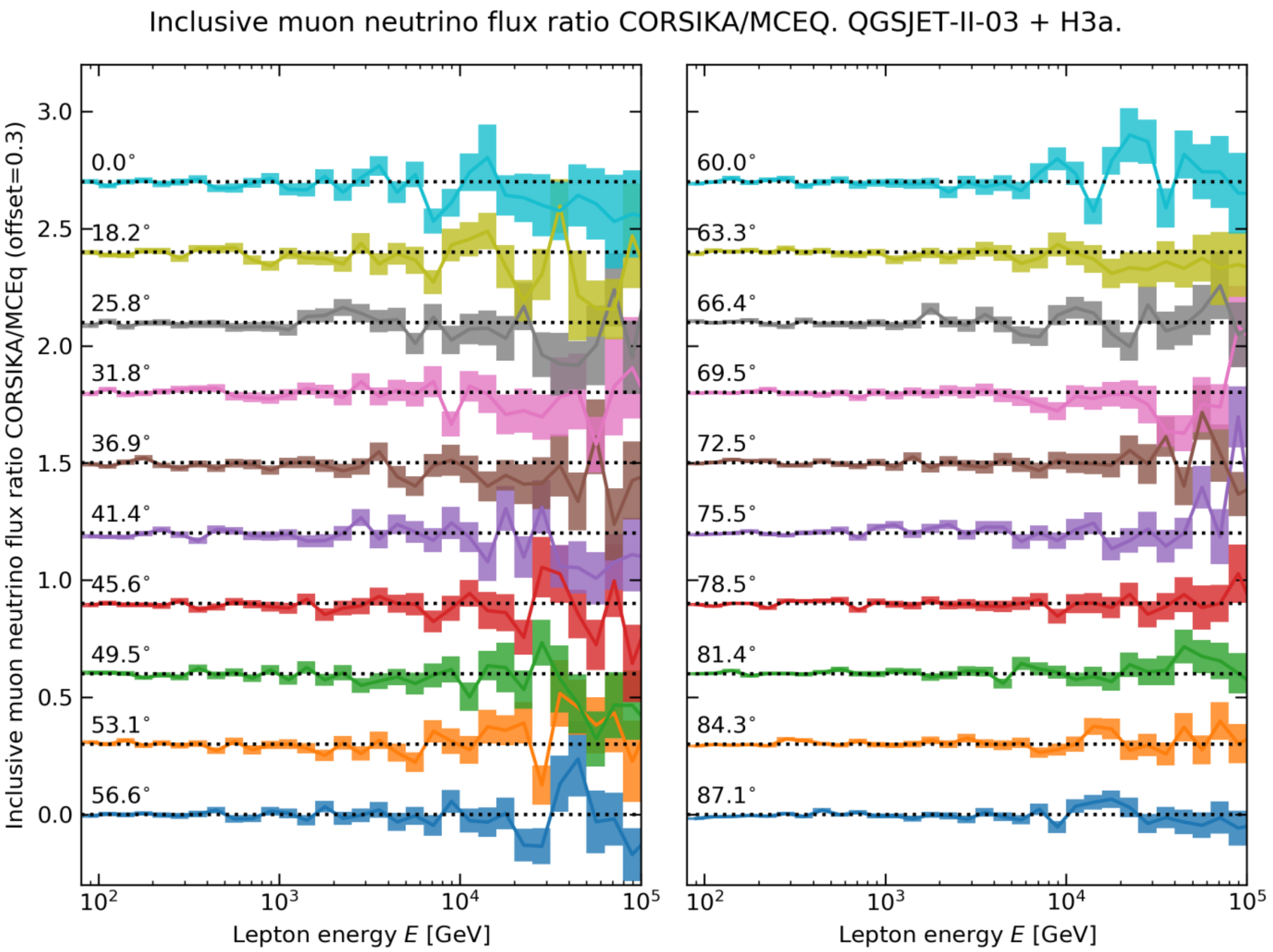


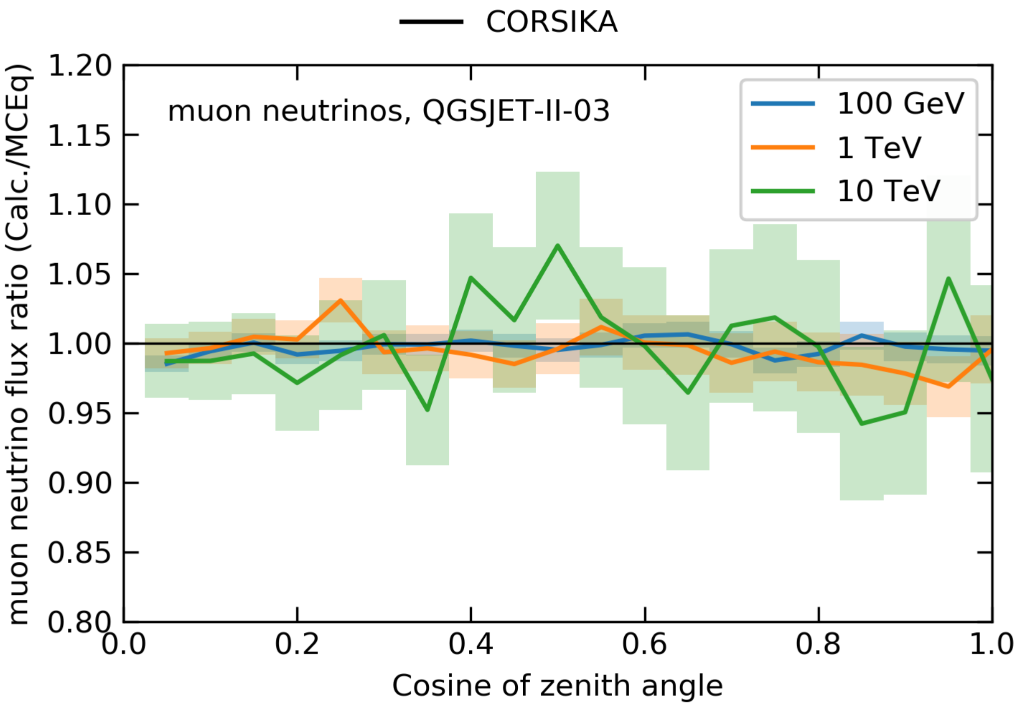
Interaction matrix C



matrices are sparse → high performance

MCEq vs (thinned) CORSIKA calculation in 1D



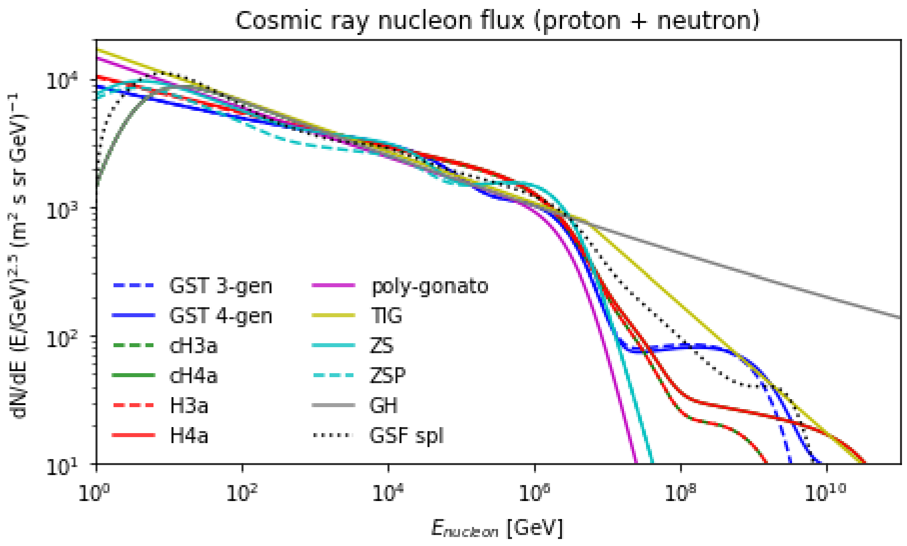


- BSD licensed @ https://github.com/afedynitch/MCEq

CORSIKA: A. Fedynitch, J. Becker Tjus and P. Desiati, PRD 2012
MCEq: Code paper, AF, R. Engel, in prep.

Available models

Cosmic ray flux models are in the independent crflux module.



Hadronic interaction models are:

- SIBYLL-2.3c (=d: will be fixed in next release)
- SIBYLL-2.3c01
- SIBYLL-2.3
- SIBYLL-2.1
- EPOS-LHC
- QGSJet-II-04
- QGSJet-II-03
- QGSJet-01c
- DPMJET-III-3.0.6
- DPMJET-III-19.1
- SIBYLL-2.3c_pp (for proton-proton collisions)

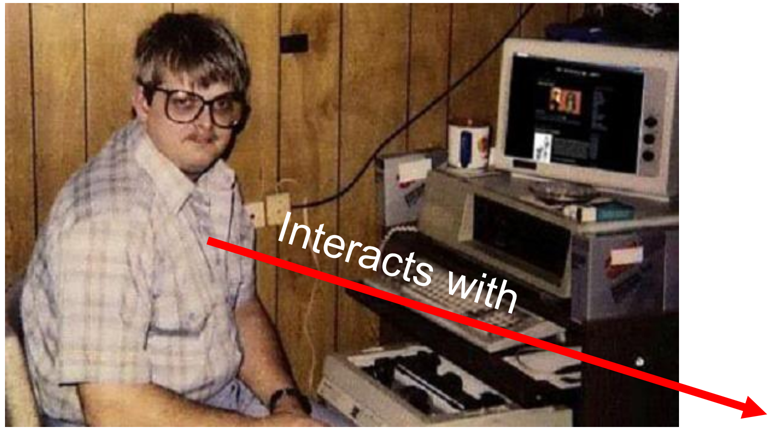
Atmosphere models from

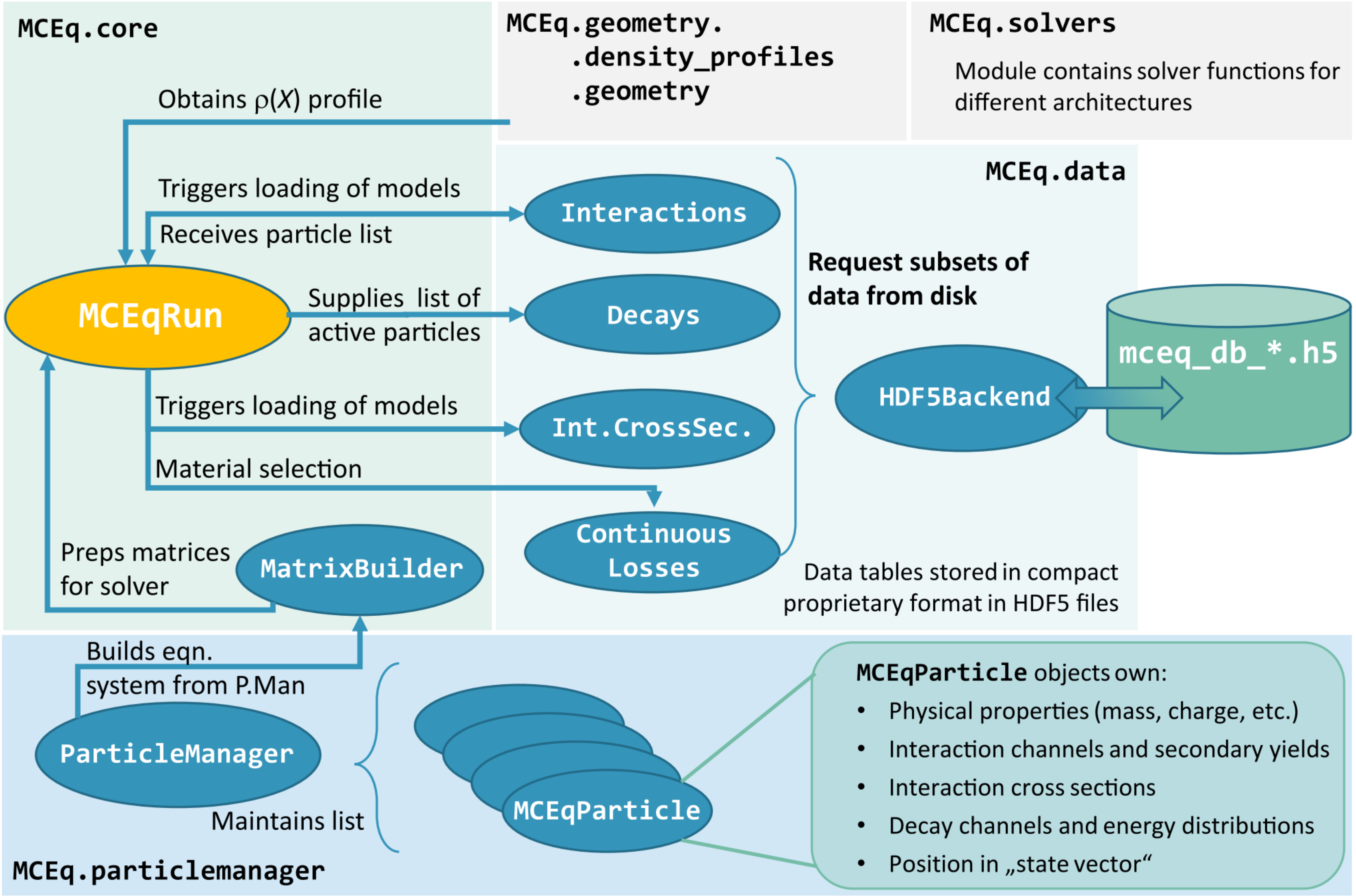
- CORSIKA7 (multiple locations)
- NRLMSISE-00 (global, “static”)
- Some special cases and interface to tabulated atm.

Live demo

(you can run it yourself using the CKA8_WS_demo.ipynb on indico)

Code architecture





Key points:

- ☺ mostly python
- ☺ heavy on numpy
- ☺ some scipy
- ☺ some ctypes modules
- ☺ likes Intel MKL and/or Nvidia cuSparse (via cupy)
- ☺ File(s) are HDF5
- ☺ Builds on Azure as binary wheel
- ☺ Deployed to PyPi
- ☺ Supports Linux, Windows and Mac, Python 2.7 - 3.8

Runtime: CUDA vs MKL in MCEq

CUDA (cuSparse)

```
Timer unit: 0.001 s

Total time: 0.714455 s
File: c:\users\afedy\devel\git\mceq\MCEq\solvers.py
Function: solve_step at line 111

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   111                                           def solve_step(self, rho_inv, dX):
   112                                               """Makes one solver step on GPU using cuSparse (BLAS)"""
   113
   114     11530          8.6      0.0      1.2      self.cp.cusparse.csrmv(a=self.cu_int_m,
   115     11530          5.8      0.0      0.8                             x=self.cu_curr_phi,
   116     11530          5.6      0.0      0.8                             y=self.cu_delta_phi,
   117     11530          5.2      0.0      0.7                             alpha=1.,
   118     11530        300.7      0.0     42.1                             beta=0.)
   119     11530          8.8      0.0      1.2      self.cp.cusparse.csrmv(a=self.cu_dec_m,
   120     11530          5.5      0.0      0.8                             x=self.cu_curr_phi,
   121     11530          5.4      0.0      0.8                             y=self.cu_delta_phi,
   122     11530          5.1      0.0      0.7                             alpha=rho_inv,
   123     11530        296.4      0.0     41.5                             beta=1.)
   124     11530         11.4      0.0      1.6      self.cubl.saxpy(self.cubl_handle, self.cu_delta_phi.shape[0], dX,
   125     11530          7.0      0.0      1.0                      self.cu_delta_phi.data.ptr, 1,
   126     11530         48.9      0.0      6.8                      self.cu_curr_phi.data.ptr, 1)
```

Intel MKL

```
   239     11540         13.1      0.0      0.1  for step in range(nsteps):
   240                                               # delta_phi = int_m.dot(phi)
   241     11530         14.6      0.0      0.2      gemv(trans, m, m, cdone,
   242     11530         11.2      0.0      0.1          matdsc, int_m_data, int_m_ci, int_m_pb, int_m_pe, phi,
   243     11530       7198.0      0.6     77.3          cdzero, delta_phi)
   244                                               # delta_phi = rho_inv * dec_m.dot(phi) + delta_phi
   245     11530         41.3      0.0      0.4      gemv(trans, m, m, byref(fl_pr(rho_inv[step])),
   246     11530         12.1      0.0      0.1          matdsc, dec_m_data, dec_m_ci, dec_m_pb, dec_m_pe, phi,
   247     11530       1869.8      0.2     20.1          cdone, delta_phi)
   248                                               # phi = delta_phi * dX + phi
   249     11530        132.9      0.0      1.4      axpy(m, fl_pr(dX[step]), delta_phi, cione, phi, cione)
```

- Most of the time spent in HPC libs
- No python overhead = no performance gain from C++
- For CKA8 it is straight forward to write a small core in C++ using local density & geometry
- Matrices can be generated with MCEq as is or stored/cached in HDF5 or so
- Requires a sparse matrix BLAS lib. Any recommendation?
- Several additional ways to boost performance:
  - Multiple "RHS"
  - Segmented atmosphere
  - Newton iterations
  - At the cost of higher mem. consumption or longer initialization
  - No meaning to do it now

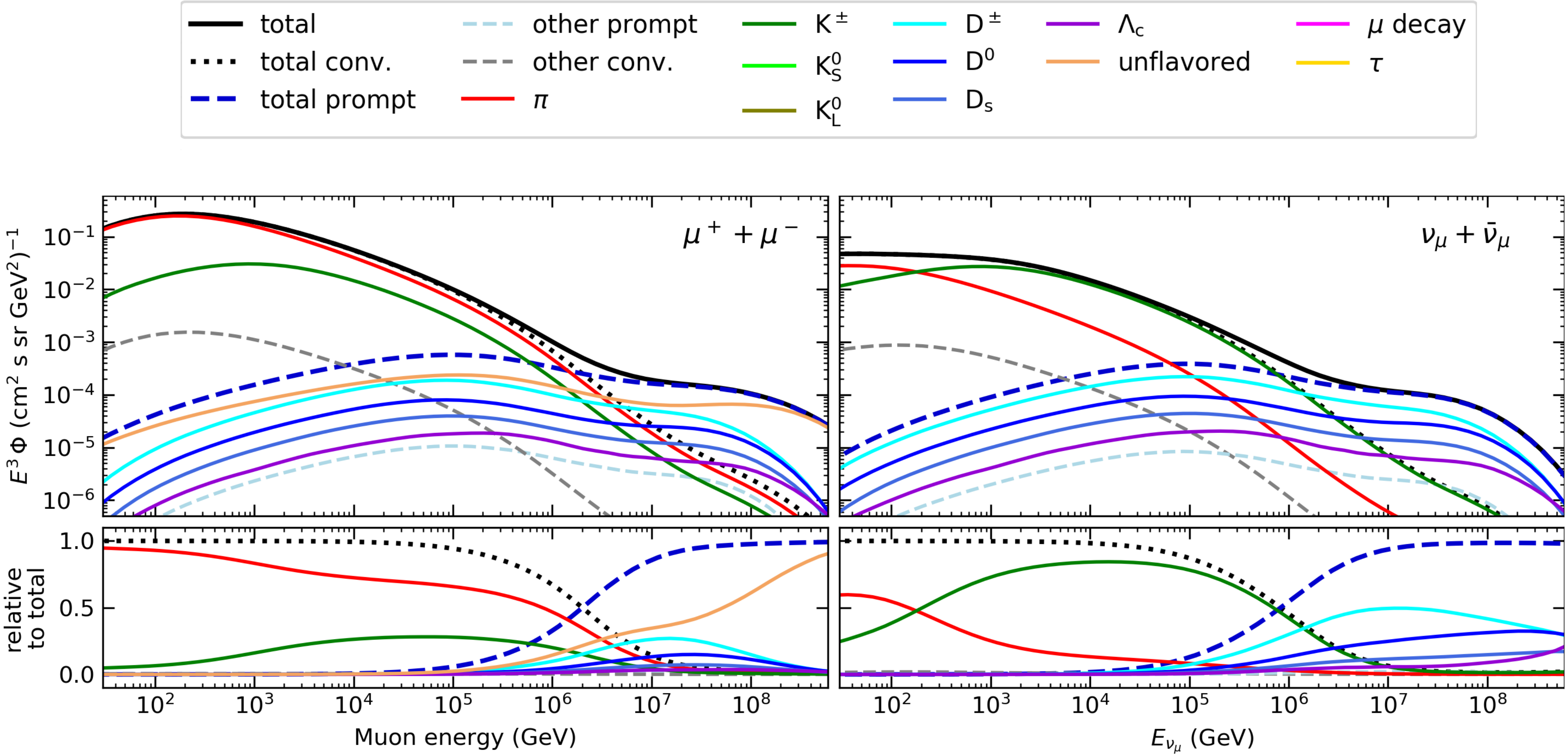
Summary/conclusion/discussion

- CORSIKA and MCEq are based on the same equations. MCEq gives the average “air shower”
- Code is mature and in development since 2013, and recently cross checked with CKA8 (see Max’ talk from Monday)
- Can be further accelerated when needed, many possibilities
- Hybrid shower generation MC + CE + “de-thinning” not well documented and sometimes obscure
- Proposal: implementation in CKA8 as a small solver module that uses matrices generated by official MCEq code, either in runtime or cashed
- Potential developments:
  - Solve for the moments (get fluctuations)
  - 3D cascade eqn. solvers/meshed geometries
- Do hybrid techniques need revising, which communities are interested in this?

MCEq news

1. Since last year: MCEq distributed via PyPi and fluxes published:
   a) The recommended installation method is **“pip install MCEq“**
   b) The differences in fluxes between models are discussed Section VI of the AF et al. PRD100 paper (sorry, arxiv version no yet up-to-date)
2. Version 1.1.x is stable and physics-wise identical to 1.0.X versions
3. Typically version bumps mean
   a) X.X.**O**: bugfix, or small features without changing interfaces or physics
   b) X.**O**.X: significant feature update, that may involve change in the interface or physics
   c) Current stable version is 1.2.1
4. Version 1.2.1 has some physics changes because a bug found in decay matrices and because of some adjustments in sync with CKA8
   a) Fluxes change by ~+10% change below 10-30 GeV
   b) Air-showers/bundles: particle yields (single_primary_particle) increase by several percent at ~GeV energies
   c) For fluxes > 40 GeV no visible change.
5. Follow the CHANGELOG on github to stay informed what’s new in each version
6. When using MCEq for physics, use official releases (from PyPi or git tags), then you can go easily back even with pip.
7. For questions about MCEq use the issues on github. Errors and discussions will be indexed by google and easier to find.

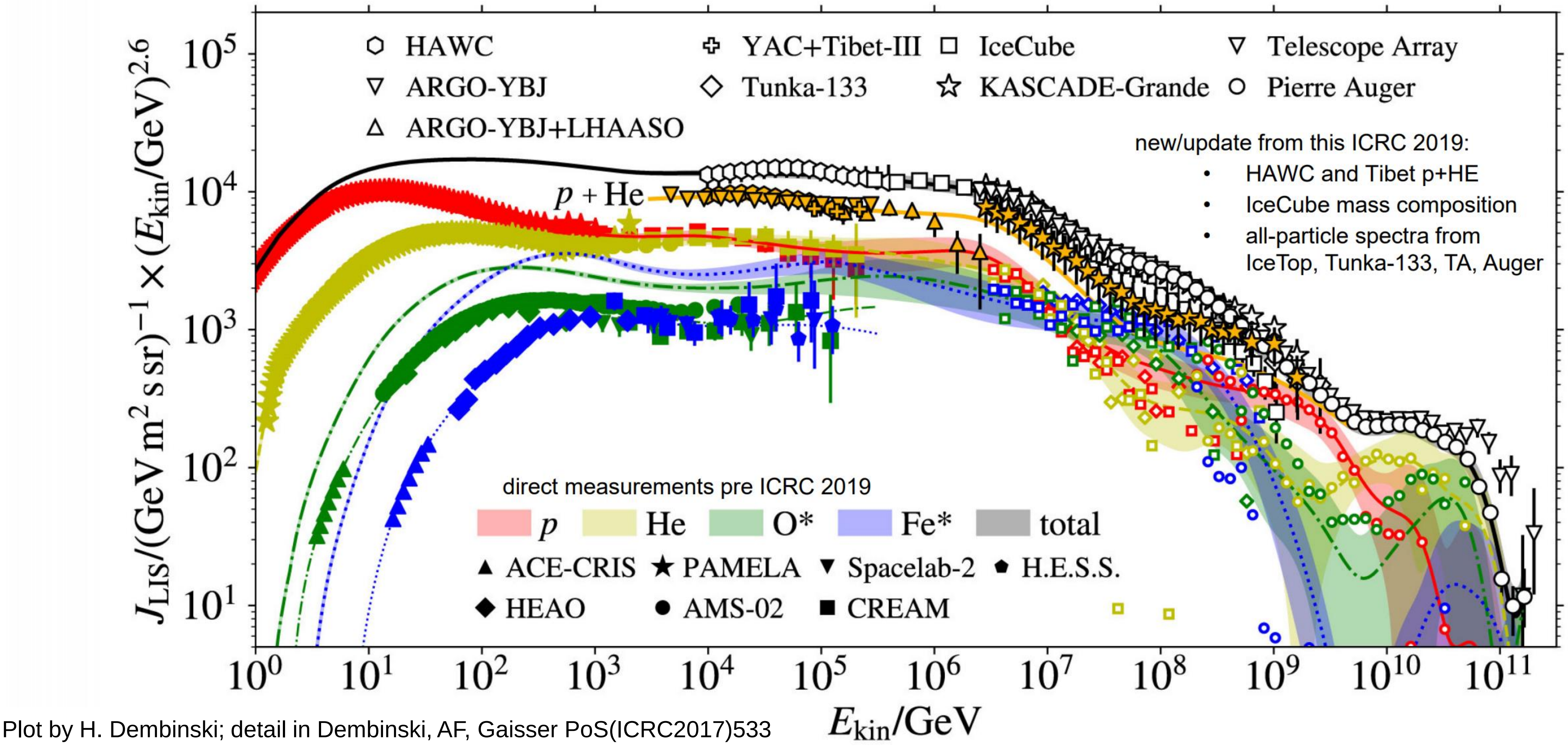
Hadrons contributing to muonic leptons



AF, F. Riehn, R. Engel, T.K. Gaisser, T. Stanev, PRD 100 2019

CR flux: GSF - Global Spline Fit (one problem "almost solved")

F. Schröder, ICRC 2019 rapporteur talk



Plot by H. Dembinski; detail in Dembinski, AF, Gaisser PoS(ICRC2017)533

Very good precision achieved!
Error band smaller than markers/curves

Remaining composition and energy scale uncertainties between experiments