Kolja Kauder

Center for Frontiers in Nuclear Science

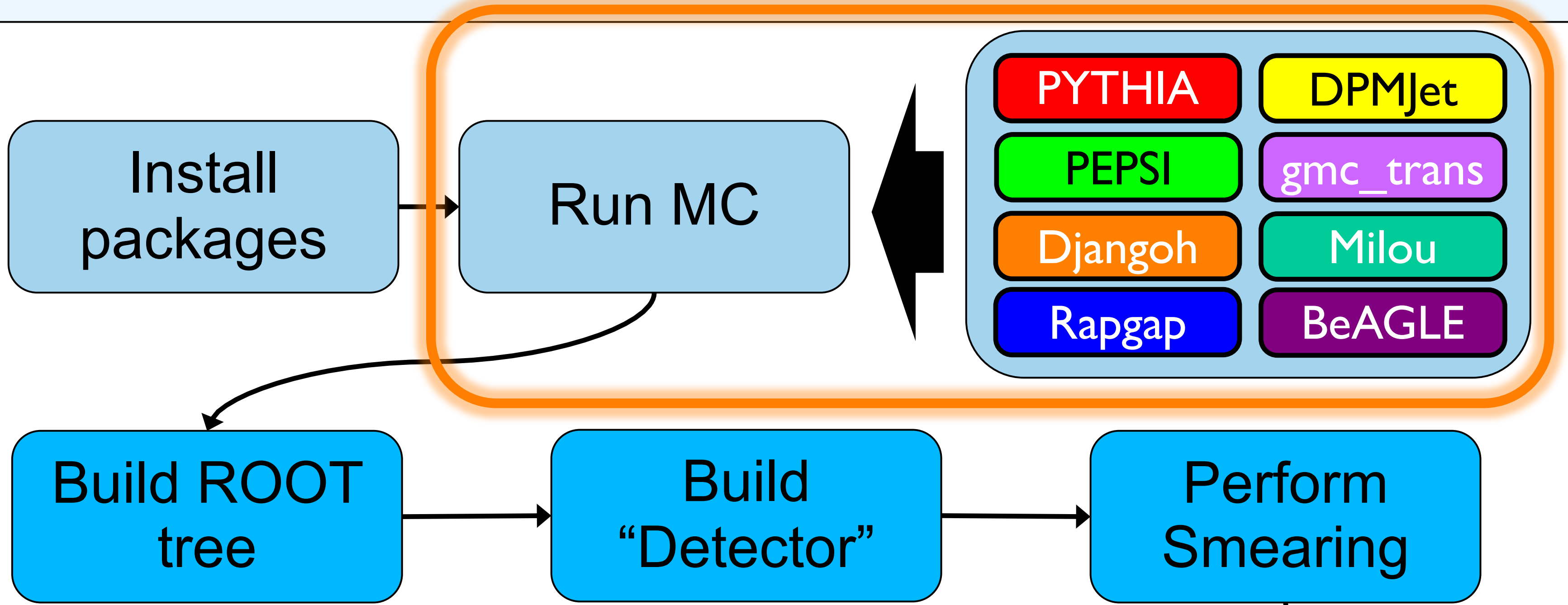
BROOKHAVEN NATIONAL LABORATORY

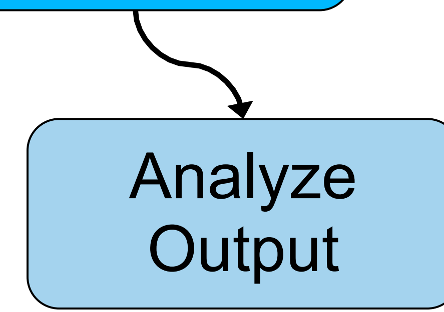
EIC-Smear and MC Generators to Fun4All

ECCE Simulation Workshop
Online, March 2021

U.S. DEPARTMENT OF ENERGY | Office of Science

Workflow





- Standard Format: analyze different generators without change
- Truth ↔ Smeared: Friend Trees

Supported Generators in eic-smear

- HERA era generators
- Any others using the Lund-style format, e. g. BeAGLE
- HepMC2 or HepMC3 data

Tested for Pythia8, others may need minor tweaking (HEPMC for DIS is not rigorously defined)

→ any of the above can be gzipped (factor ~6 compression)

https://gitlab.com/eic/mceg
https://eic.github.io/software/mcgen.html

PYTHIA6
DPMJet
PEPSI
Sartre
Djangoh
Milou
Rapgap
BeAGLE
Derivatives

Any HepMC (2 & 3):
PYTHIA8
eSTARLIGHT,
HERWIG,
SHERPA, …

Multi-purpose e+P

PYTHIA-RAD-CORR

- e+P collisions, no polarization
- Based on pythia 6.4.28 with radiative corrections
- Tuned for HERA and improved for the EIC

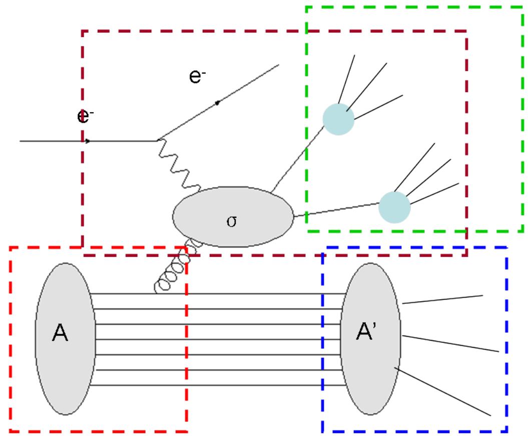
```
$ pythiaeRHIC < ep_hiQ2.20x250.small.txt.gz > log.txt
```

https://eic.github.io/software/pythia6.html

Multi-purpose e+A

DPMJet

- Based on the Dual Parton Model (DPM)
- using Glauber, FLUKA for nuclear fission



→ **BeAGLE** (via the deprecated DPMJetHybrid)

- Benchmark eA Generator for LEptoproduction
- Added pythia, quenching, LHAPDF

```
$ setenv BEAGLESYS $EICDIRECTORY/PACKAGES/BeAGLE
$ mkdir outForPythiaMode

$ ln -s $BEAGLESYS/nuclear.bin .
$ ln -s $BEAGLESYS/eAt1noq .
$ $BEAGLESYS/BeAGLE < steer.inp > out.log
```

https://eic.github.io/software/beagle.html

Polarized e+P and e+A

PEPSI https://eic.github.io/software/pepsi.html

- Polarized DIS, evolved from LEPTO
- Deprecated for polarized in favor of

```
$ ln -s $EICDIRECTORY/PACKAGES/PEPSI/pdf
$ pepsieRHICnoRAD < steer.txt > out.log
```

→ DJANGOH

- NC and CC events including QED and QCD radiation
- Upgraded to polarization and nuclear PDFs

```
$ # Update LHAPDF pdfset locations in the steer file
$ djangoh < steer.txt > out.log
```

https://eic.github.io/software/djangoh.html

DVCS, Vector Mesons, e+P and e+A

MILOU

https://eic.github.io/software/milou.html

- DVCS based on generalized parton distributions (GPDs) evolved to next-to-leading order
- 32 bit only
- Hard-coded output name

```
$ ln -s $EICDIRECTORY/PACKAGES/milou/*.dat .
$ $EICDIRECTORY/bin/milou
```

Sartre

- exclusive diffractive vector meson production and DVCS in e+p and e+A
- Based on dipole model, bSat and bCGC, with saturation

https://sartre.hepforge.org

and **eSTARlight!** cf. later slide

That was the 30k ft view!

- Or 10 km

The **right choice of generator,**

and the **right choice of steering cards** is way outside the scope of this talk (and my expertise)

- The community, and the YR conveners, are an invaluable resource

Common Lund-style ASCII Format

```
<generator name> EVENT FILE
============================================
<generator-specific event variable names>
============================================
Track variable names
============================================
0 <generator-specific event data>
============================================
1 KS KF parent child1 childN px py pz E m x y z
2 KS KF parent child1 childN px py pz E m x y z
...
N KS KF parent child1 childN px py pz E m x y z
=============== Event finished ===============
... <repeat event structure>
```

6-line file header

(N_{tracks}+3)-line event

https://eic.github.io/software/pythia6.html#output-file-structure

Observations

- Format does not contain cross section
  → **keep the log file** (stdout!), eic-smear can extract from that
- Generator-specific variables need to be handled **case by case**
  → eic-smear does that
- No detailed agreement on **placement** of beams, virtual particle, scattered particles, …
  → eic-smear handles that case by case
- **No reader class** or other interface to analysis frameworks
  → In practice, that IS eic-smear

Also supports LEPTO, LEPTO-PHI, gmc_trans, comptonRad, generic generators using this format

Note to MCEG developers: Please use HepMC3 going forward
→ eic-smear can handle that too

e+A via HepMC

eSTARlight

- Coherent vector meson photo- and electro-production in e+A collisions
- Compare to STARlight in UPC AA
- Includes template analysis

```
$ # adapt slight.in
$ estarlight output.txt
$ # event record is in slight.out and slight.hepmc
```

https://github.com/eic/estarlight#estarlight

The Big Three (via HepMC)

	NLO Matching	Multijet Merging
Herwig 7	Internally automated	Internally automated
Pythia 8	External	Internal, ME via event files
Sherpa 2	Internally automated	Internally automated

- NLO QCD corrections: **“Off the shelf”**
- NNLO starts to become available for more and more processes

Focus of last two decades: **LHC**

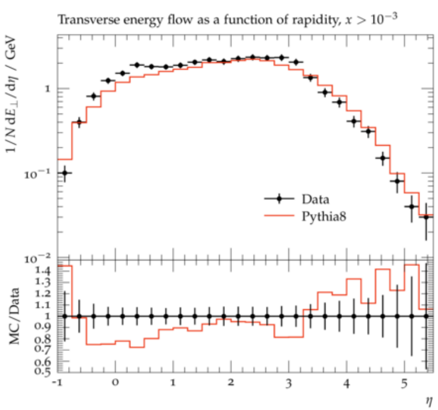
- **lesson learned** high-precision QCD measurements require high-precision MCEGs
- MCEG not about tuning but about physics

ready to work on ep/eA

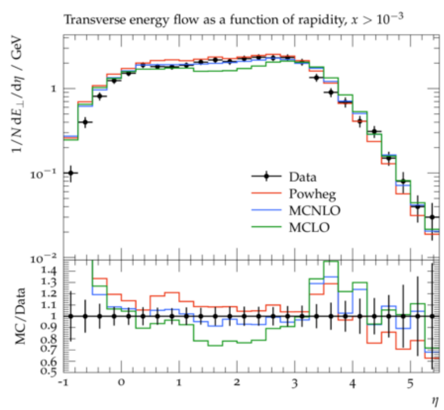
Compare MCEGs Results with HERA Data

Current SWG activity

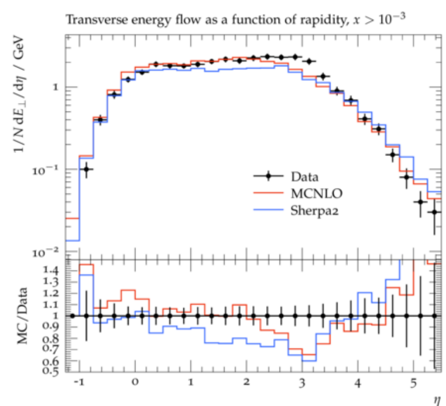
- Comparison to published results using RIVET and understand differences
- **Provide initial findings and results in a EICUG report**
- Overview of where we stand in understanding HERA data with current physics and models implement in MCEGs



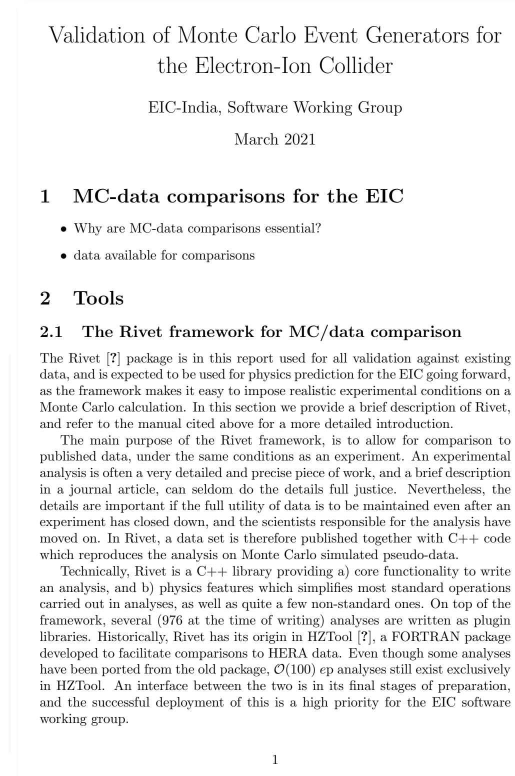
Pythia8



Herwig7



Sherpa2



Validation of Monte Carlo Event Generators for the Electron-Ion Collider

EIC-India, Software Working Group

March 2021

1 MC-data comparisons for the EIC

- Why are MC-data comparisons essential?
- data available for comparisons

2 Tools

2.1 The Rivet framework for MC/data comparison

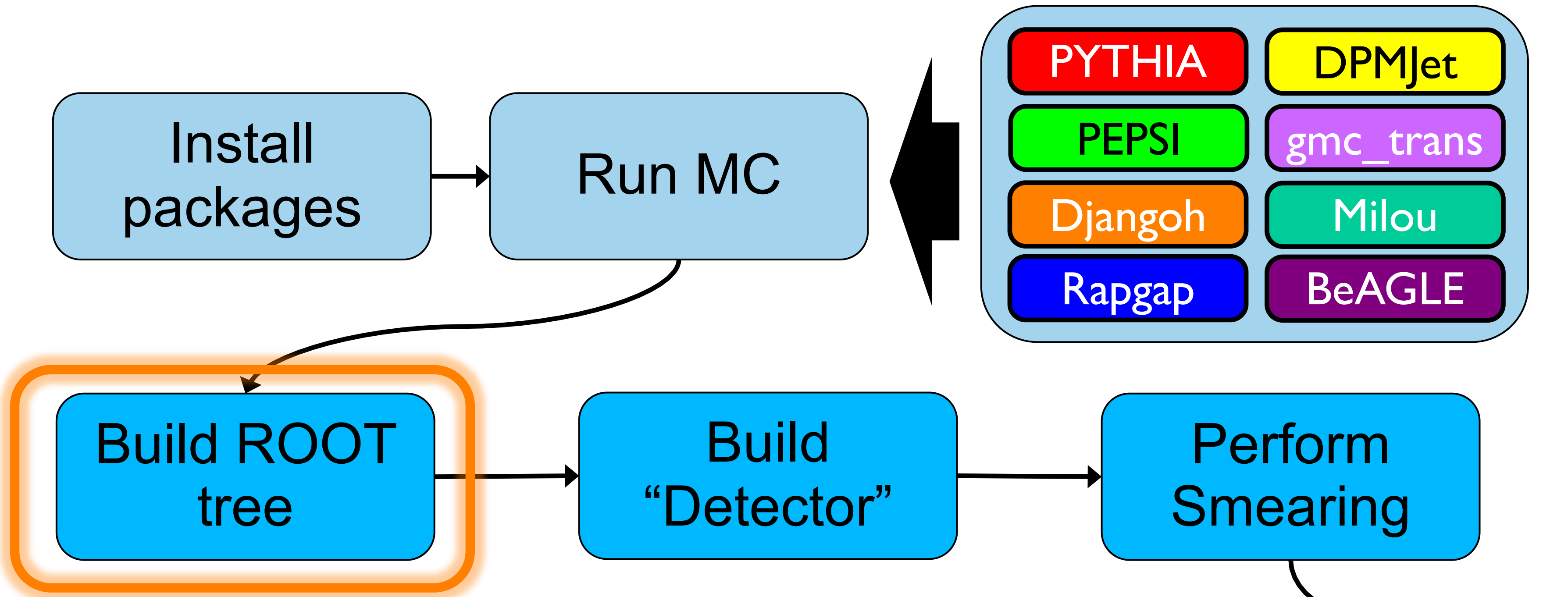
The Rivet [?] package is in this report used for all validation against existing data, and is expected to be used for physics prediction for the EIC going forward, as the framework makes it easy to impose realistic experimental conditions on a Monte Carlo calculation. In this section we provide a brief description of Rivet, and refer to the manual cited above for a more detailed introduction.

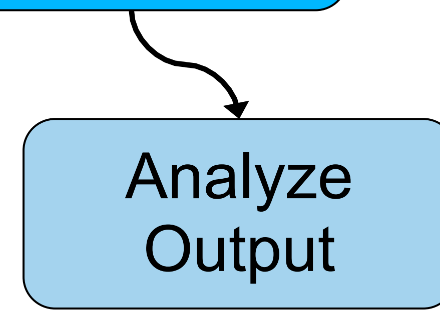
The main purpose of the Rivet framework, is to allow for comparison to published data, under the same conditions as an experiment. An experimental analysis is often a very detailed and precise piece of work, and a brief description in a journal article, can seldom do the details full justice. Nevertheless, the details are important if the full utility of data is to be maintained even after an experiment has closed down, and the scientists responsible for the analysis have moved on. In Rivet, a data set is therefore published together with C++ code which reproduces the analysis on Monte Carlo simulated pseudo-data.

Technically, Rivet is a C++ library providing a) core functionality to write an analysis, and b) physics features which simplifies most standard operations carried out in analyses, as well as quite a few non-standard ones. On top of the framework, several (976 at the time of writing) analyses are written as plugin libraries. Historically, Rivet has its origin in HZTool [?], a FORTRAN package developed to facilitate comparisons to HERA data. Even though some analyses have been ported from the old package, $\mathcal{O}(100)$ ep analyses still exist exclusively in HZTool. An interface between the two is in its final stages of preparation, and the successful deployment of this is a high priority for the EIC software working group.

1

Workflow





- Standard Format: analyze different generators without change
- Truth ↔ Smeared: Friend Trees

Transformation

- Generated 1M PYTHIA6 events, MinBias, $Q^2 = 10 - 100\ \mathrm{GeV}^2$
  - ~ 1 hr to generate
- Transformed with `BuildTree`
  - ~ 17 min to transform

Large file (9GB).
Better to gzip afterward

Needed for cross-section

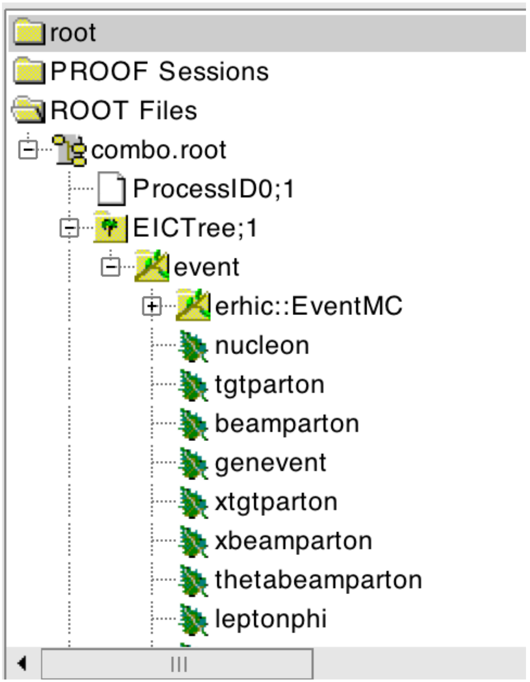
```
$ root -l
root [0] gSystem->Load("libeicsmear")
root [1] BuildTree("pythia.txt.gz",".",-1,"log.txt")
```

Or compile:

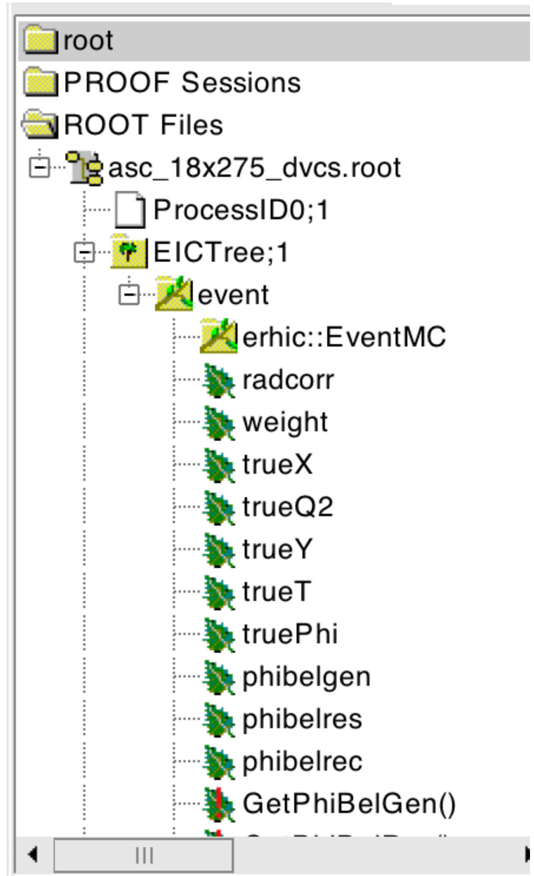
```
$ g++ `root-config --cflags` \
`root-config --libs` \
-I<…>/include -L<…>/lib -leicsmear …
```

```
#include <TSystem.h>
#include <eicsmear/functions.h>
int main(){
  gSystem->Load("libeicsmear");
  BuildTree("largepythia.txt", ".", -1);
  return 0;
}
```

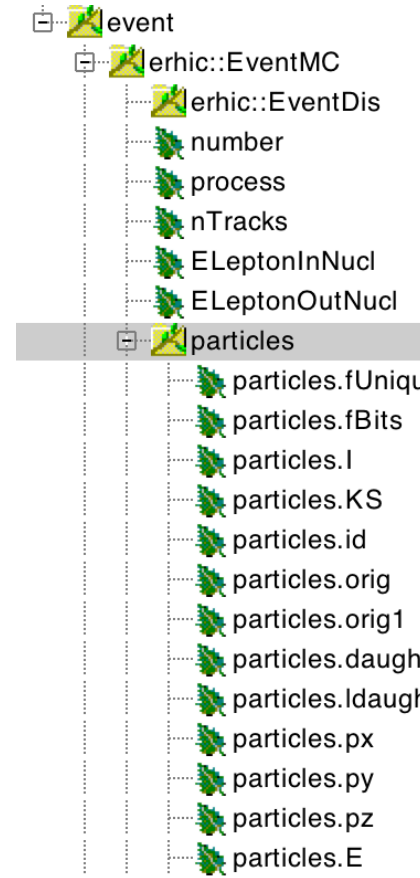
ROOT Format



PYTHIA event

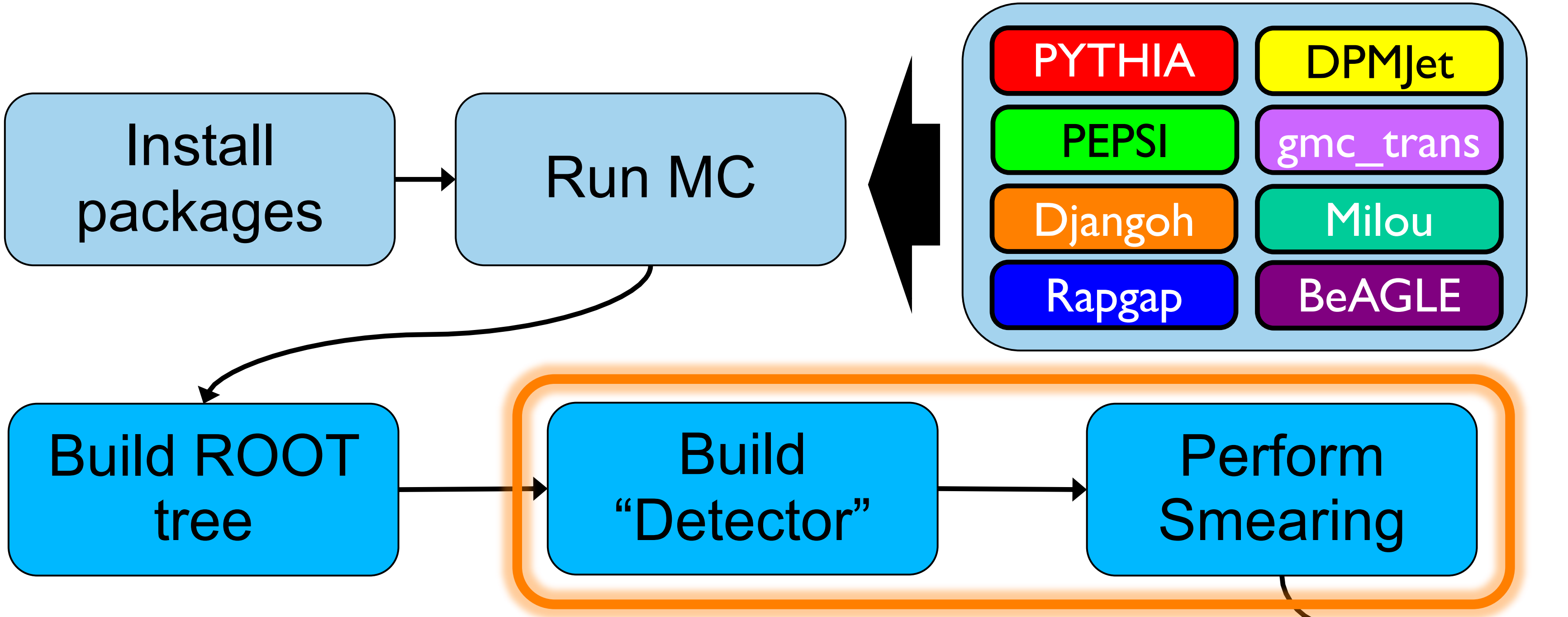


MILOU event

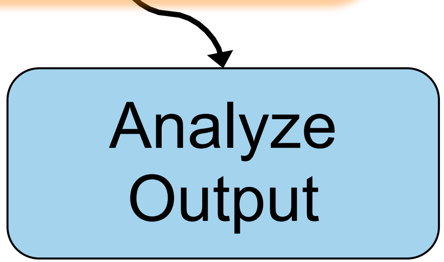


Particles

Workflow



- Standard Format: analyze different generators without change
- Truth ↔ Smeared: Friend Trees



Smearing

"Smearer" defines some element of performance + acceptance

NOT a "physical detector"

Represents the **overall performance** in measuring a quantity.

- Standard Smearers are provided
- Define your own via inheritance

(single) quantity, X, to smear: E, p, θ, φ + Function defining σ(X) = f([E, p, θ, φ]) + Acceptance for X in E, p, θ, φ, pT, pZ

Smearer

Smearer

Smearer

Smearer

Smearer

→ "Detector"

- Smearers applied to **each final particle**
- Optionally, recalculate **derived values** e. g. x, Q^2

In Practice

BeAST Calo parameters from eRHIC design study

```
Smear::Detector BuildBeAST() {

   // Calorimeter resolution usually given as sigma_E/E = const% + stocastic%/Sqrt{E}
   // EIC Smear needs absolute sigma: sigma_E = Sqrt{const*const*E*E + stoc*stoc*E}

   // Create the EM Calorimeter
   Smear::Device emcalBck(Smear::kE, "sqrt(0.01*0.01*E*E + 0.015*0.015*E)");
   Smear::Device emcalMidBck(Smear::kE, "sqrt(0.01*0.01*E*E + 0.07*0.07*E)");
   Smear::Device emcalMid(Smear::kE, "sqrt(0.01*0.01*E*E + 0.10*0.10*E)");
   Smear::Device emcalFwd(Smear::kE, "sqrt(0.01*0.01*E*E + 0.07*0.07*E)");
```

Function returning a Smear::Detector

Component	Pseudorapidity Range	Resolution
Back EMCal	$-4.5 < \eta < -2$	$\frac{1.5\%}{\sqrt{E}} \oplus 1\%$
Mid-Back EMCal	$-2 < \eta < -1$	$\frac{7\%}{\sqrt{E}} \oplus 1\%$
Mid EMCal	$-1 < \eta < 1$	$\frac{10\%}{\sqrt{E}} \oplus 1\%$
Fwd EMCal	$1 < \eta < 4.5$	$\frac{7\%}{\sqrt{E}} \oplus 1\%$

- Acceptance (in θ)
- Genre selects set of PIDs
  → can be customized

```
   // Set Up EMCal Zones
   Smear::Acceptance::Zone emBck(2.8726,3.1194,0.,    ...
   // Assign acceptance to calorimeters
   emcalBck.Accept.SetGenre(Smear::kElectromagnetic);
```

… same for HCal

Example Continued

```
// Create our tracking capabilities, by a combination of
// momentum, theta and phi Devices.
// The momentum parametrization (a*p + b) gives sigma_P/P in percent.
// So Multiply through by P and divide by 100 to get absolute sigma_P
// Theta and Phi parametrizations give absolute sigma in miliradians

// Track Momentum
Smear::Device momentum(Smear::kP, "(P*P*(0.0182031 + ...
Smear::Device trackTheta(Smear::kTheta, "((1.0/(1.0*P))*( ...
Smear::Device trackPhi(Smear::kPhi, "((1.0/(1.0*P))*(...
```

Tracking follows similar TFormula's.

```
// Create a detector and add the devices
Smear::Detector det;
det.AddDevice(emcalBck);
det.AddDevice(emcalMidBck);
...

// The detector will calculate event kinematics from smeared values
det.SetEventKinematicsCalculator("NM JB DA");
return det;
}
```

Assemble devices

Kinematics options

Object Oriented

- Formulas are good, but eic-smear is completely OO

```
/**
   Smearing class describing ePHENIX momentum resolu

   The ePHENIX momentum resolution is too complicate
   parameterisation via the Smear::Device class.
   Therefore we define a custom Smearer class to imp
   See the email in comments at the end of the file
   resolution values we use.
*/
class EPhenixMomentum : public Smear::Smearer {
public:
  /**
     Destructor.
  */
  virtual ~EPhenixMomentum();
  /**
     Constructor.

     If multipleScattering is true, apply the multip
     the region where it is known, 2 < eta < 4.
     Otherwise apply only the linear resolution term
  */
  EPhenixMomentum(bool multipleScattering = true);
```

Complex Parameterizations

```
namespace Smear{
  class TofBarrelSmearer : public Smear::NumSigmaPid {
  public :

    /** standard ctor
     */
    TofBarrelSmearer( double radius=100, double etaLow=-1.0, double etaHigh=1.0, double sigmaT=10 ){
      ThePidObject = std::make_shared<tofBarrel>(radius, etaLow, etaHigh,sigmaT);
    };
  };
};
```

Wrapper Classes

New Devices

```
/**
 A cylindrical calorimeter.
 Main motivation for this class is realistic angular resolution,
 but for compactness we derive from Device and allow for an E resolution string.
 Information needed is spatial resolution, such as in slide 2 here:
 https://indico.bnl.gov/event/8231/contributions/37910/attachments/28335/43607/talk_eic_yr-cal_2020_05.pdf
 plus geometric and material properties
 */
class BarrelCalo : public Device {
```

Interactive use

- Automatic loading of libraries and version information with small wrapper

```
$ eic-smear
Using eic-smear version: 1.1.2
Using these eic-smear libraries :
/Users/kkauder/software/lib/libeicsmear.dylib
/Users/kkauder/software/lib/libeicsmeardetectors.dylib
eic-smear [0] BuildTree("pythia.txt",".", -1, "log.txt")
```

```
Replaces:
root [0] gSystem->Load("libeicsmear");
root [1] gSystem->Load("libeicsmeardetectors")
```

- Load the script or use shortcut function

```
eic-smear [1] .L SmearCore_0_1_B3T.cxx
eic-smear [2] auto d = BuildCore_0_1_B3T();
# or
eic-smear [1] auto d = BuildByName("coreB3")
```

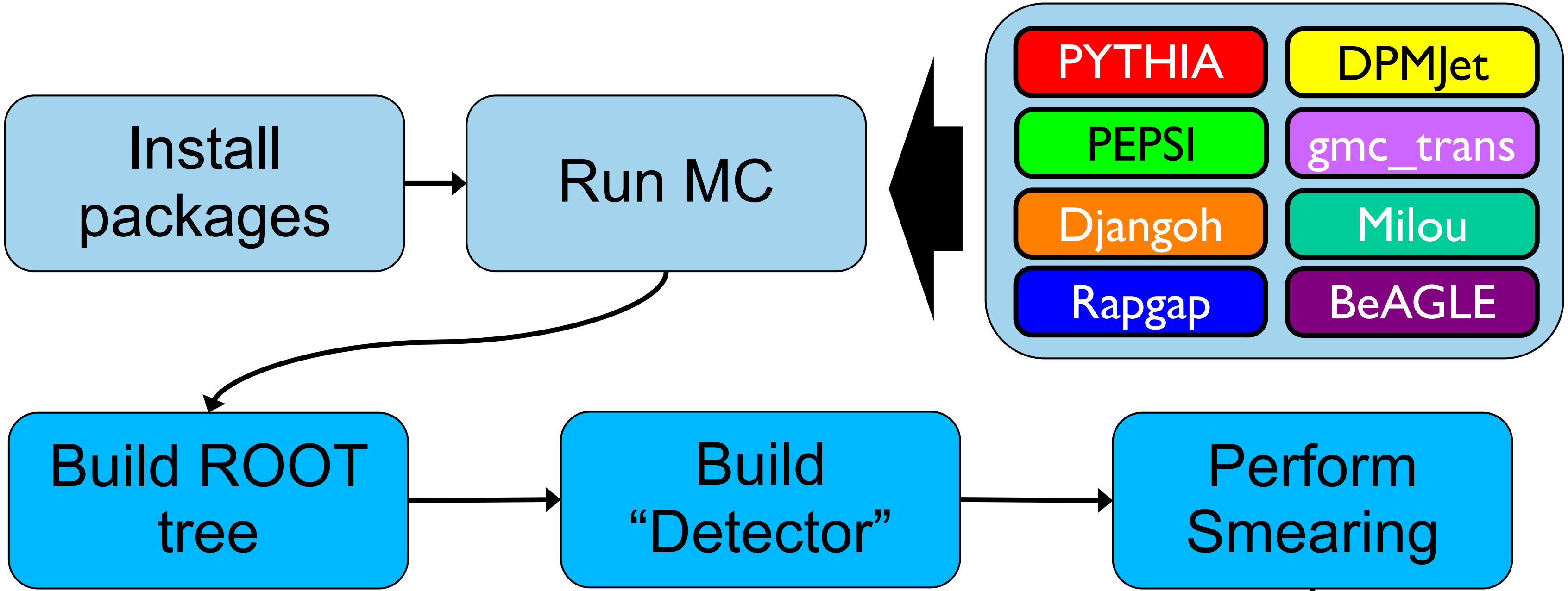
Put together yesterday
NOT official

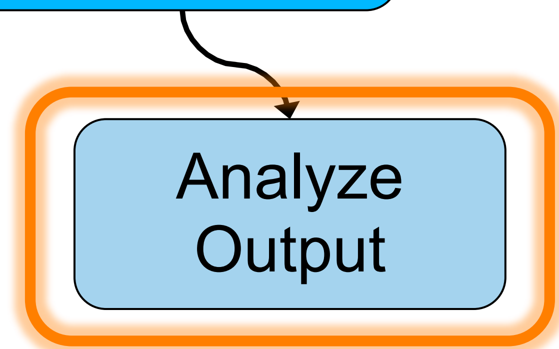
https://github.com/eic/eicsmeardetectors/blob/core_detector/SmearCore_0_1_B3T.cxx

- All in one directly from the shell:

```
$ echo 'SmearTree(BuildByName("coreB3"), "pythia.root")' | eic-smear
```

Workflow





- Standard Format: analyze different generators without change
- Truth ↔ Smeared: Friend Trees

Analysis

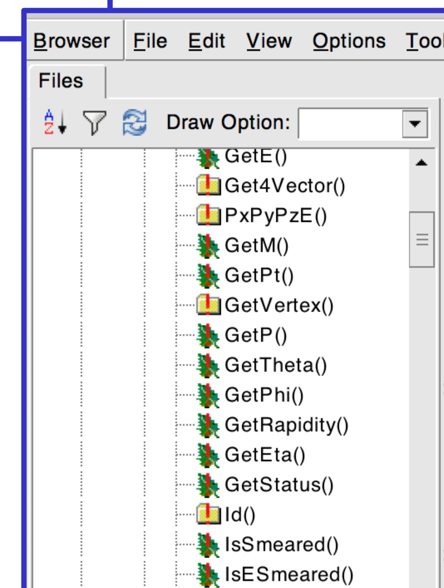
```
TChain* inTree = new TChain("EICTree");
inTree->Add(inFileName1);
inTree->AddFriend("Smeared",inFileName2);

// Setup Input Event Buffer
erhic::EventPythia* inEvent(NULL);     inTree->SetBranchAddress("event",&inEvent);
Smear::Event* inEventS(NULL);          inTree->SetBranchAddress("eventS",&inEventS);
```

Befriend and get matched branches

Particle Loop

```
      const Smear::ParticleMCS* inParticleS = inEventS->GetTrack(j); // Smeared Particle
      const Particle* inParticle = inEvent->GetTrack(j); // Unsmeared Particle
```



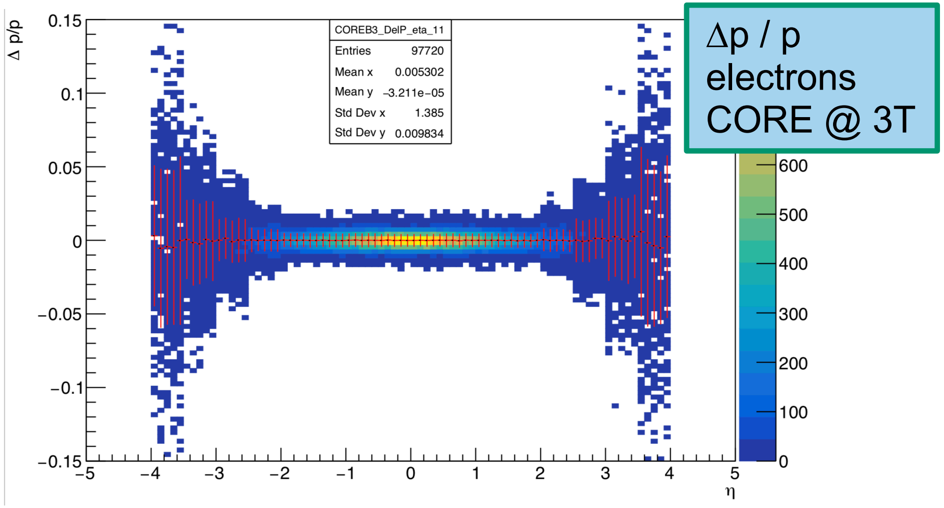
- Access properties:

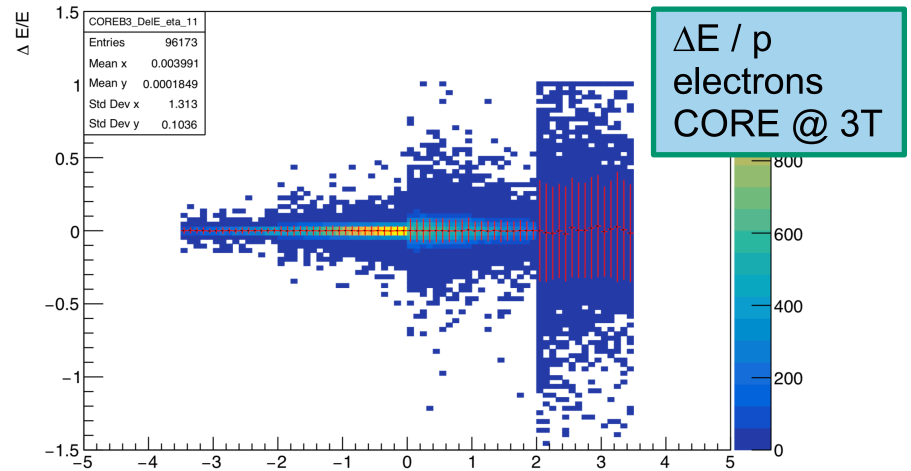
```
if ( inParticleS->IsPSmeared() ){
  auto delP = (inParticle->GetP() - inParticleS->GetP()) / inParticle->GetP();
  coll.DelP_th->Fill(inParticle->GetTheta(), delP);
  coll.DelP_eta->Fill(inParticle->GetEta(), delP);
}
```

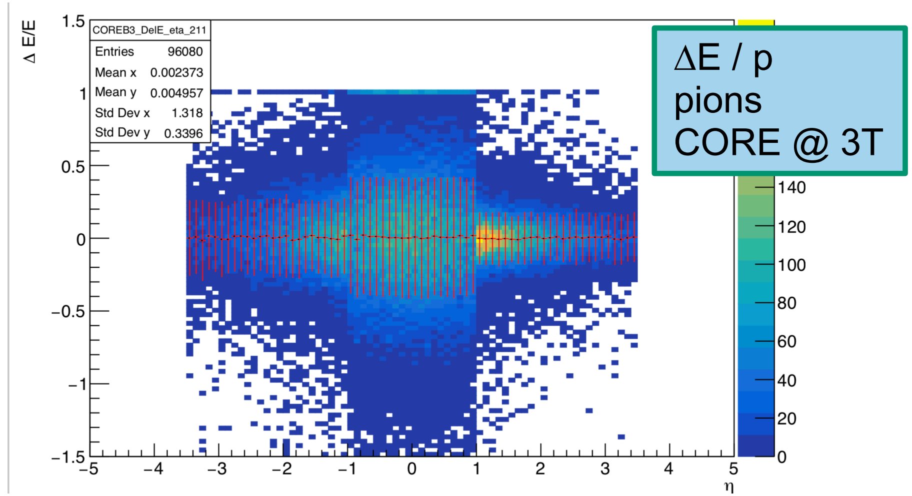
Provided qaplots.cxx demonstrates usage

https://github.com/eic/eicsmeardetectors/blob/master/tests/qaplots.cxx

Particle Gun Examples (CORE)

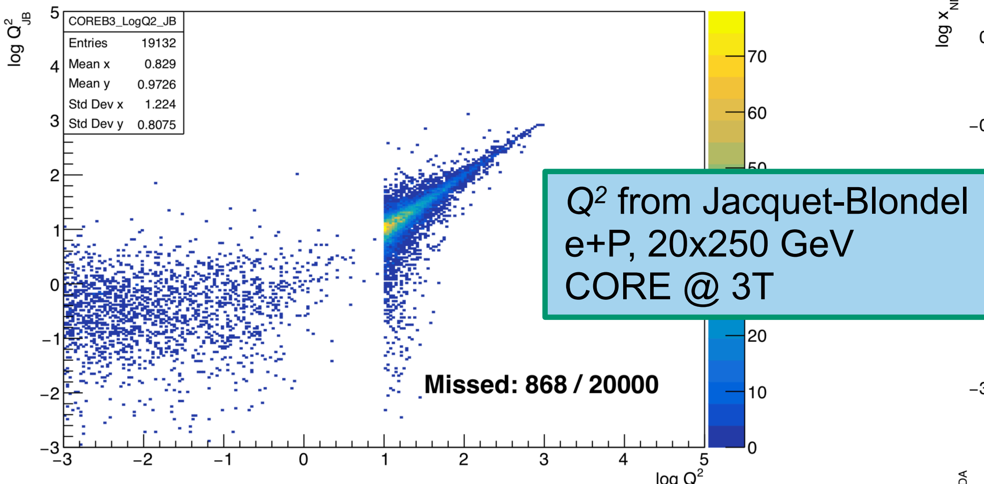


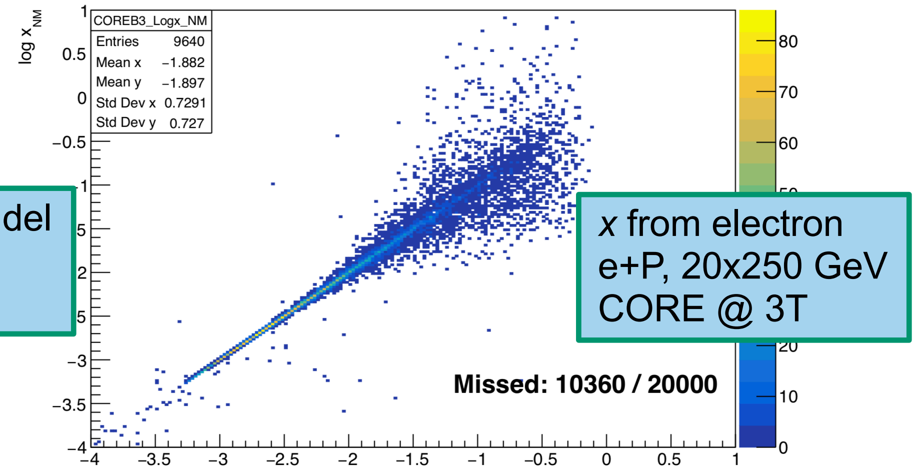


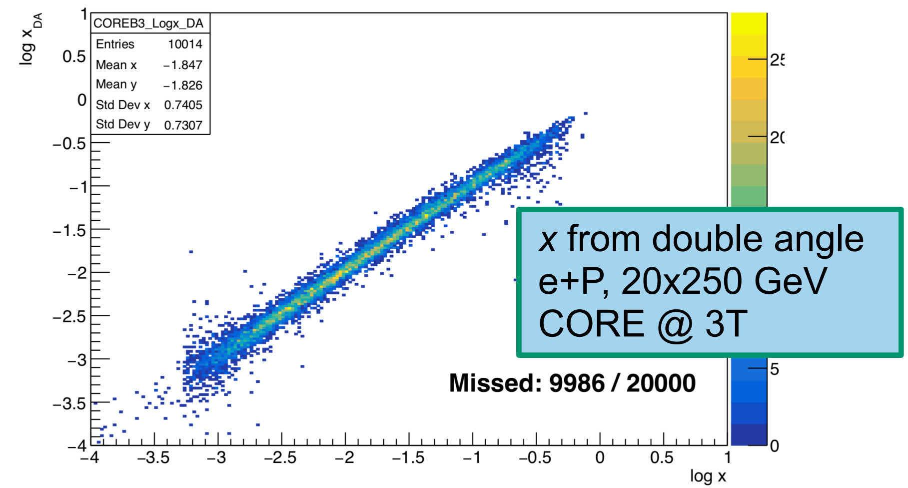


NOT vetted or complete
40 more Pgun plots at
https://github.com/eic/eicsmeardetectors/blob/core_detector/build/qaplots.pgun.COREB3.pdf

Derived Quantities







NOT vetted or complete
50 more pythia6 plots at
https://github.com/eic/eicsmeardetectors/blob/core_detector/build/qaplotsCOREB3.pdf

Limitations

No concept of geometry, no B-Field

- no physically overlapping unrelated neutral and charged depositions
- Devices can and do have internal geometry though

No high-level analysis tools

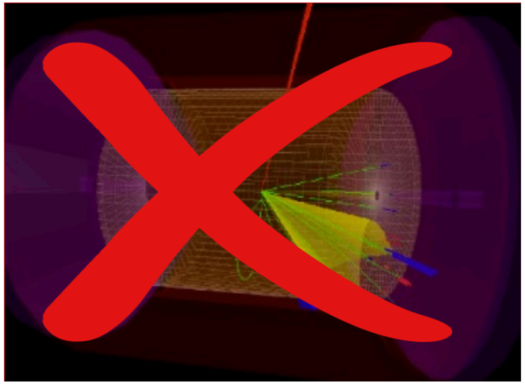
- Fit into existing analysis chain
- Some logic exists for best kinematics calculation but (currently) no weighted mean for example
    - But can DELPHES handle crossing angles etc.?

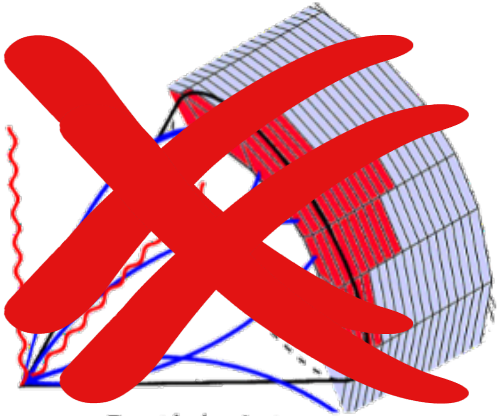
Vertex smearing untested

No decays, material budget, …

No efficiency → can be added

Min. E and p are supported but only at truth level

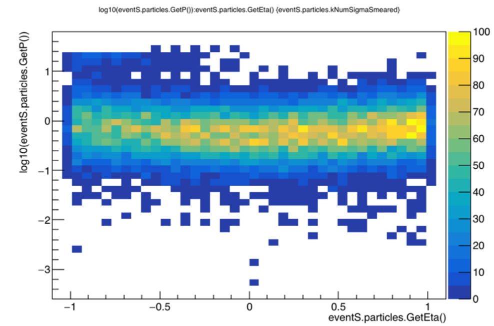




In the Works

NumSigmaPid class, based on https://gitlab.com/preghenella/pid

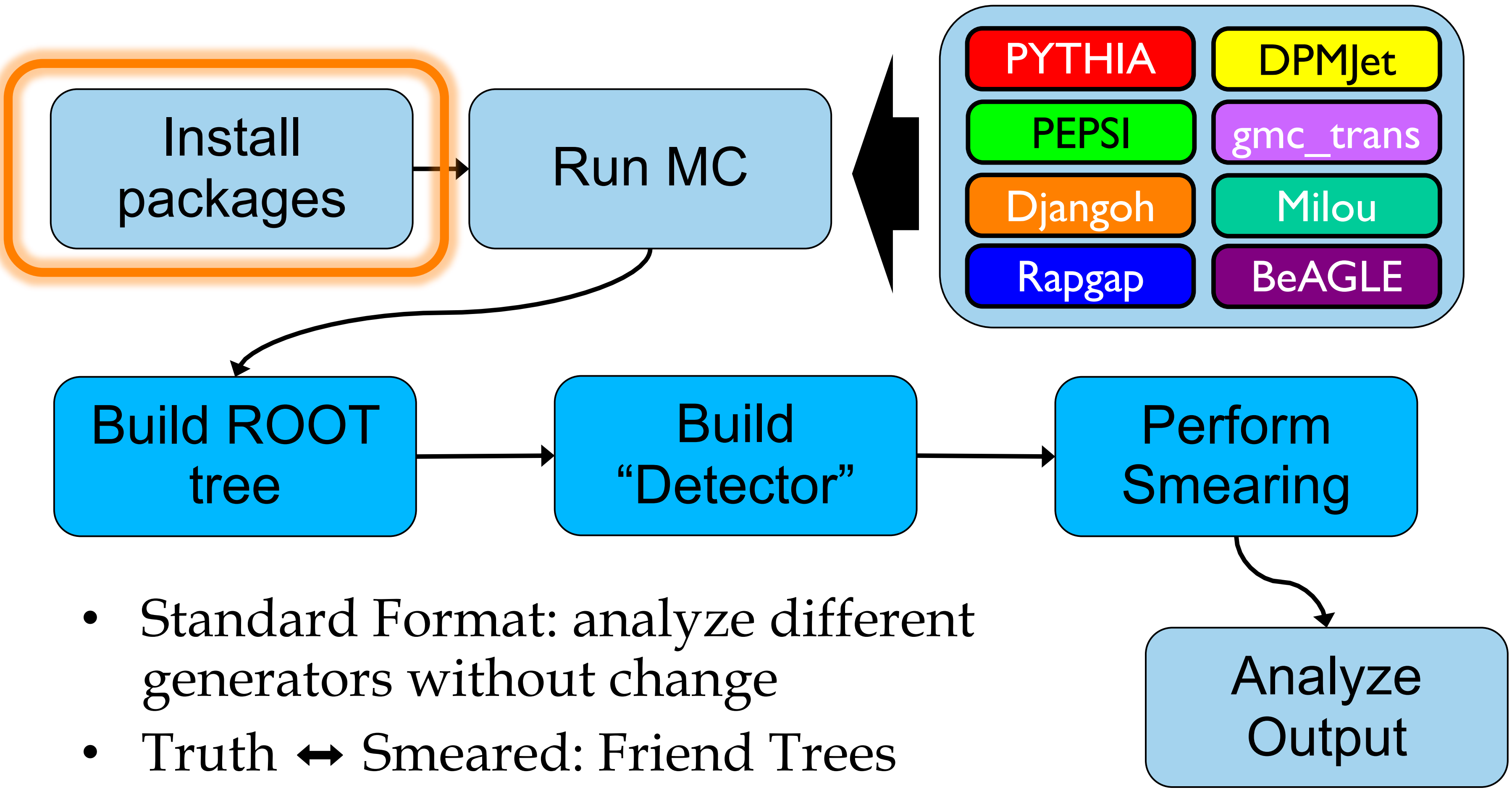
- Not suited (yet) for individual particles
- Not fully uniform
- Need some more agreementon interaction withmomentum smearing



Angular smearing in calorimeters

- Code exists for barrel and endcap, for any level of projectivity
- Needs sign-off / vetting by experts

Workflow



- Standard Format: analyze different generators without change
- Truth ↔ Smeared: Friend Trees

How to Install

Don't

Just use

```
$ setenv EIC_LEVEL dev
$ source /cvmfs/eic.opensciencegrid.org/x8664_sl7/MCEG/releases/etc/eic_cshrc.csh
```

Or the bash version

```
$ export EIC_LEVEL=dev
$ source /cvmfs/eic.opensciencegrid.org/x8664_sl7/MCEG/releases/etc/eic_bash.sh
```

- Works at BNL, JLab, most likely many other places right away
- Otherwise, use singularity as a precursor. fun4all works!
- Contact me for larger scale job submission (works at labs, OSG)
- ESCalate, spack, docker, …
- True install: Easy for eic-smear (especially the detectors), generators will need more packages (cernlib, lhapdf, …)
    - Big Three: Best to use their containers - ask Markus Diefenthaler!

Summary

- **eic-smear is fast, light-weight, extensible**
- No dependencies beyond ROOT
- Meant to be **part of a tool chain**
- First stage unifies a host of EIC-relevant MC output

  Why not just transform Lund to HepMC?

  → On my To-Do list, very relevant for e. g. RIVET

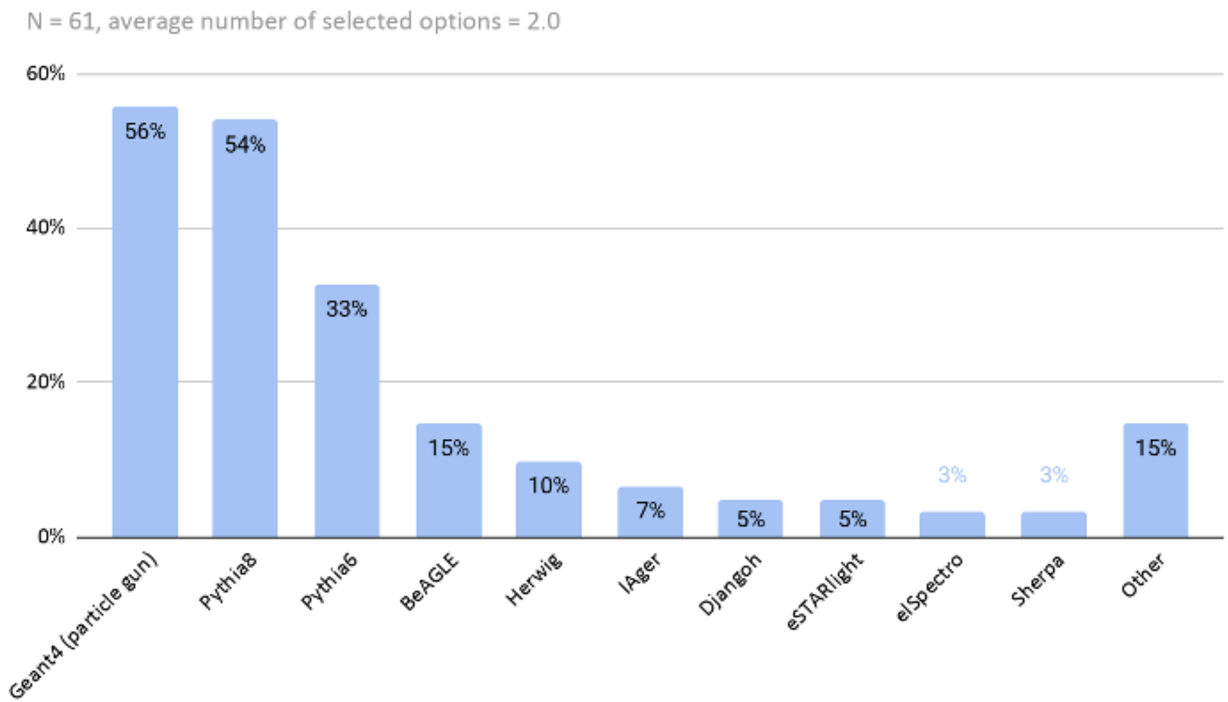
  → But the devil is in the details; needs tuning and checking for every generator

  → For this workshop: Unified ROOT Trees are sufficient

- Cannot replace a full simulation
  – but gives a good estimate of detector effects on observables in <10% of the time it takes to generate PYTHIA6.

BACKUP

MCEGs used for Yellow Report



Other (N = 9): personal computer codes (N = 2), ACT, CLASDIS, ComptonRad, GRAPE-DILEPTON, MADX, MILOU, OPERA, RAYTRACE, Sartre, Topeg, ZGOUBI

From Software Survey

Collaboration on physics event generation

Unique MCEG requirements for EIC Science

- MCEG for polarized ep, ed, and eHe^3
  - including novel QCD phenomena: GPDs, TMDs, etc.
- MCEG for eA

MCEG community MCnet

- focus of last two decades: **LHC**
  - **lesson learned** high-precision QCD measurements require high-precision MCEGs
  - MCEG not about tuning but about physics
- ready to work on ep/eA

How to obtain

Necessary for installation:

- c++ and fortran compilers
- cmake 3.1+
- ROOT (6 is preferred)

Good to have:

- HepMC3, zlib

`gcc 4.8.5 is *just* enough for most things`

Even easier: singularity, fun4all, ESCalate, …
https://eic.github.io/
https://eic.github.io/software/eicsmear_generators_singularity.html

https://eic.github.io/software/eicsmear.html

Generators may need LHAPDF, CERNLIB, potentially FLUKA

Changing or adding to the scripts

- Install your own version, instructions at eicsmeardetectors
- copy or modify – add to `eicsmeardetectors.hh`
  - optional: `BuildByName.cxx` and `cint/LinkDef.h`
- make again!

Important: Add the path which contains your libeicsmeardetectors to the front of LD_LIBRARY_PATH (or DYLD_LIBRARY_PATH on a Mac)

```
bash $ export LD_LIBRARY_PATH=my/path:$LD_LIBRARY_PATH
macbash $ export DYLD_LIBRARY_PATH=my/path:$DYLD_LIBRARY_PATH
tcsh $ setenv LD_LIBRARY_PATH my/path:$LD_LIBRARY_PATH
```

- eic-smear will show that you're doing it right

```
$ eic-smear
/Users/kkauder/software/lib/libeicsmear.dylib
/Users/kkauder/eicdev/eicsmeardetectors/build/libeicsmeardetectors.dylib
```

Resources

- All hosted at github.com/eic (and https://gitlab.com/eic/mceg)
- Use issue tracker for bugs & requests!

- Slack channel: eicug.slack.com/#software-support
- Mailing list: eicug-software@eicug.org, eic-bnl-soft-l@lists.bnl.gov
- Contact: kkauder@bnl.gov

- Online users' guide: https://eic.github.io/software/eicsmear.html
- Class documentation:
  https://eic.github.io/doxygen/
  https://eic.github.io/doxygen/d9/dd8/namespaceSmear.html
  https://eic.github.io/doxygen/db/dfc/namespaceerhic.html

Far forward support

- In **`SmearMatrixDetector_0_1_FF.cxx`**, added (rough) parameterization and acceptance from https://wiki.bnl.gov/eicug/index.php/Yellow_Report_Detector_Forward-IR
- Alex is better suited to speak to the consistency checks ☺, but we can run it right now and look together at the output

```
$ echo 'SmearTree(BuildByName("matrixff",275),
"ep_hiQ2.20x250.small.root")' | eic-smear

$  eic-smear ep_hiQ2.20x250.small.smear.root

eic-smear [] Smeared->Draw("particles.GetEta()",
"particles.IsESmeared() || particles.IsPSmeared()")
```