chromo

An event generator frontend for particle and astroparticle physics

Hans Dembinski[1], Anatoli Fedynitch[2], Anton Prosekin[2]
[1]TU Dortmund, [2]Academia Sinica, Taipei, Taiwan

Workshop on the tuning of hadronic interaction models, University Wuppertal,
Jan 22 – 25, 2024

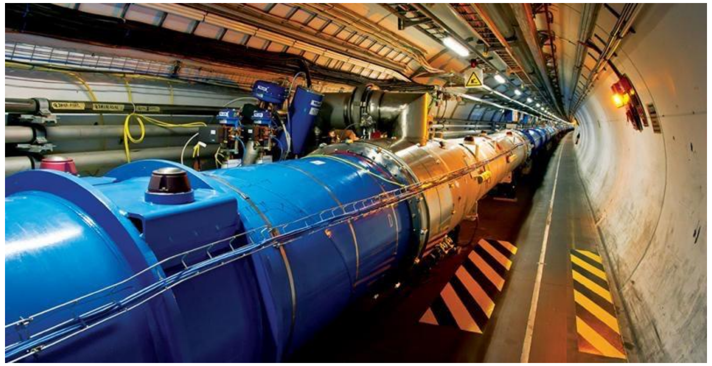
Quick facts chromo

Cosmic ray and HadROnic interactiOn MOnte-carlo frontend

- Python frontend to generators written in Fortran & C++
  - DPMJet-III*, PhoJet*, EPOS-LHC, Pythia-6.4, Pythia-8.3, QGSJet*, QGSJet- II*, SIBYLL*, SOPHIA, UrQMD 3.4 (* = several versions)
  - Use as Python library or command-line interface
- Open source development on Github
  - https://github.com/impy-project/chromo
  - BSD 3-clause license, contributions welcome
- Main authors
  - Anatoli Fedynitch (project lead), Hans Dembinski, Anton Prosekin
- Available on PyPI
  - Authors already use it for science projects
  - pip install chromo to install
  - For installation from source, see README.md

Introduction

- Applications in (astro)particle physics require simulations of particle production in interactions of photons, hadrons, and nuclei
  - Cosmic ray propagation through galaxy
  - Air showers
  - Min-bias physics and underlying event at colliders
- No standard event generator (yet)
  - Common: compute result with input from several generators to estimate systematic uncertainty
- Event generators have no standard interface
  - Varying event representations, particle IDs, and data structures
- Most generators implemented in Fortran 77
  - modern generators in C++
- Majority of scientific computing, education, and data science have moved to Python ecosystem
- Chromo (formerly named impy) provides
  - Standard Python interface over generators
  - Taps into rich Python ecosystem for extra features
  - CLI to generate HepMC & ROOT output or SVG images



Supported event generators

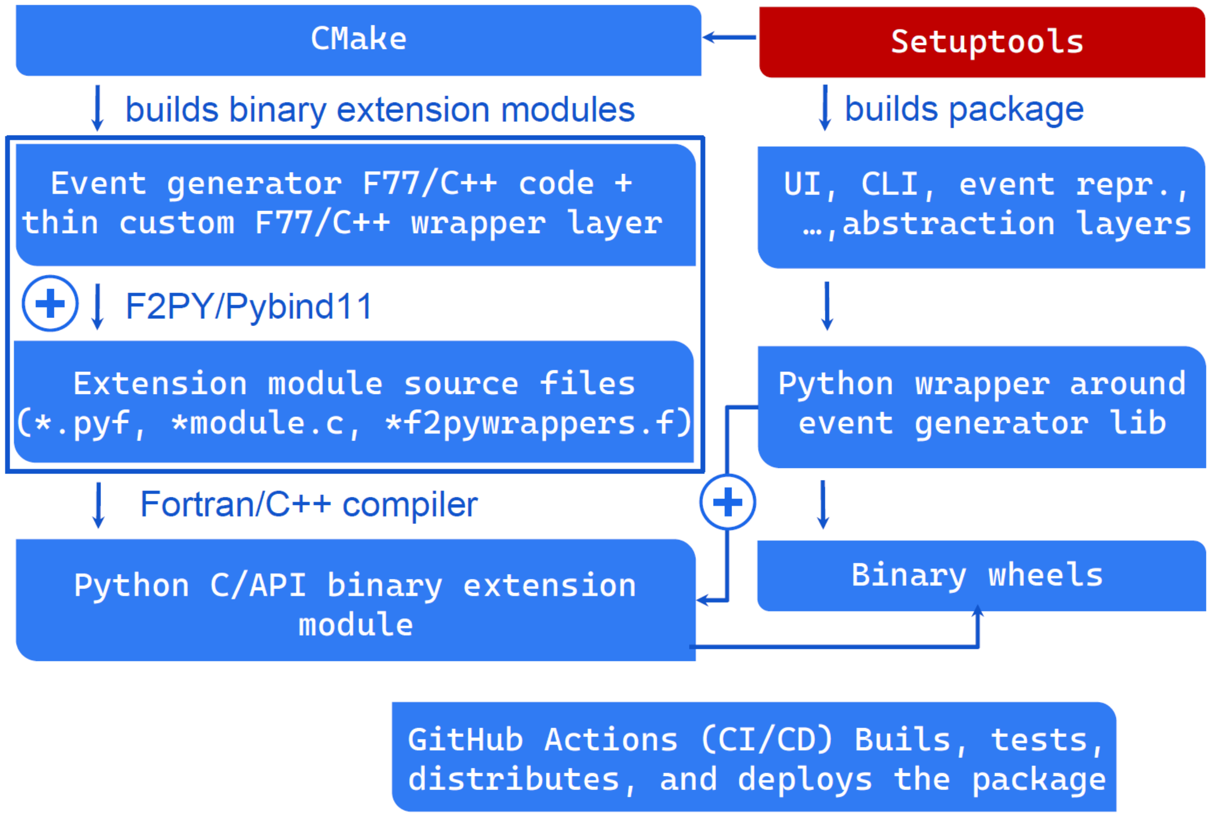
DPMJET Models :	PYTHIA Models :	QGSJet Models :	SIBYLL Models :	Other Models:
• DPMJET-III 3.0.6 • PHOJET 1.12-35 • DPMJET-III 19.1 • PHOJET 19.1 • DPMJET-III 19.3 • PHOJET 19.3	• PYTHIA 6.4 • PYTHIA 8.3	• QGSJet-01 • QGSJet-II-03 • QGSJet-II-04	• SIBYLL-2.1 • SIBYLL-2.3 • SIBYLL-2.3c • SIBYLL-2.3d • SIBYLL*	• EPOS-LHC • SOPHIA 2.0 • UrQMD 3.4 • FLUKA (in progress)

Supported event generators

Interaction model	Supported proj/targ	Comment
DPMJET-III 3.0.7 & PHOJET 1.12-36	*hN, γγ, γN, hA, γA, AA*	
DPMJET-III & PHOJET 19.1 and 19.3 (repo on GitHub)	*hN, γγ, γN, hA, γA, AA*	
EPOS-LHC	*hN, hA, AA*	
PYTHIA 6.4	*hN, ee, γγ, γN*	
PYTHIA 8.3 (https://pythia.org/)	*hN, ee, γγ, γN* & *hA, AA* (Argantyr)	unavailable on Windows
QGSJet-01	*hN, hA, AA*	
QGSJet-II-03	*hN, hA, AA*	
QGSJet-II-04	*hN, hA, AA*	
SIBYLL-2.1	*hN, hA (A<=20)*	
SIBYLL-2.3d	*hN, hA (A<=20)*	incl. legacy versions -2.3/-2.3c
SOPHIA 2.0	*γN*	
UrQMD 3.4 + second citation	hN, hA, AA*	unavailable on Windows

Technical concept

- Multiple layers:
  - original Fortran/C++ code of event generators
  - a custom Fortran/C++ integration layer
  - F2PY/Pybind instructions for building Python C/API extension modules,
  - Python code implementing the library
- Code follows an object-oriented approach with some functional-style code for internal auxiliary tasks.



CI/CD and Testing Workflow

GitHub Actions is used for CI/CD for automatic:

- Building with **cibuildwheels** package
- Testing with **pytest** package
- Deploying with **GitHub Actions**

Testing Workflow:

- Any code changes trigger pre-commit.ci code style validation and test workflows.
- Tests include compilation, building, and installation on Windows, Ubuntu, and macOS.
- Extensive testing with about 1100 unit tests managed by pytest framework.
- About 580 tests evaluate event generators across various permutations.
    - Monte Carlo methods are sensitive to small changes in floating-point calculations.
    - Probabilistic comparisons ensure correctness due to differences in mathematical libraries.

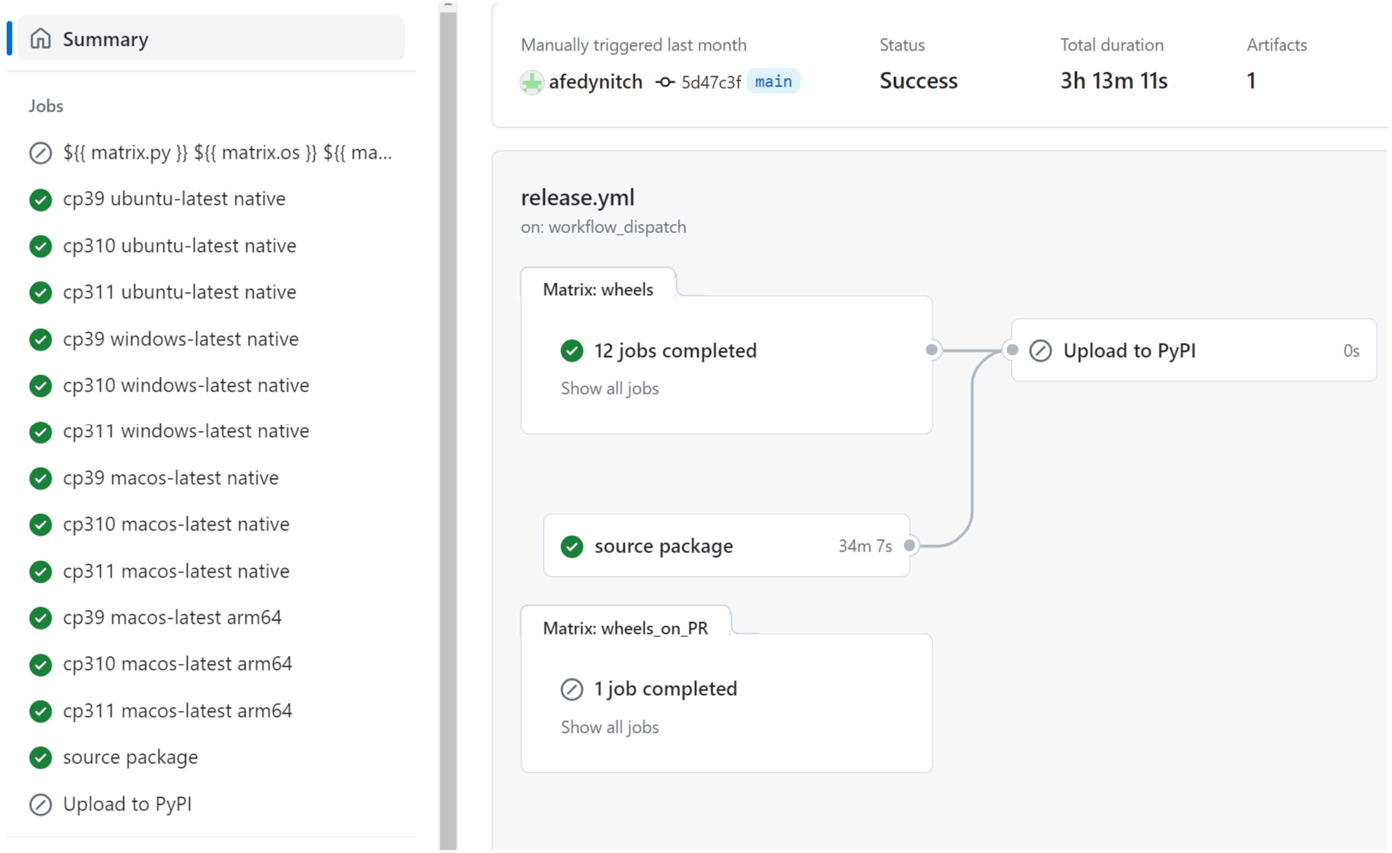
```
cibuildwheel on Linux
Building cp39-manylinux_x86_64 wheel
CPython 3.9 manylinux x86_64

▶ Setting up build environment...
                                                              ✓ 0.08s
▶ Building wheel...
                                                              ✓ 753.36s
▶ Repairing wheel...
                                                              ✓ 6.65s
▶ Testing wheel...
                                                              ✓ 1520.59s

✓ cp39-manylinux_x86_64 finished in 2280.75s
▶ Copying wheels back to host...
                                                              ✓ 0.13s

1 wheels produced in 39 minutes:
  chromo-0.4.1-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl

 ========== 1032 passed, 39 skipped, 32 xfailed in 1496.08s (0:24:56) ==
```

Automated Distribution

Build Process:
•Wheels consist of around 20 pre-compiled extension modules.
•Compilation and wheel construction are automated using **CMake** integrated with **setuptools**.
•A wheel is compiled for each platform and Python version.

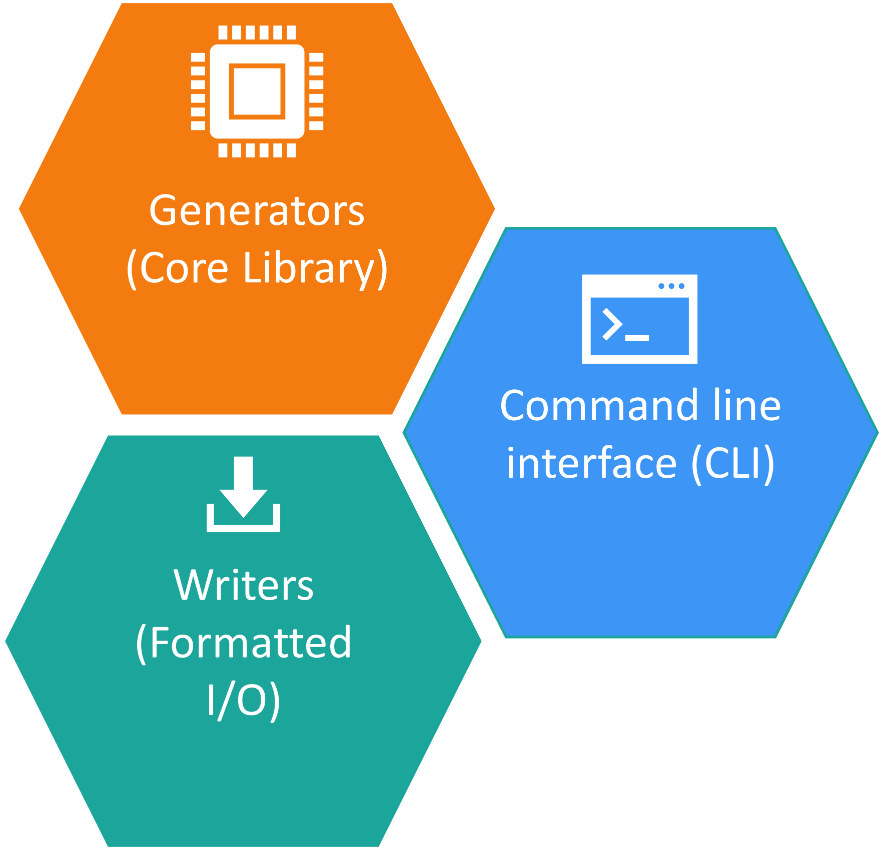
Release Workflow:
•Builds wheels for all platform and Python version combinations.
•Automated testing and upload to PyPI if all tests pass.
•**cibuildwheel** tool automates system-agnostic wheel creation.

Automated Distribution:
•Chromo is distributed as Python wheels through PyPI.
•Wheels are available for Windows 64-bit, Linux 64-bit, macOS Intel, and macOS Apple Silicon.
•Supported Python versions include 3.8, 3.9, 3.10, and 3.11.

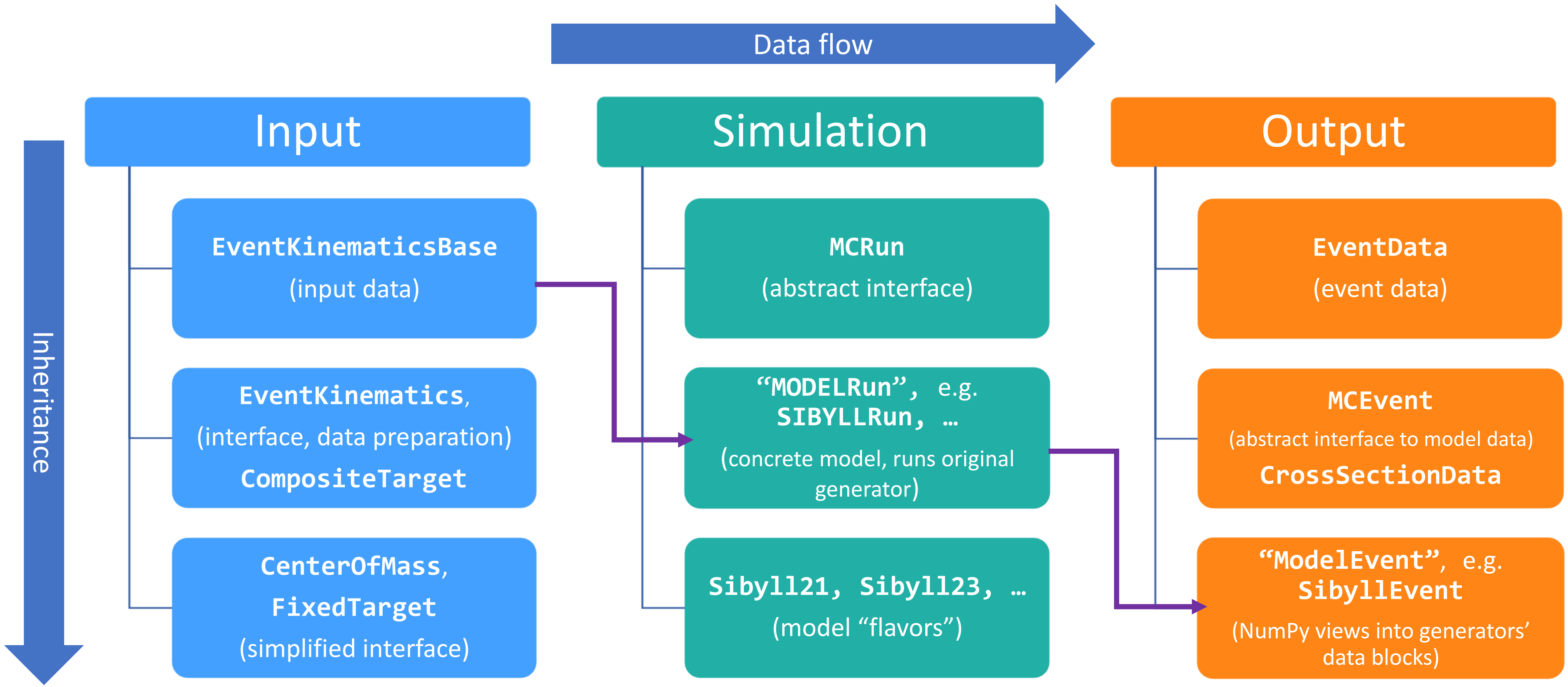


Components and integration

- Core library:
  - python scripts
  - jupiter notebooks
- Command line interface (CLI):
  - pipeline with other programs
  - drop-in substitution CRMC
- Writers (Formatted I/O representation of events)
  - SVG
  - Hepmc
  - Root

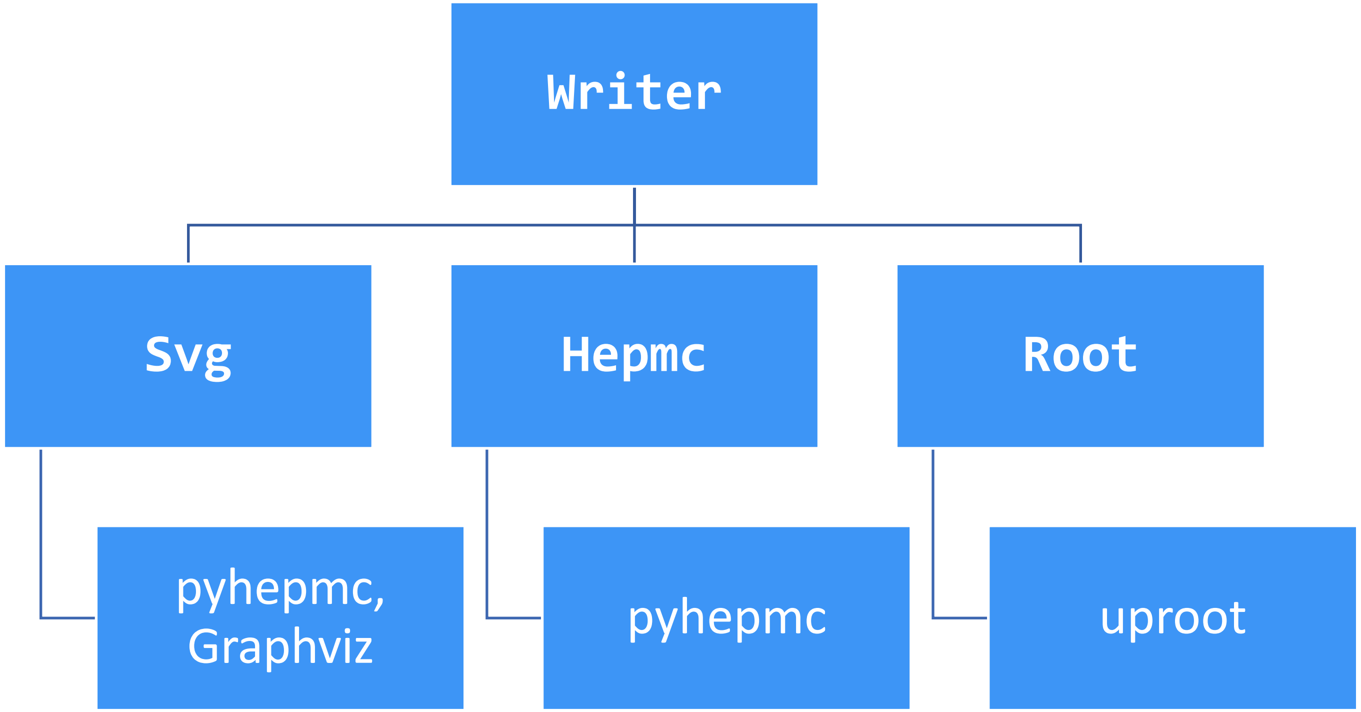


Core architecture



Formatted I/O and dependencies

- **`Writer`** is abstract class for wrapper classes over libraries that write to the corresponding formats



HepMC: Lingua franca for simulation software used at CERN

SIBYLL-2.1, pp, sqrt(s) = 20 GeV


```
HepMC::Version 3.02.05
HepMC::Asciiv3-START_EVENT_LISTING
T SIBYLL\|2.1\|
E 0 7 23
U GEV MM
P 1 0 2224 -8.9205041527748108e-02 1.3491769134998322e-01 2.0344371795654297e+00 2.3833706378936768e+00 1.2309999465942383e+00 2
P 2 0 111 8.9463070034980774e-02 -4.4863110780715942e-01 3.4179518222808838e+00 3.4519975185394287e+00 1.3496999442577362e-01 2
P 3 0 331 1.4990800619125366e-01 4.6003237366676331e-01 2.0358951091766357e+00 2.3069748878479004e+00 9.5749998092651367e-01 2
P 4 0 221 -1.6207817196846008e-01 -2.4423867464065552e-01 1.9545348882675171e+00 2.0511729717254639e+00 5.4879999160766602e-01 2
P 5 0 -211 -5.2947159856557846e-02 2.5346320867538452e-01 -5.1904711872339249e-02 2.9869863390922546e-01 1.3956999778747559e-01 1
P 6 0 111 -2.4904966354370117e-02 3.6575528979301453e-01 -1.8918764591217041e+00 1.9317893981933594e+00 1.3496999442577362e-01 2
P 7 0 2212 8.9764386415481567e-02 -5.2129906415939331e-01 -7.4990553855895996e+00 7.5760145187377930e+00 9.3826997280120850e-01 1
P 8 1 2212 -1.1027524620294571e-01 9.2852763831615448e-02 1.1708897352218628e+00 1.5073537826538086e+00 9.3826997280120850e-01 1
P 9 1 211 2.1069953218102455e-02 4.2065303772687912e-02 8.6355310678482056e-01 8.7602353096008301e-01 1.3956999778747559e-01 1
P 10 2 22 -1.5905208885669708e-02 -2.5956141948699951e-01 1.9417071342468262e+00 1.9595654010772705e+00 0.0000000000000000e+00 1
P 11 2 22 1.0539282113313675e-01 -1.8919278681278229e-01 1.4771823883056641e+00 1.4933792352676392e+00 0.0000000000000000e+00 1
P 12 3 211 1.3005101121962070e-02 9.5943860709667206e-02 2.2564361989498138e-01 2.8325849771499634e-01 1.3956999778747559e-01 1
P 13 3 -211 9.5603697001934052e-02 6.0167539864778519e-02 1.2169665843248367e-01 2.1799579262733459e-01 1.3956999778747559e-01 1
P 14 3 221 4.1300963610410690e-02 3.0392596125602722e-01 1.6885783672332764e+00 1.8057471513748169e+00 5.4879999160766602e-01 2
P 15 4 22 -1.3510279357433319e-01 4.4372059404850006e-02 1.5330873727798462e+00 1.5396685600280762e+00 0.0000000000000000e+00 1
P 16 4 22 -2.6978071779012680e-02 -2.8861477971076965e-01 4.2147985100746155e-01 5.1153844594955444e-01 0.0000000000000000e+00 1
P 17 6 22 -4.5049645006656647e-02 2.6162242889404297e-01 -1.5857362747192383e+00 1.6078042984008789e+00 0.0000000000000000e+00 1
P 18 6 22 2.0137846469879150e-02 1.0423322767019272e-01 -3.0665934085845947e-01 3.2451510429382324e-01 0.0000000000000000e+00 1
P 19 14 211 1.6239669173955917e-02 1.1180111020803452e-01 3.0710572004318237e-01 3.5670593380928040e-01 1.3956999778747559e-01 1
P 20 14 -211 -5.0198074430227280e-02 7.7754214406013489e-02 2.6761403679847717e-01 3.1666630506515503e-01 1.3956999778747559e-01 1
P 21 14 111 7.5260899960994720e-02 1.1438217759132385e-01 1.1139335632324219e+00 1.1324540376663208e+00 1.3496999442577362e-01 2
P 22 21 22 6.8888634443283081e-02 -3.4993162844330072e-03 5.4559254646301270e-01 5.5093902349472046e-01 0.0000000000000000e+00 1
P 23 21 22 6.3929148018360138e-03 1.1791287362575531e-01 5.6864660978317261e-01 5.8182567358016968e-01 0.0000000000000000e+00 1
HepMC::Asciiv3-END_EVENT_LISTING
```

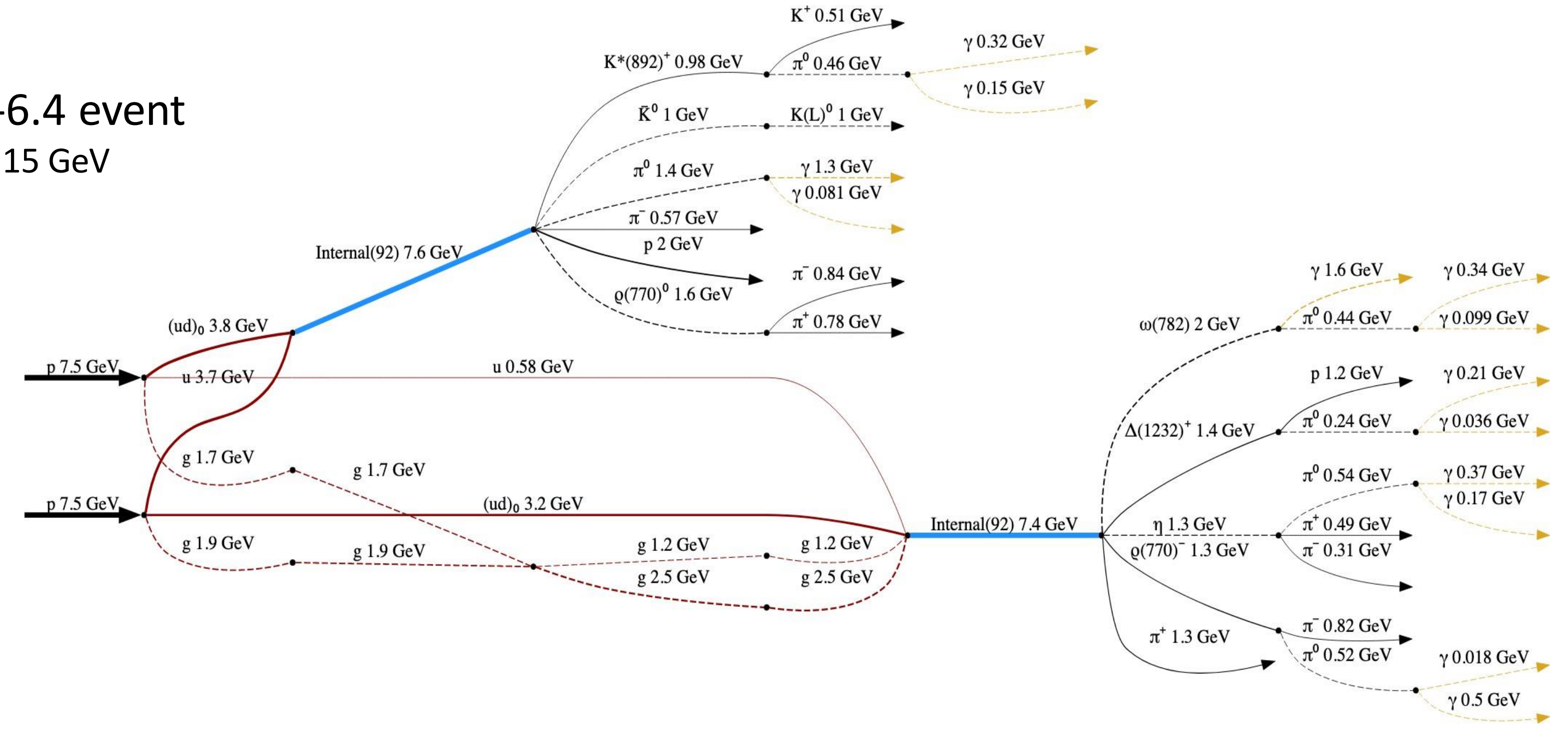

Example

```python
with Hepmc("file.hepmc", model) as writer:
    writer.write(event)
```

Event visualization

- If **graphviz** is installed, event (EventData object) will be visualized directly in the notebook via automatic conversion to HepMC3 event using **pyhemc** library
- Tooltip information about the particles and vertices is available by hovering the mouse over the lines and nodes
- History (mother and daughter particles) of some event generators (e.g. DpmJet) are repaired and rectified before output to be a valid HepMC event and be able to processed by Rivet

Pythia-6.4 event
sqrt(s) = 15 GeV



Command line interface

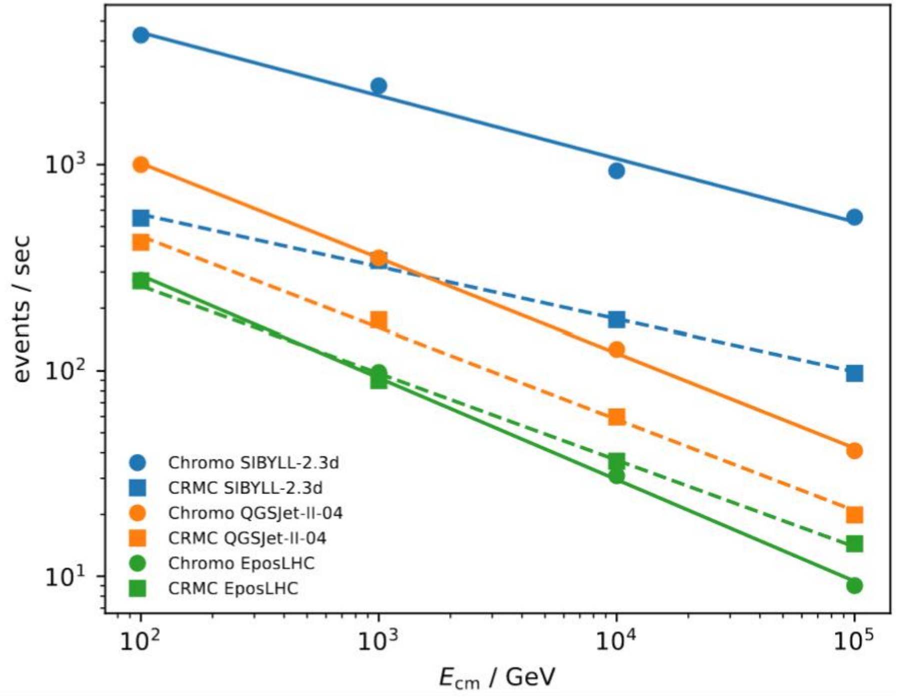
- Interface mimics CRMC to ease transition
- Powered by Python libraries: argparse, rich
  - Comprehensive help output and a flexible system to select models via a string
  - Informative summary of setup
  - Progress bar with ETA, events / sec
- Generate output in HepMC format, ROOT, or generate SVG images

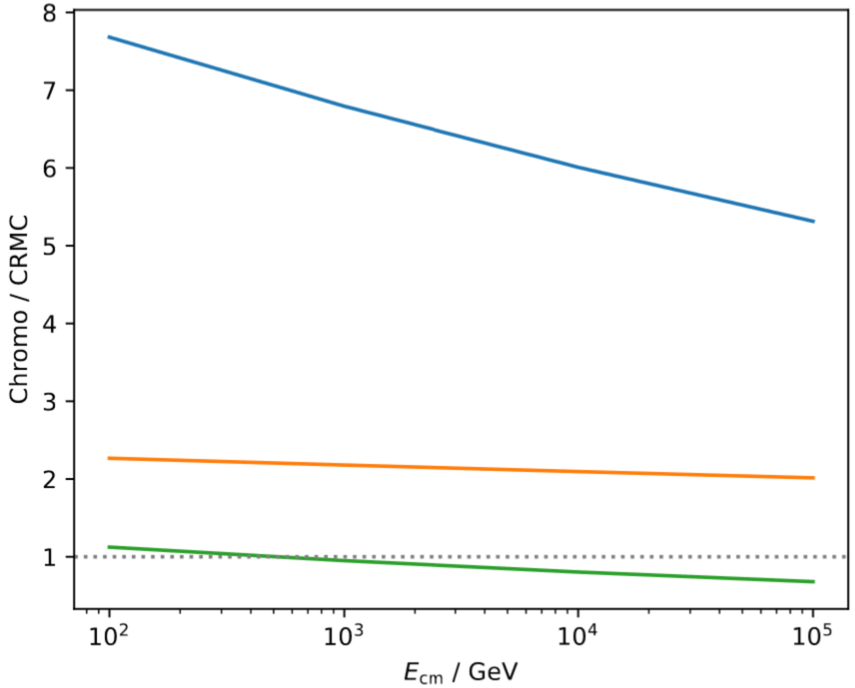
```
(env_impy) -bash-4.2$ chromo -m sibyll-2.1 -n 10000 -S 1000 -o root -f hey.root
╭──────────────────────────────── chromo 0.4.0 ────────────────────────────────╮
│   Model           SIBYLL-2.1                                                 │
│   Projectile      p (2212)                                                   │
│   Target          p (2212)                                                   │
│   sqrt(s)         1000 GeV                                                   │
│   Collisions      10000                                                      │
│   Seed            1225156269                                                 │
│   Format          root                                                       │
╰──────────────────────────────────────────────────────────────────────────────╯
```

```
     3    6.310E+06    11.00   267.28   174.01    36.76    0.125    0.140    8.687
     3    7.943E+06    11.38   275.24   178.72    37.64    0.125    0.133    8.756
     3    1.000E+07    11.77   283.28   183.48    38.54    0.125    0.126    8.819
 10000/10000 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% ETA 0:00:08 1271/s
 (env_impy) -bash-4.2$
```

Performance: Chromo vs CRMC

- Event rate for pp collisions as a function of the center-of-mass energy (Intel 2.8 GHz Quad-Core i7)
- Chromo starts slower but more efficiently output to disc (buffering)





- Python code “glue” fast compiled libraries written in Fortran/C++
- Runtime is limited below by the runtime of Fortran/C++ code performance of wrapped event generator
- NumPy array view (pointers) into hepevt common block if possible
- Avoid copy and hot Python loops
- Buffering of output
- Further optimization: put all heavy lifting of EventKinematics into C++ code

Prior work: CRMC

Cosmic Ray Monte Carlo Package, CRMC

Ulrich, Ralf [1]; Pierog, Tanguy [1]; Baus, Colin [1]

- CRMC: Command-line interface written in C++
  - Used by ATLAS, CMS, LHCb, NA61, TOTEM
- Source compilation required, no binary packages
- Output in ROOT, HepMC, LHE formats
- No direct access to event record from Python or other language
- No built-in event visualization
- Extra models only in Chromo
  - SIBYLL-(2.1, 2.3, 2.3c), SOPHIA, Pythia-8.3, UrQMD-3.4
- Models not in Chromo
  - HIJING, GHEISHA (outdated), UrQMD 1.3 (outdated)

Workflow

Installation

```
(env_test) -bash-4.2$ pip install chromo
Collecting chromo
Downloading chromo-0.4.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (23.7 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 4.7 MB/s eta 0:00:00
Installing collected packages: chromo
Successfully installed chromo-0.4.0
```

Typical workflow

```python
from chromo.kinematics import CenterOfMass
from chromo.models import EposLHC

kinematics = CenterOfMass(100, "p", "p")
event_generator = EposLHC(kinematics)

for event in event_generator(1000):
    # process the result of the collision
    # represented by ‘event‘ object
```

Event under the hood

```python
class EventData:
    """
    Data structure to keep filtered data.
    """
    generator: Tuple[str, str]
    kin: EventKinematics
    nevent: int
    impact_parameter: float
    n_wounded: Tuple[int, int]
    pid: np.ndarray
    status: np.ndarray
    charge: np.ndarray
    px: np.ndarray
    py: np.ndarray
    pz: np.ndarray
    en: np.ndarray
    m: np.ndarray
    vx: np.ndarray
    vy: np.ndarray
    vz: np.ndarray
    vt: np.ndarray
    mothers: Optional[np.ndarray]
    daughters: Optional[np.ndarray]
```

Event properties

```python
class EventData:
    …
    @property
    def p_tot(self):
        """Return total momentum in
    GeV/c."""

    @property
    def eta(self):
        """Return pseudorapidity."""

    @property
    def y(self):
        """Return rapidity."""

    @property
    def xf(self):
        """Return Feynman x_F.""“

    …
```

Example: multiplicity

Import libraries

```
from chromo.constants import TeV
from chromo.kinematics import CenterOfMass
from chromo.models import Sibyll23d, DpmjetIII193, EposLHC
```

Prepare histograms (our choice boost.histogram)

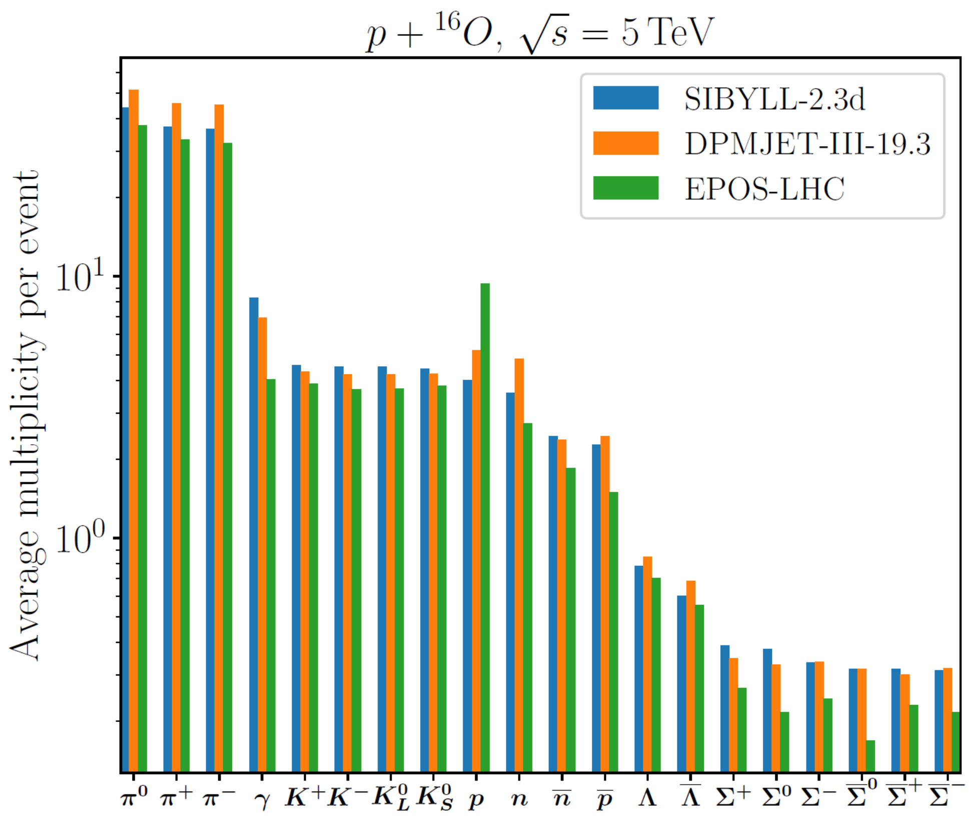
```
import boost_histogram as bh
hist_pid = bh.Histogram(bh.axis.StrCategory([], growth=True),
                        bh.axis.IntCategory([], growth=True))
```

Set kinematics and compared models

```
kinematics = CenterOfMass(5*TeV, "proton", "O16")
models = [Sibyll23d, DpmjetIII193, EposLHC]
nevents = 1000
```

Initialize models in the loop and generate 1000 event for each

```
for model in models:
    event_generator = model(kinematics, seed=1)
    for event in event_generator(nevents):
        event_fs = event.final_state()
        hist_pid.fill(event_generator.pyname, event_fs.pid)
```



Example: other distributions

Import libraries

```python
import chromo
from chromo.constants import TeV
import numpy as np
import boost_histogram as bh
import matplotlib.pyplot as plt
```

Prepare histograms (our choice boost.histogrgam)

```python
pid_categories = bh.axis.IntCategory([2212, 111, 211, -211])
hist_xf  = bh.Histogram(pid_categories, bh.axis.Regular(50, -1, 1))
hist_eta = bh.Histogram(pid_categories, bh.axis.Regular(50, -7, 7))
```

Initialize an event generator instance

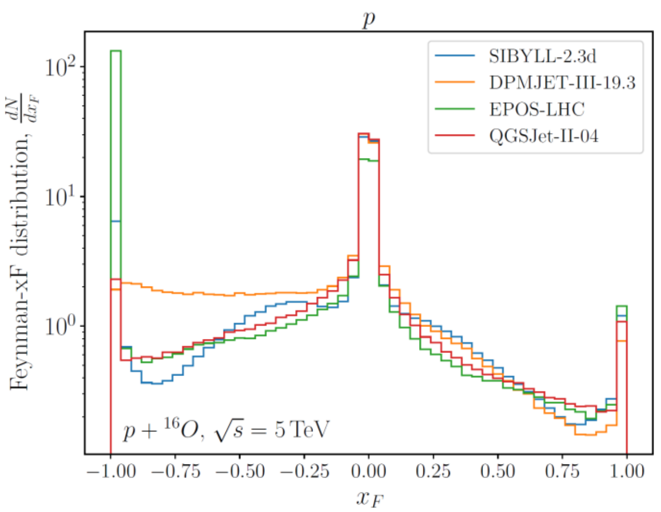
```python
kinematics = chromo.kinematics.CenterOfMass(5 * TeV, "proton", "O16")
event_generator = chromo.models.Sibyll23d(kinematics)
```

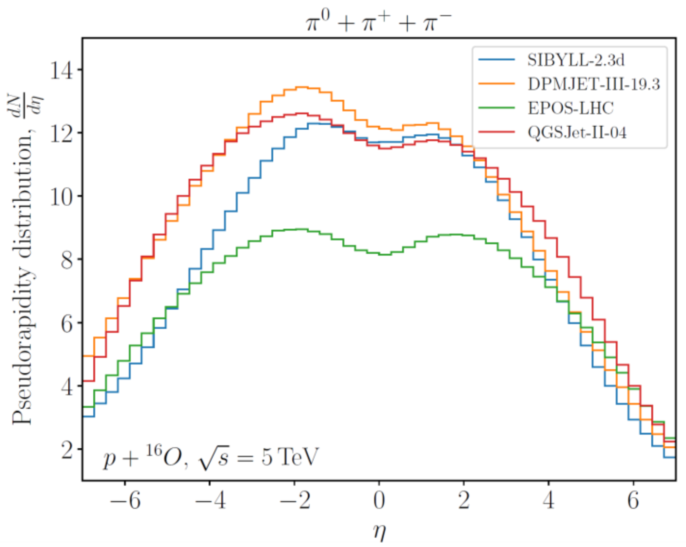
Generate 10000 events

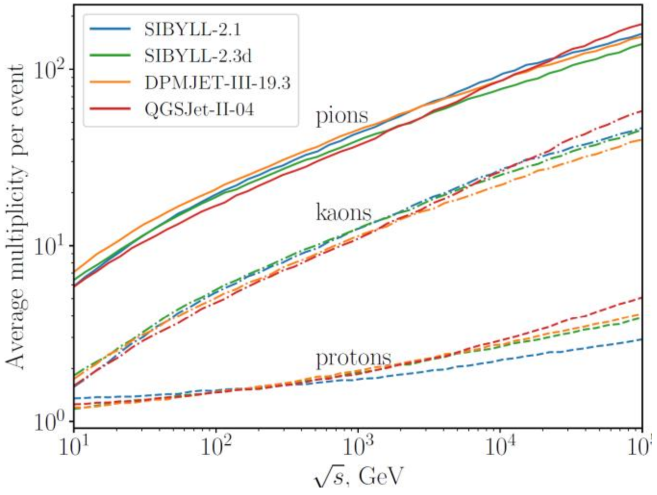
```python
for event in event_generator(10000):
    event = event.final_state()
    hist_xf.fill(event.pid, event.xf)    # Feynman-x distributions
    hist_eta.fill(event.pid, event.eta) # Pseudorapidity distributions
```

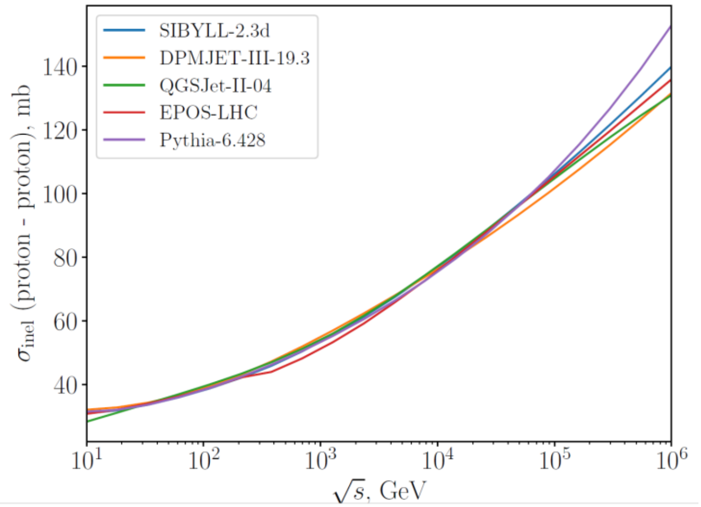
Plot Feynman-x distribution for protons

```python
xf_grid = hist_xf.axes[1]
prot_hist = hist_xf.values(True)[0, 1:-1]
prot_xf_dist = prot_hist / 10000 / xf_grid.widths
plt.stairs(prot_xf_dist, xf_grid.edges)
```









Additional options

- All generated particles are checked to follow stable/unstable settings
- Some generators do not decay particles
- Pythia8 is used to decay any particles via **Pythia8DecayHandler**
- Set by default only for **QGSJet** as it incur some overhead

To configure an `QGSJetII04` event generator to treat charged pions(PDG ID 211 and -211) and muons (PDG ID 13 and -13) as stable particles in the final state:

```
evt_kin = FixedTarget(100, "p", "p")
generator = QGSJetII04(evt_kin)
generator.final_state_particles = [211, -211, 13, -13]
# for any other generator:
self._activate_decay_handler(on=True)
generator.final_state_particles = select_long_lived(tau_stable)
```

- History (mother and daughter particles) of some event generators (e.g. DpmJet) are repaired and rectified before output to be a valid HepMC event and be able to processed by Rivet

Repair event history and pre-/appending beam particle info can be optionally disabled to save some CPU time

```
generator._restore_beam_and_history = False
```

Summary

- **Easy** comparisons between a wide variety of event generators
- **Easy** visualization and manipulation of events using rich Python ecosystem
- **Easy** installation**:** automated packaging and distribution of binaries via PyPI for Linux, MacOS, and Windows
  - Excellent choice for application and education in (astro)particle physics
- **Easy change** of simulation settings (on-the-fly)
- **Command-line interface**
  - Mimics CRMC to ease transition
- **Fast** thin wrapper, processing optimized
- **Output in standard formats**
  - HepMC (via pyhepmc), optionally gzipped
  - Root (via uproot)
  - SVG images
- Used in cosmic ray community , high-energy neutrino physics (IceCube), and HEP community (LHCb)
- **To-do**
  - Finish packaging for Windows (Pythia 8)
  - Add LHE output (via pyhepmc)
  - Optimize for fast the changes in kinematics
  - Add parameter settings for some generators (Pythia 8)
  - Add more event generators, e.g. EPOS 4.0, Fluka