Kolja Kauder

Center for Frontiers in Nuclear Science

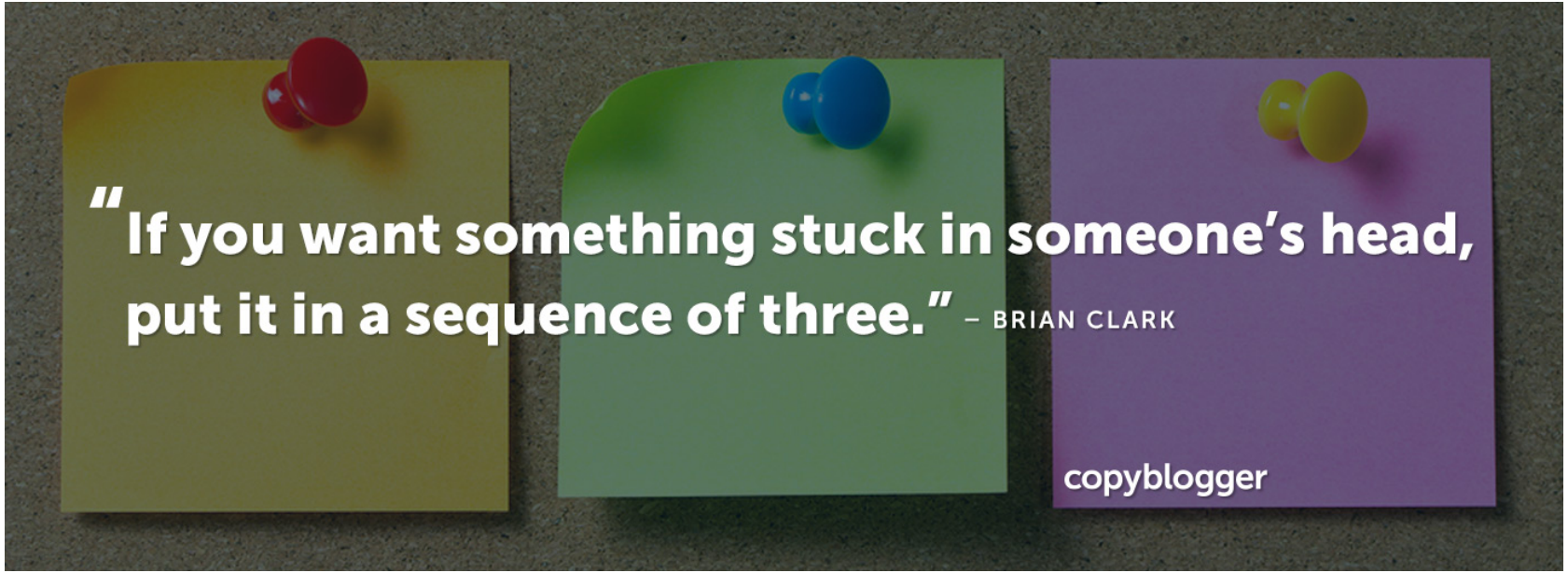
BROOKHAVEN NATIONAL LABORATORY

Fast Simulation with Eic-Smear

Ad-hoc CORE workshop
Online, March 2021

U.S. DEPARTMENT OF ENERGY | Office of Science

Eic-smear – Contents



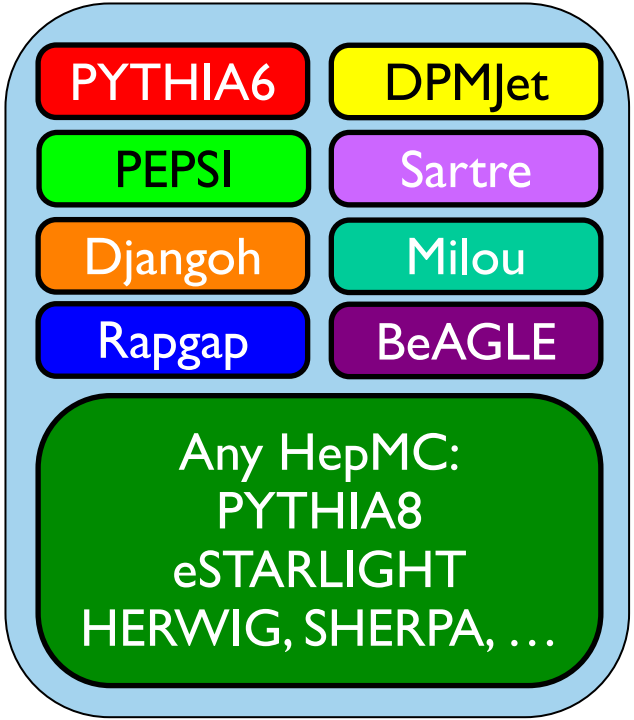
1. Unify EIC MC generator output in ROOT
2. Smear said output quickly
3. Calculate derived quantities

Accepted Generators

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

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

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



Typical generator command:

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

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

Transformation

- Generated 1M PYTHIA6 events, MinBias, $Q^2 = 10 - 100\ GeV^2$
  - ~ 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;
}
```

Common 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

Smearing

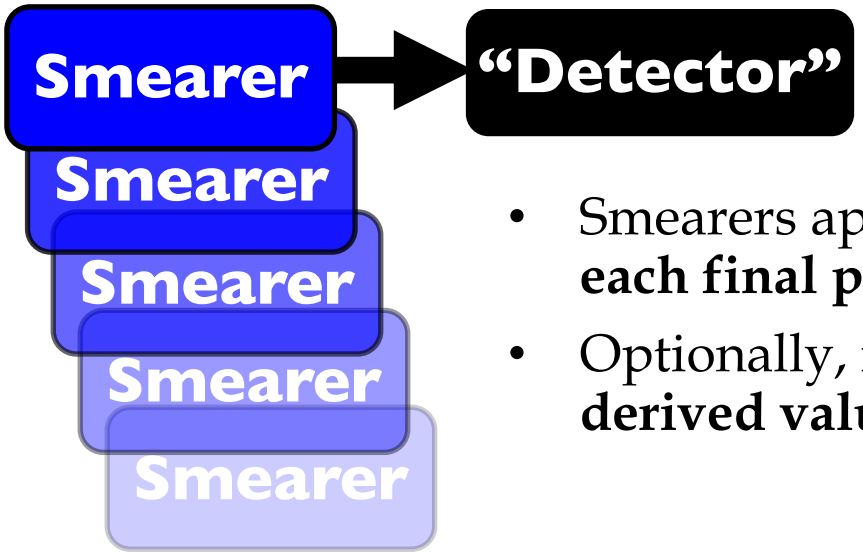
"**Smearer**" defines some element of performance + acceptance

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



- 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

```
26    // Create our tracking capabilities, by a combination of
27    // momentum, theta and phi Devices.
28    // The momentum parametrization (a*p + b) gives sigma_P/P in percent.
29    // So Multiply through by P and divide by 100 to get absolute sigma_P
30    // Theta and Phi parametrizations give absolute sigma in miliradians
31
32    // Track Momentum
33    Smear::Device momentum(Smear::kP, "(P*P*(0.0182031 + ...
34    Smear::Device trackTheta(Smear::kTheta, "((1.0/(1.0*P))*( ...
35    Smear::Device trackPhi(Smear::kPhi, "((1.0/(1.0*P))*(...
36
```

Tracking follows similar TFormula's.

```
224    // Create a detector and add the devices
225    Smear::Detector det;
226    det.AddDevice(emcalBck);
227    det.AddDevice(emcalMidBck);
228    ...
229
230    // The detector will calculate event kinematics from smeared values
231    det.SetEventKinematicsCalculator("NM JB DA");
232    return det;
233 }
```

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
```

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);
```

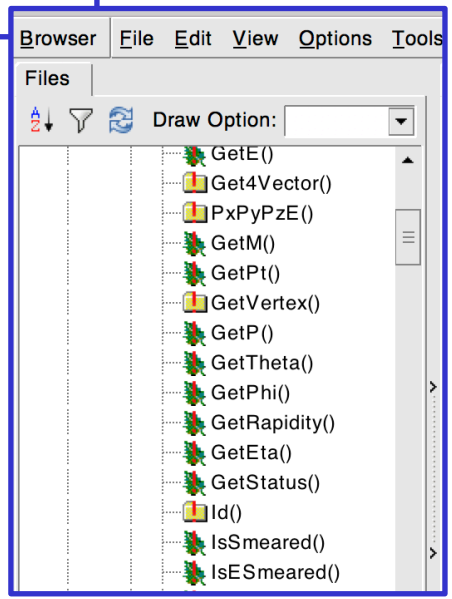
Befriend and get matched branches

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

Particle Loop

- Access properties:

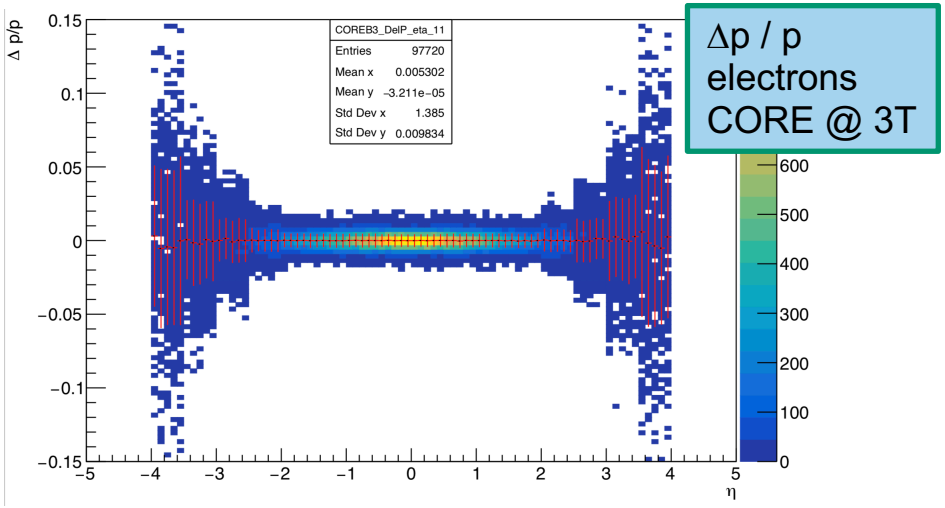
```
if ( inParticleS->IsPSmeared() ){
  auto delP = (inParticle->GetP() - inParticleS->GetP()) / inParticle->GetP();
  coll.DelP_th->Fill(inParticle->GetTheta(), delP);
  coll.DelP_eta->Fill(inParticle->GetEta(), delP);
}
```



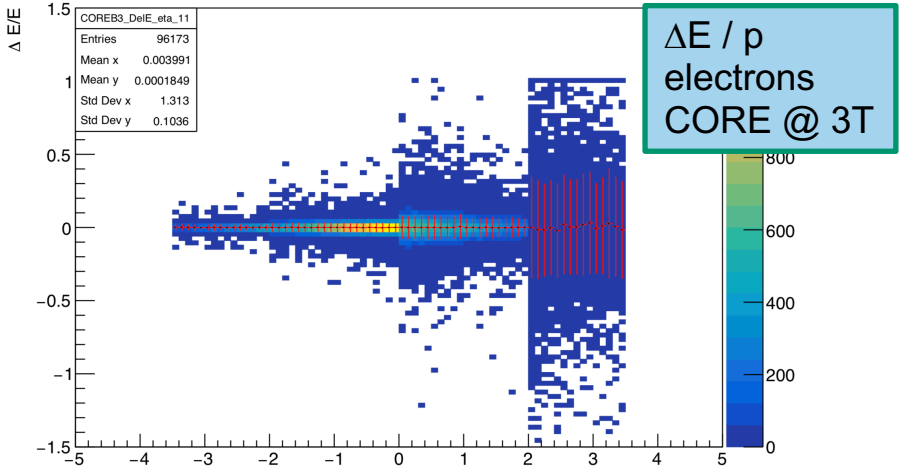
Provided qaplots.cxx demonstrates usage

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

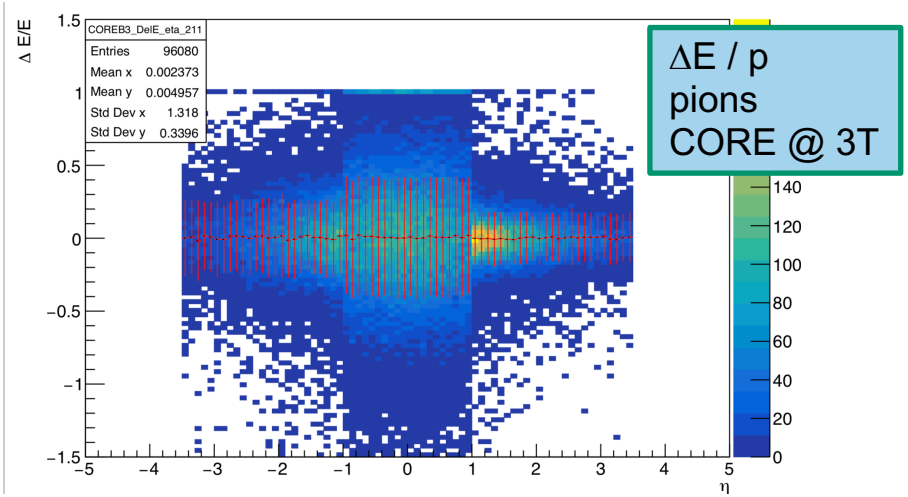
Particle Gun Examples



Δp / p
electrons
CORE @ 3T



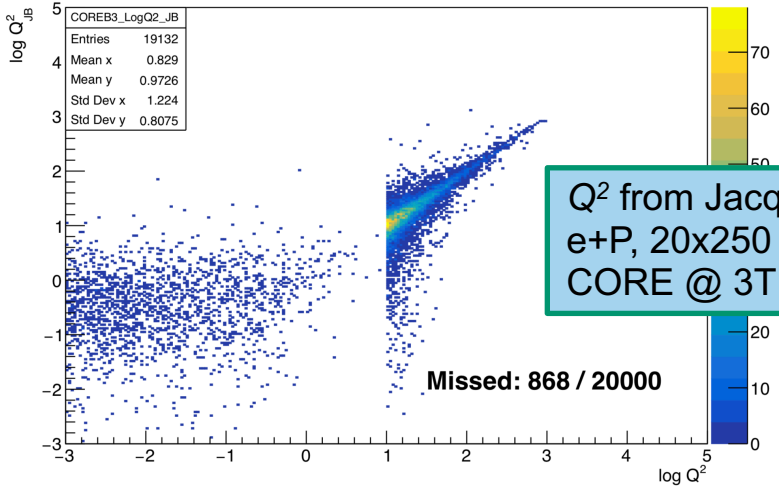
ΔE / p
electrons
CORE @ 3T



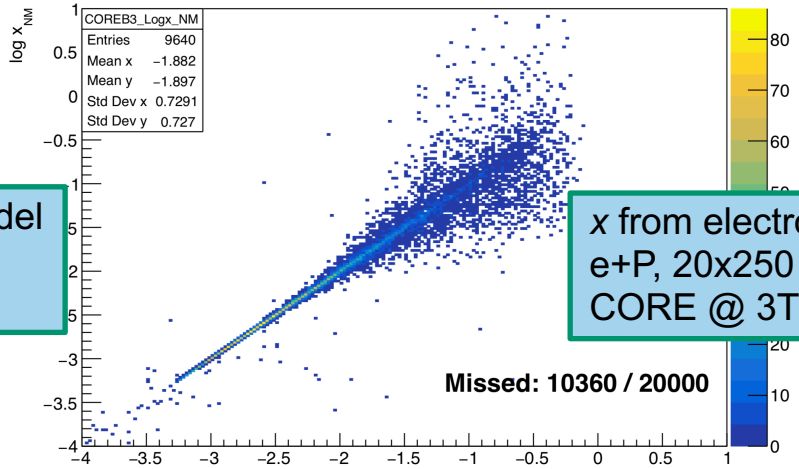
ΔE / p
pions
CORE @ 3T

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

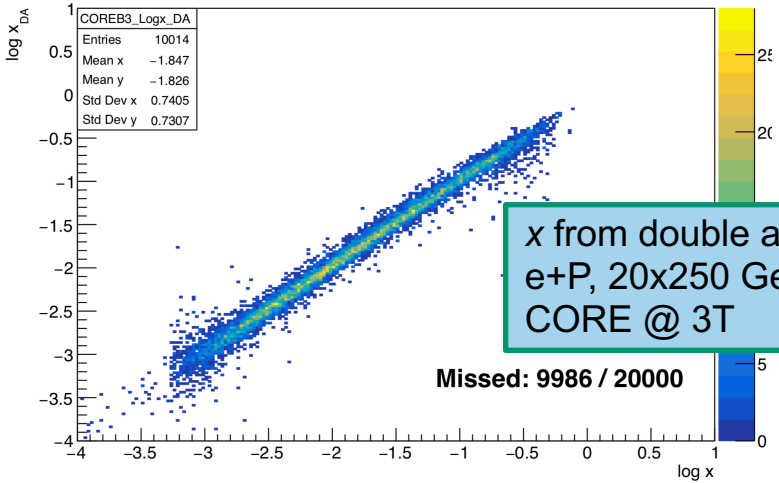
Derived Quantities



Q^2 from Jacquet-Blondel
e+P, 20x250 GeV
CORE @ 3T



x from electron
e+P, 20x250 GeV
CORE @ 3T



x from double angle
e+P, 20x250 GeV
CORE @ 3T

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

Will add numbers from Craig, Oleg, … and ask for "sign-off"

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

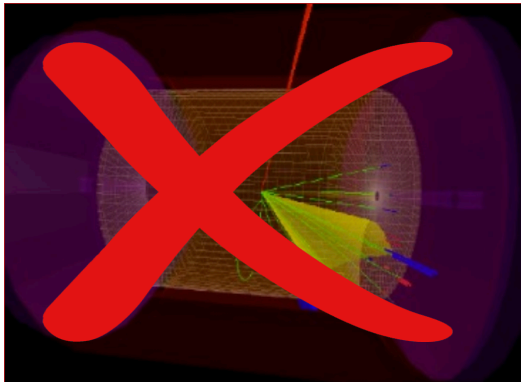
- Fit into existing analysis chain
- Some logic exists for best kinematics calculation but (currently) no weighted mean for example
- But can DELPHES handle crossing angles etc.?

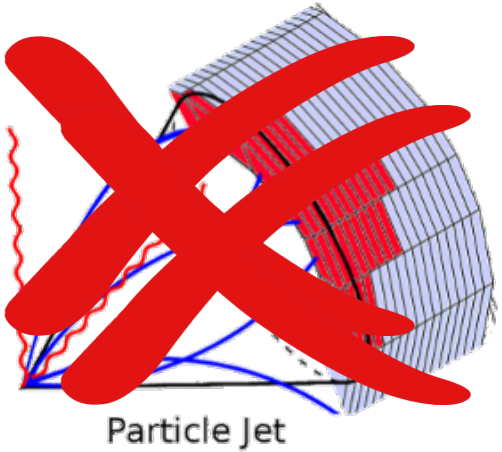
Vertex smearing untested

No decays, material budget, …

No efficiency → can be added

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

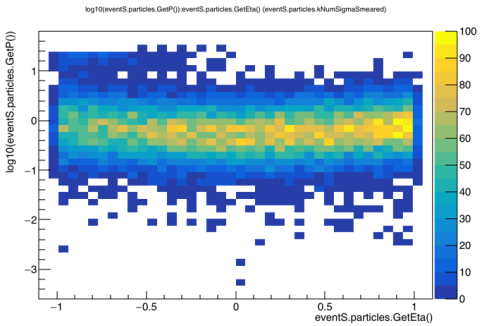




In the Works

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

Summary

- **eic-smear is fast, light-weight, extensible**
- No dependencies beyond ROOT
- First stage unifies a host of EIC-relevant MC output
- Meant to be part of a tool chain (and already is part of many, like fun4all, EicRoot, ESCalate)

- 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.

- For interested parties: eic-smear and fun4all in ECCE this Friday: https://indico.bnl.gov/event/11112/

BACKUP

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
```

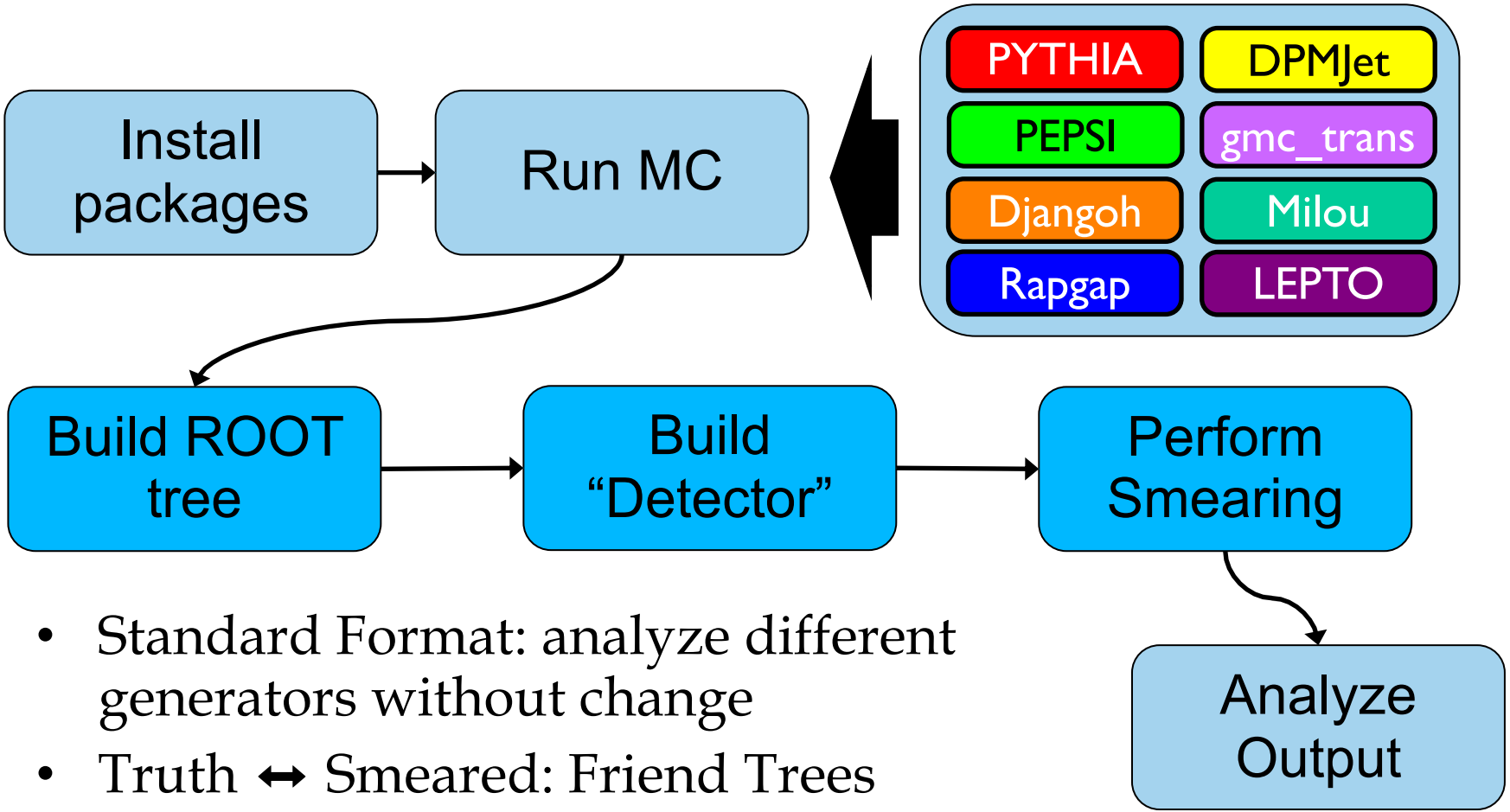
Resources

- 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

Work Flow



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

6. 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()")
```