- assistant professor at the University of Cincinnati
- focused on both experiment and pheno
- physics interests ranging from QCD to dark sectors

LHCb

CODEX-b

HepMC3

AST
ARK

MLhaD

1. specify beam types and momenta
2. request specific physics processes to generate
3. generate momenta for outgoing particles from collision

MC generators bridge the gap between first principle and pheno calculations

- 1978: JetSet from the Lund theory group
- 1997: merged into Fortran based Pythia 6
- 2004: rewrite into C++ began
- 2007: first release of C++ based Pythia 8.1
- 2014: mature Pythia 8.201 released
- 2019: adopted `C++11` standard with Pythia 8.301
- 2023: release of Pythia 8.310 yesterday

. . . another development was that the newly-formed LEP collaborations mainly opted for JETSET

. . . [other] programs had been written on the DESY mainframe, making extensive but inefficient use of existing DESY software, such that they did not fit in the smaller CPU memory of the CERN mainframe at the time.

. . . an ambition remained for programs to be designed to run standalone and have a modest footprint.

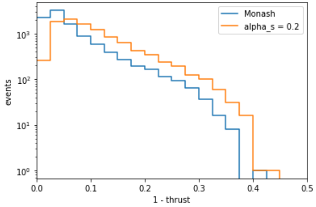
. . . Lund was a remote isolated place in the days before Internet, making the effort spent on writing detailed manuals essential for successful usage elsewhere.

❶ **no external dependencies**

❷ **comprehensive documentation**

- published physics manuals (6.4 at 576 pages and 8.3 at 315 pages)
- technical manual shipped with each release
- DOXYGEN reference avalable at `pythia.org`
- issue tracker (269 issues answered and growing)
- examples and 112 and growing shipped with each release
- tutorials for summer schools and workshops

```
In [14]: # Compare the two thrust histograms.
         plot([monash.hist, varAlpha.hist],
              xlim = (0, 0.5), ylog = True, xlabel = "1 - thrust", ylabel = "events")
```



Try using a few different values of α_s to see what happens with the distribution. For example, we could try setting α_s to a lower value than the Monash tune, so something like 0.1. In the cells above, we can do this by creating another `AnalyzeThrust` object and then plotting all the different distributions using the command `plot([analysis1.hist, analysis2.hist, analysis3.hist, ...])`. From a model perspective, try to build an intuition as to why the thrust changes as it does for these different parameter values.

We can also explore different values of the shower cut-off value sepcified by the parameter `TimeShower:pTmin`. This is by default set to 0.5 GeV, but we can change it to something higher, like 2 GeV. We can also try lower values as well, although at some point this will cause failures in the hadronization process.

- 8.219 – 8.245: monolothic block of bindings automatically generated per release with SWIG
- 8.301 – present: use bespoke DOCKER container with BINDER installation to automatically generate PYBIND11 bindings
  - bindings can be regenerated by users, including user code
  - partial (default) or full bindings can be generated
  - custom BINDER configuration can be used
  - support for both PYTHON 2 and 3

```
cd plugins/python/ && ./generate
cd - && ./configure --with-python && make
```

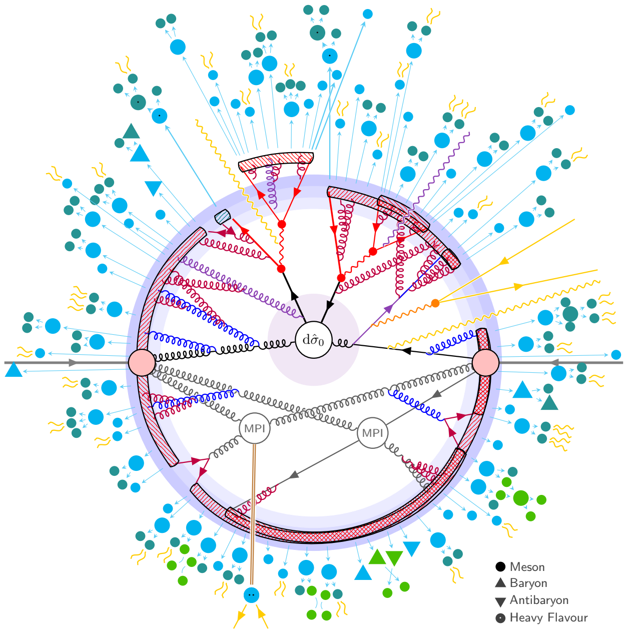
```
void bind_Pythia8_Basics(std::function< pybind11::module &(
    std::string const &namespace_) > &M) {
  { // Pythia8::Vec4 file:Pythia8/Basics.h line:32
    pybind11::class_<Pythia8::Vec4, std::shared_ptr<Pythia8::Vec4>> cl(
      M("Pythia8"), "Vec4", "");
    pybind11::handle cl_type = cl;
    cl.def( pybind11::init( [](){ return new Pythia8::Vec4(); } ), "doc" );
    cl.def( pybind11::init( [](double const & a0){
          return new Pythia8::Vec4(a0); } ), "doc" , pybind11::arg("xIn"));
    cl.def( pybind11::init( [](double const & a0, double const & a1){
          return new Pythia8::Vec4(a0, a1); } ),
      "doc" , pybind11::arg("xIn"), pybind11::arg("yIn"));
  }
}
```

- support for multi-threaded `PythiaParallel`

```
# Configure Pythia.
pythia = pythia8.PythiaParallel()
pythia.readString("HardQCD:all = on")

# Configure the parallel runs.
pythia.readString("Parallelism:numThreads = 4")
pythia.readString("Parallelism:processAsync = off")
pythia.readString("Main:numberOfEvents = 10000")

# Define the histogram to fill.
mult = pythia8.Hist("charged multiplicity", 100, -0.5, 799.5)

# Initialize (can pass custom initializer).
pythia.init(init)

# Generate events.
def analyze(pythiaNow):
    mult.fill(pythiaNow.event.nFinal(True))
pythia.run(analyze)

# Finish.
pythia.stat()
print(mult)
```

- bi-directional support for relevant classes like `UserHooks`

```
# Write own derived UserHooks class.
class MyUserHooks(pythia8.UserHooks):

    # Constructor creates anti-kT jet finder with (-1, R, pTmin, etaMax).
    def __init__(self):
        pythia8.UserHooks.__init__(self)
        self.slowJet = pythia8.SlowJet(-1, 0.7, 10., 5.)
        self.pTHat   = 0.

    # Allow process cross section to be modified...
    def canModifySigma(self):
        return True

    # ...which gives access to the event at the trial level, before →
         selection.
    def multiplySigmaBy(self, sigmaProcessPtr, phaseSpacePtr, inEvent):
        return 1

# Generator.
pythia = pythia8.Pythia()

# Set up to do a user veto and send it in.
myUserHooks = MyUserHooks()
pythia.setUserHooksPtr(myUserHooks)
```


$d\hat{\sigma}_0$
MPI
MPI
● Meson
▲ Baryon
▼ Antibaryon
⊙ Heavy Flavour

- `UserHooks` allow interaction with PYTHIA at steps throughout the generation
- other classes also allow for modification of PYTHIA, *e.g.* `ShowerModel`, `PDF`, . . .
- new plugin system allows for runtime loading of user classes
- allows for external dependencies in PYTHON interface without regenerating bindings

```
pythia.readString("Init:plugins = {libMyPlugins.so::MyUserHooks}")
```

- can also return shared pointer in C++

```
shared_ptr<PDF> pdf = make_plugin<PDF>("libMyPlugins.so", "MyPDF");
```

- simple macro definitions for defining plugin libraries

```
PYTHIA8_PLUGIN_CLASS(BaseClass, MyClass, RequirePythia, RequireSettings,
  RequireLogger)
PYTHIA8_PLUGIN_XML("pluginLibraryName/xmldoc/startFile.xml")
PYTHIA8_PLUGIN_VERSIONS(8310, 8311)
```

- similar in principle to FastJet-Contrib, but more complicated ecosystem
- provide template generation and general requirements and guidelines
- give contributors full control of their own repository under the `pythia8-contrib` group on `gitlab.com`

```
cd template
./generate PACKAGE CLASS/NAME ... [OPTIONS]
```

- single point of entry for users to build plugins

```
cd contrib
./enable PACKAGE[/VERSION] ... [OPTIONS]
./configure [OPTIONS]
make
```

Pythia-Contrib is in alpha phase, but let me know if you are interested

1. any general feedback or wisdom is appreciated
2. we don't provide a Pythia Python package (although others do), should we?
3. can we improve the Python interface to Pythia, without introducing a significant maintenance overhead or external dependencies?
4. can we provide interoperability with the HEP Python ecosystem?
5. do we expand the Pythia-Contrib platform to include Python contributions?