TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Physik Institut für Kern- und Teilchenphysik

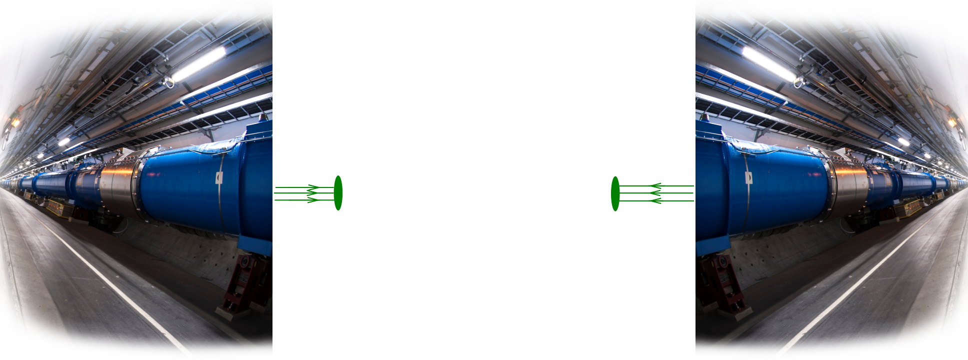
Rivet – a toolkit for theory-data comparisons at high-energy colliders

Introduction and tutorial

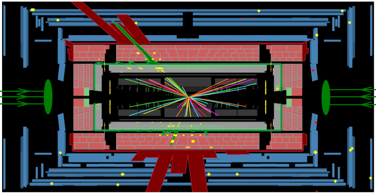
Frank Siegert, TU Dresden

Freiburg Graduate School Seminar, May 2019

Theory-data comparisons at colliders



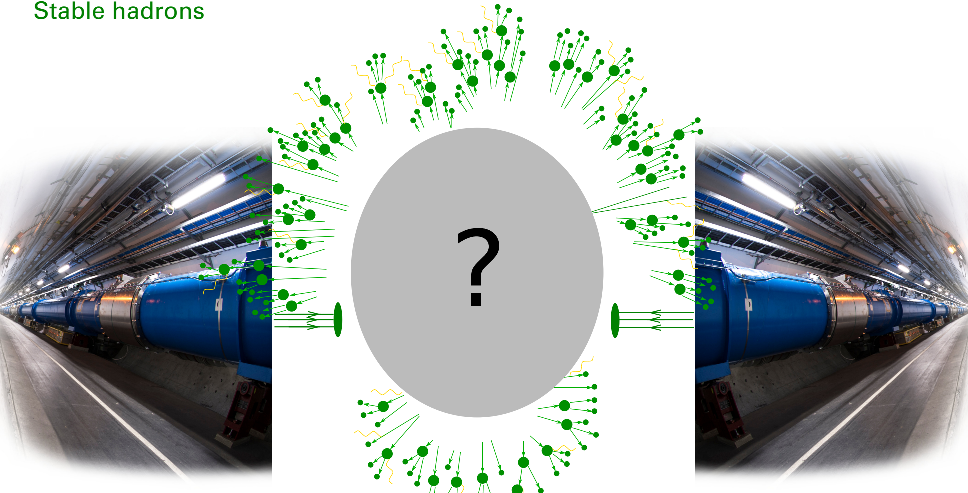
In our detectors:
Energy deposits & tracks



[ATLAS event display from 13 TeV collisions]

In our detectors:
Energy deposits & tracks

Particles interacting with detector:
Stable hadrons

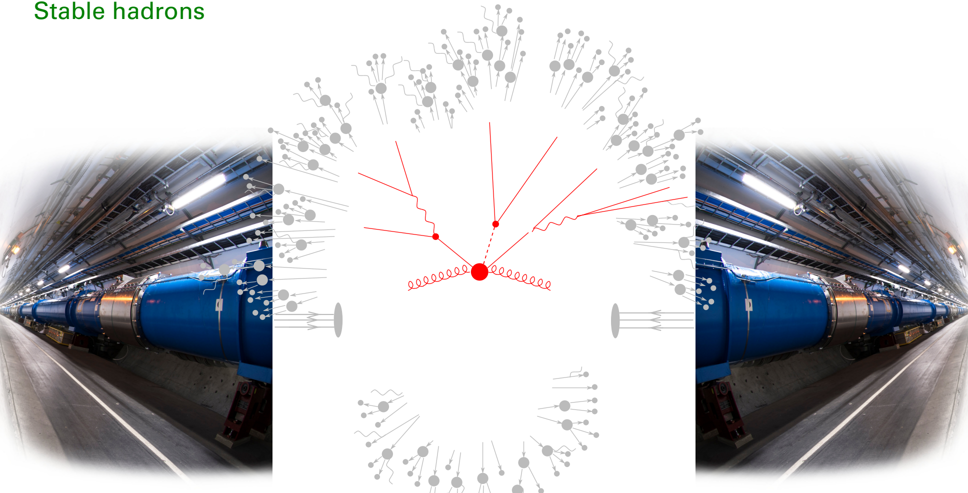


Theory-data comparisons at colliders

In our detectors:
Energy deposits & tracks

Particles interacting with detector:
Stable hadrons

Interesting for our understanding:
Fundamental physics!



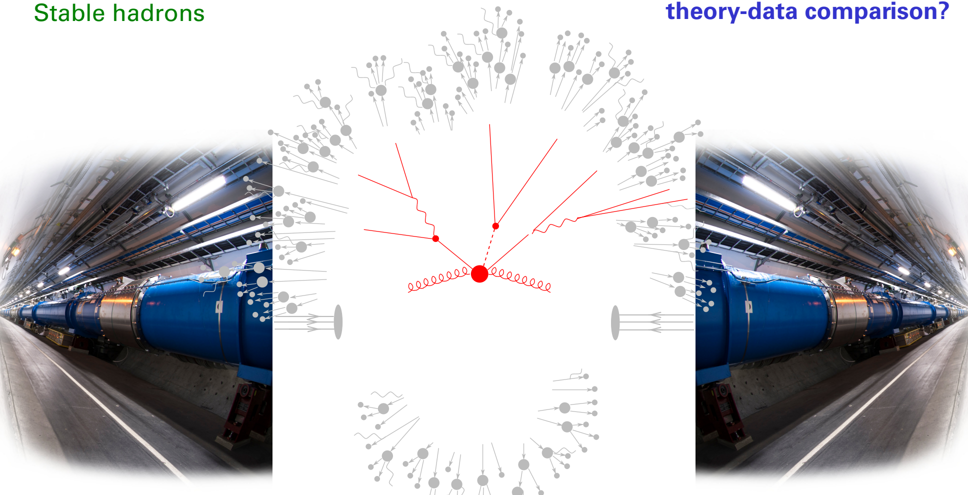
Theory-data comparisons at colliders

In our detectors:
Energy deposits & tracks

Particles interacting with detector:
Stable hadrons

Interesting for our understanding:
Fundamental physics!

What's the best level for theory-data comparison?



Detector (reconstruction) level

Many LHC analyses compare data and theory at detector level:

- Data digitised and reconstructed into high-level calibrated objects: jets, e, μ, τ, γ
- Theory simulated in Monte-Carlo event generators and particles passed through detector simulation + digi + reco

$+$ Straightforward – no thinking needed once detector simulation available

$+$ Robust

$-$ Comparing to multiple theories needs CPU for detector simulation

$-$ Reproduction of analysis needs experimental experts ($\rightarrow$ analysis preservation!)

$-$ Physicists outside experiment cannot repeat comparisons to data, e.g. with new calculations or models

Mainly used in searches for BSM physics.

Parton level

Some analyses correct data to the parton level using MC generators

- Correction for detector effects and non-perturbative effects:
  - (parton shower and QED FSR)
  - hadronisation and hadron decays
  - multiple parton interactions
  - beam remnants and intrinsic $k_\perp$

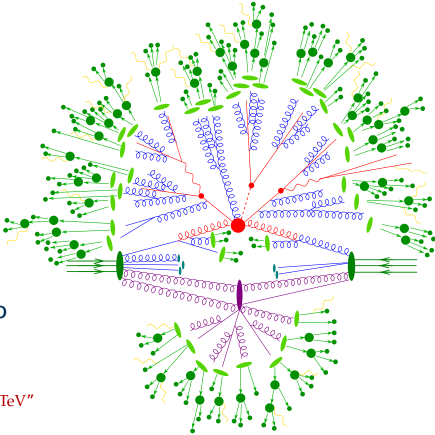
$+$ Simple comparison to parton-level calculations

– Additional effort to determine NP corrections robustly

– MC model dependence transferred to measurement!

"Measurement of the $p_\perp$ spectrum of Pythia v8.235 status-23 top-quarks with the XYZ detector at $\sqrt{s} = 13\,\mathrm{TeV}$"

$\rightarrow$ useless analysis preservation?



Fortunately not used very often for final LHC measurements!

Particle level

Rivet[1] advocates measurements at the $\{$particle$|$hadron$|$truth$\}$ level

- Measurements (as opposed to searches) resemble our final word from the experiment on a given process/observable
  - $\rightarrow$ should be independent from current theory understanding and MC
- Only detector effects removed by unfolding procedure
  - – If done correctly, very model-independent

– Needs robust unfolding procedure during analysis

+ Easy comparison to other (newer) theories and MC modelling outside the original experiment

+ Can typically also be used to compare to parton-level calculations
  - Might need NP corrections for PL calculation, depending on observable
  - Notable exception: heavy-flavour tagged jets – often defined based on B-hadrons

...and tries to make their implementation/sharing simple.

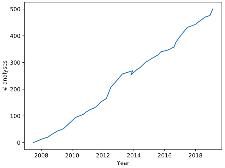
[1](together with basically all of the LHC community)

Rivet

Rivet is an analysis system for MC events + *lots* of analyses

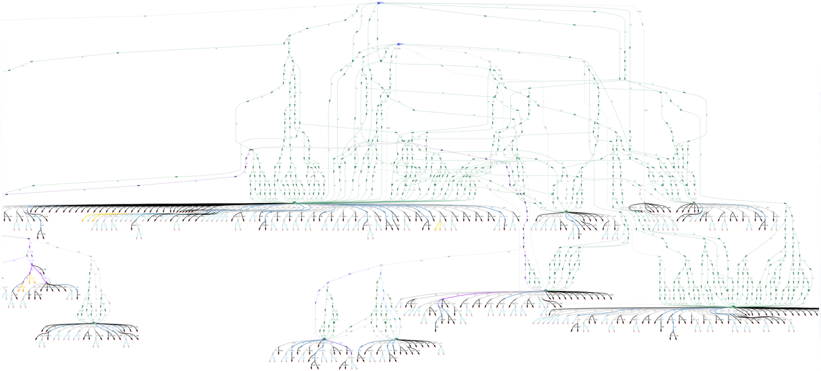
∼ 500 built-in! ∼ 50 are pure MC, and some double-counting

- Easy and powerful way to get physics numbers & plots from *any* MC gen
- LHC standard for preserving data analyses: standard in ATLAS & CMS SM
- Origins in SM, and particularly QCD for MCs – extended for search preservation since v2.5 by adding detector transfer-function features
- C++ library with Python interface, analyses are plugins, code is “clean”
- **“If you can’t write a Rivet analysis for it, it’s probably unphysical”!**



Generator independence

A Pythia8 $t\bar{t}$ event visualised from HepMC output:



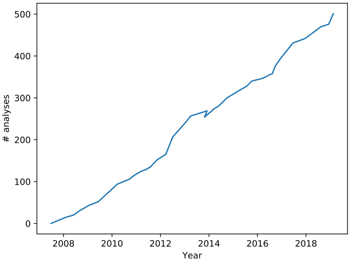
PDF link

Most of this is not standardised: Herwig and Sherpa look *very* different.
But final states and decay chains have to have equivalent meaning.

Analysis coverage / wishlist

Lots of analyses, but we're still missing a lot! You can help…

Semi-automatic Rivet LHC analysis wishlist



Rivet LHC analysis coverage

Rivet analyses exist for 218/827 papers = 26%. 116 priority analyses required.

Total number of CDS papers scanned = 2185, at 2018-06-14

Breakdown by identified experiment (in development):

Key	ALICE	ATLAS	CMS	LHCb	Unknown
Rivet wanted:	226	332	302	82	0
Rivet REALLY wanted:	28	25	53	10	0
Rivet provided:	11 (6%)	136 (44%)	60 (24%)	11 (15%)	0

Show greylist Show blacklist

2622139: Search for a dimuon resonance in the Υ mass region [LHCB]
CDS Inspire

2622094: Search for chargino-neutralino production using recursive jigsaw reconstruction in final states with two or three charged leptons in proton-proton collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector [ATLAS]
CDS Inspire

2621963: Search for pair production of heavy vector-like quarks decaying into high-p_T W bosons and top quarks in the lepton-plus-jets final state in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector [ATLAS]
CDS Inspire HepData

2621727: Search for resonant WZ production in the fully leptonic final state in proton-proton collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector [ATLAS]
CDS Inspire HepData

2621538: Search for pair-produced resonances each decaying into at least four quarks in proton-proton collisions at $\sqrt{s} = 13$ TeV [CMS]
CDS Inspire

2621428: Measurement of the weak mixing angle using the forward-backward asymmetry of Drell-Yan events in pp collisions at 8 TeV [CMS]
CDS Inspire

2621423: Search for narrow and broad dijet resonances in proton-proton collisions at $\sqrt{s} = 13$ TeV and constraints on dark matter mediators and other new particles [CMS]
CDS Inspire

2320693: Search for new phenomena using the invariant mass distribution of same-flavour opposite-sign dilepton pairs in events with missing transverse momentum in $\sqrt{s} = 13$ TeV pp collisions with the ATLAS detector [ATLAS]
CDS Inspire HepData

2320574: p-p, p-Λ and Λ-Λ correlations studied via femtoscopy in pp reactions at $\sqrt{s}$ = 7 TeV [ALICE]
CDS Inspire

First Rivet runs

Command-line interface

`rivet` and other command line tools to query and run routines

- List available analyses:
  `rivet --list-analyses`
- List ATLAS analyses:
  `rivet --list-analyses "ATLAS|CMS"`
- Show some pure-MC analyses' full details:
  `rivet --show-analysis MC_`

Same metadata and API docs online at http://rivet.hepforge.org

All Rivet commands start with `rivet-`, so tab-complete lists them all

Running existing analyses

To avoid huge files, we get the events from generator to Rivet by writing HepMC (from Py8) to a filesystem pipe

```
$ mkfifo fifo.hepmc

$ run-pythia -n 200000 -e 8000 -c Top:all=on -o fifo.hepmc &

$ rivet fifo.hepmc -a MC_TTBAR,MC_JETS,MC_FSPARTICLES
    -a ATLAS_2015_I1404878,CMS_2016_I1473674

$ rivet-mkhtml Rivet.yoda:'Pythia 8 $t\bar{t}$'
```

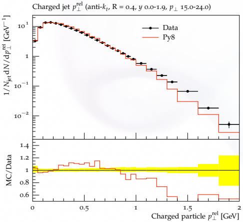
By default *unfinalised* histos are written every 1000 events: monitor progress through the run. Killing with `Ctrl-C` is safe: finalizing is run

Plotting

“YODA” stats library — http://yoda.hepforge.org
Bin-width handling, bin gaps, object ownership, thread-safety ⇒ non-ROOT histogramming

- Separation of stats from presentation: plotting via **`make-plots`** script
- Text-based data format with all second-order stat moments: full stat merging up to all means and variances
- YAML metadata and zipped read/write from v1.7.0
- Being gradually extended to handle more complex physics data types

CLI tools: **`yodals`**, **`yodadiff`**, **`yodamerge`**, **`yodascale`**, **`yoda2root`**, etc.



Writing a first analysis

Writing an analysis

Writing an analysis is of course more involved

But the C++ interface is pretty friendly: most analyses are short, simple, and readable

An example is usually the best instruction: take a look at
https://rivet.hepforge.org/analyses/MC_FSPARTICLES.html

Code is "mostly normal":

- Typical init/exec/finalize loop structure
- Histograms ∼normal; titles, etc. → external `.plot` file
- Particle, Jet and FourMomentum classes with some nice things like `abseta()` and `abspid()`, constituents, decay-chain searching, and compatibility with FastJet objects
- Use of *projections* for auto-cached computations

Projections

Projections are just observable calculators: given an **`Event`** object, they *project* out physical observables.

Automatic caching of results leads to slightly odd calling code:

Declaration with a string name in the **`init`** method:

```
void init() {
  ...
  const SomeProj sp(foo, bar);
  declare(sp, "MySP");
  ...
}
```

Application in the **`analyze`** method via the same name:

```
void analyze(const Event& evt) {
  ...
  const SomeProjBase& mysp = apply<SomeProj>(evt, "MySP");
  mysp.foo()
  ...
}
```

Then query it about the things it has computed, via the object/ref API

Particle finders & final-state projections

Rivet is mildly obsessive about calculating from final state objects

So a *very* important set of projections is those used to extract final state particles, which inherit from `FinalState`

- The `FinalState` projection finds all final state particles in a given η range, with a given p_{T} cutoff.
- Subclasses `ChargedFinalState` and `NeutralFinalState` have the predictable effect!
- `IdentifiedFinalState` can be used to find particular particle species. Nowadays arguably done more nicely via a `Cut`
- `VetoedFinalState` finds particles *other* than specified. Ditto
- `VisibleFinalState` excludes invisible particles like neutrinos, LSP

NB. Most FSPs can take another FSP as a constructor argument and augment it

Using an FSP to get final state particles

```
void init() {
  ...
  const ChargedFinalState cfs(Cuts::pT > 500*MeV && Cuts::abseta < 2.5);
  declare(cfs, "ChFS");
  ...
}
```

```
void analyze(const Event& evt) {
  ...
  const FinalState& cfs = apply<FinalState>(evt, "ChFS");
  MSG_INFO("Total charged mult. = " << cfs.size());
  for (const Particle& p : cfs.particles()) {
    MSG_DEBUG("Particle eta = " << p.eta());
  }
  ...
}
```

More complex projections like `DressedLeptons`, `FastJets`, `WFinder`, `TauFinder` ...implement expt-like strategies for dressing, tagging, mass-windowing, etc.

Selection cuts

Passing ordered lists of doubles to configure "automatic" cut rules is inflexible, illegible, and error-prone. So…

Combinable `Cut` objects:

- `FinalState(Cuts::pT > 0.5*GeV && Cuts::abseta < 2.5)`
- `fs.particles(Cuts::absrap < 3 || (Cuts::absrap > 3.2 && Cuts::absrap < 5), cmpMomByEta)`

Can also use cuts on PID and charge:

- `fs.particlesByPt(Cuts::abspid == PID::ELECTRON)`, or
- `FinalState(Cuts::charge != 0)`

Use of *functions/functors* for ParticleFinder filtering is also possible: very general, especially with C++ *lambdas*

Jets

One more important projection set is those which find *jets*

There's a `JetAlg` abstract interface, but almost always use FastJet, via `FastJets`

Define the input particles (via a `FinalState`), and the jet alg & params:

```
const FinalState fs(-3.2, 3.2);
declare(fs, "FS");
FastJets fj(fs, FastJets::ANTIKT, 0.6,
            JetAlg::ALL_MUONS, JetAlg::ALL_INVISIBLES);
declare(fj, "Jets");
```

Get the jets and loop over them in decreasing p_T order:

```
const Jets jets =
  apply<JetAlg>(evt, "Jets").jetsByPt(20*GeV);
for (const Jet& j : jets) {
  for (const Particle& p : j.particles()) {
    const double dr = deltaR(j, p); //< auto-conversion!
  }
}
```

Remember to `#include "Rivet/Projections/FastJets.hh"`

NB. Lots of handy functions in `Rivet/Math/MathUtils.hh`!

Jet flavour

`FastJets` automatically ghost-tags jets using b and c hadrons (and τ's):

- `if (myjet.bTagged()) ...`
- `if (myjet.bTags().size() > 1) ...`

And you can use `Cuts` to refine the truth tag:

- `myjet.bTagged(Cuts::abseta < 2.5 && Cuts::pT > 5*GeV)`

Jet substructure

Looking inside jets is now common practice.

Rivet doesn't duplicate existing tools: best just to use FastJet directly

```
const PseudoJets psjets = fj.pseudoJets();
const ClusterSequence* cseq = fj.clusterSeq();

Selector sel_3hardest = SelectorNHardest(3);
Filter filter(0.3, sel_3hardest);
for (const PseudoJet& pjet : psjets) {
  PseudoJet fjet = filter(pjet);
  ...
}
```

Note: if using FastJet3 tools, you'll need to add `lifastjettools` to the `rivet-buildplugin` command line. And a `-L/path/to/` arg as well, until the next release. Just compilation, no magic

Rivet's `Jet` and `Particle` classes auto-convert to `PseudoJet`:
$\Rightarrow$ `d23 = cs.exclusive_subdmerge(jetproj.jetsByPt[0], 2)`

Writing, building & running your own analysis

Let's start with a simple "particle analysis", just plotting some simple particle properties like η, p_T, ϕ, etc. Then we'll try jets or W/Z.

To get an analysis template, which you can fill in with an FS projection and a particle loop, run e.g. `rivet-mkanalysis MY_TEST_ANALYSIS` – this will make the required files.

Once you've filled it in, you can either compile directly with `g++`, using the `rivet-config` script as a compile flag helper, or run `rivet-buildplugin MY_TEST_ANALYSIS.cc`

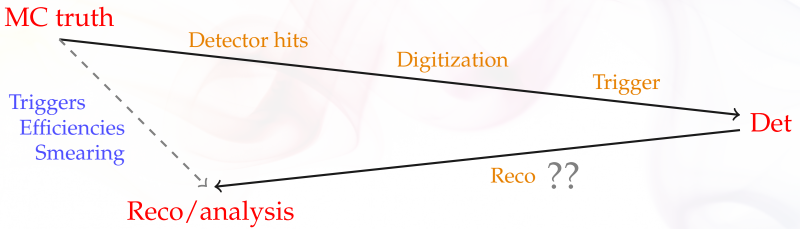
To run, first `export RIVET_ANALYSIS_PATH=$PWD`, then run `rivet` as before…or add the `--pwd` option to the `rivet` command line.

BSM searches and detector effects

Detector effects

Normal in SM, top, etc. measurements to *unfold* detector effects.
Usually "uneconomic" to do that for BSM searches

Explicit fast detector simulation vs. smearing/efficiencies:



- **(Private) reco algorithms already reverse most detector effects**
- Reco calibration to MC truth, so kinematics *usually* subleading
- Efficiency & mis-ID effs dominate – tabulated in all fast-sims
- ⇒ flexible parametrisation: effs change with analysis phase-space, experiment reco-code version, collider run, . . .

and *need to guarantee stability for preservation*

Using Rivet's fast-sim tools

Smearing is provided as "wrapper projections" on normal particle, jet, and MET finders.

Smearing configuration via efficiency/modifier functions.

To use, first `#include "Rivet/Projections/Smearing.hh"`

Examples:

```
FinalState es1(Cuts::abseta < 5 && Cuts::abspid == PID::ELECTRON);
SmearedParticles es2(es, ELECTRON_EFF_ATLAS_RUN2, ELECTRON_SMEAR_ATLAS_RUN2);
declare(es2, "Electrons");

FastJets js1(FastJets::ANTIKT, 0.6, JetAlg::DECAY_MUONS);
SmearedJets js2(fj, JET_SMEAR_ATLAS_RUN2, JET_EFF_BTAG_ATLAS_RUN2);
declare(js2, "Jets");

...

Particles elecs = apply<ParticleFinder>(event, "Electrons").particles(10*GeV);
Jets jets = apply<JetAlg>(event, "Jets").jetsByPt(30*GeV);
```

Standard global functions here, but private fns or inline lambdas better when possible

Selection tools for search analyses

Search analyses typically do a lot more “object filtering” than measurements. Lots of tools to express complex logic neatly:

- Filtering functions: `filter_select(const Particles/Jets&, FN)`, `filter_discard(...)` + `ifilter_*` in-place variants
- *Functors* for common “stateful” filtering criteria: `PtGtr(10*GeV)`, `EtaLess(5)`, `AbsEtaGtr(2.5)`, `DeltaRGtr(mom, 0.4)`, `ParticleEffFilter(FN), ...`
  - Lots of these in `Rivet/Tools/ParticleBaseUtils.hh`, `Rivet/Tools/ParticleUtils.hh`, and `Rivet/Tools/JetUtils.hh`
- `any()`, `all()`, `none()`, etc. – accepting functions/functors
- Cut-flow monitor via `#include "Rivet/Tools/Cutflow.hh"`

Let's try it out… preparations

On lxplus

```
source /cvmfs/sft.cern.ch/lcg/releases/LCG_88/gcc/6.2.0/x86_64-centos7/setup.sh
source /cvmfs/sft.cern.ch/lcg/releases/LCG_88/Python/2.7.13/x86_64-centos7-gcc62-opt/Python-env.sh
source /cvmfs/sft.cern.ch/lcg/releases/LCG_88/MCGenerators/rivet/2.7.2/x86_64-centos7-gcc62-opt/rivetenv.sh
```

… or using docker image

- Follow instructions in `https://rivet.hepforge.org/trac/wiki/Docker`
- Use Rivet version `X.Y.Z = 2.7.2`

… or installing locally

- Install locally using bootstrap script as described in `https://rivet.hepforge.org/trac/wiki/GettingStarted`

Get some event files for testing

```
wget http://www.hepforge.org/archive/rivet/LHC-Zee-LOPS.hepmc.gz
wget http://www.hepforge.org/archive/rivet/LHC-Zee-MEPS1.hepmc.gz
```

Let's start simple and analyse the number of particles with $p_T > 100$ MeV and $|\eta| < 5$.

(loosely following https://rivet.hepforge.org/trac/wiki/WritingAnAnalysis)

Create skeleton

```
rivet-mkanalysis MY_TEST_ANALYSIS
```

Boilerplate

```
class MY_TEST_ANALYSIS : public Analysis {
public:

  DEFAULT_RIVET_ANALYSIS_CTOR(MY_TEST_ANALYSIS);

  Histo1DPtr _h_nparticles;

  [...]
};

// The hook for the plugin system
DECLARE_RIVET_PLUGIN(MY_TEST_ANALYSIS);
```

Initialisation

```
void init() {

  // Initialise and register projections
  declare(FinalState(Cuts::abseta < 5 && Cuts::pT > 100*MeV), "FS");

  // Book histograms
  _h_nparticles = bookHisto1D("nparticles", 60, 0.0, 600.0);
}
```

Per-event analysis

```
void analyze(const Event& event) {

  // Apply projections to event
  FinalState fs = apply<FinalState>(event, "FS");
  Particles particles = fs.particles();

  // Fill histograms
  _h_nparticles->fill(particles.size(), event.weight());
}
```

Finalisation of histograms

```
void finalize() {
  scale(_h_nparticles, crossSection()/picobarn/sumOfWeights());
}
```

Compile!

```
$ rivet-buildplugin  MY_TEST_ANALYSIS.cc
```

Run!

```
$ rivet --pwd -a MY_TEST_ANALYSIS -H Rivet-ZeeLOPS.yoda LHC-Zee-LOPS.hepmc.gz
```

Plot!

```
$ rivet-mkhtml  Rivet-ZeeLOPS.yoda
$ firefox  rivet-plots/index.html
```

Let’s also look at the pair of leptons, which we expect in our DY events:

Boilerplate

```
Histo1DPtr _h_nparticles, _h_zmass, _h_zpt;
```

Initialisation

```
void init() {

  // Initialise and register projections
  declare(FinalState(Cuts::abseta < 5 && Cuts::pT > 100*MeV), "FS");
  FinalState fs;
  declare(DressedLeptons(fs, 0.1, Cuts::pT > 20*GeV && Cuts::abseta < 2.5),
                              "Leptons");

  // Book histograms
  _h_nparticles = bookHisto1D("nparticles", 60, 0.0, 600.0);
  _h_zmass = bookHisto1D("zmass", 60, 60.0, 120.0);
  _h_zpt = bookHisto1D("zpt", 100, 0.0, 200.0);
}
```

Per-event analysis

```
void analyze(const Event& event) {

  FinalState fs = apply<FinalState>(event, "FS");
  Particles particles = fs.particles();
  _h_nparticles->fill(particles.size(), event.weight());

  Particles leptons = apply<FinalState>(event, "Leptons").particles();
  if (leptons.size()==2) {
    _h_zmass->fill(
      (leptons[0].momentum()+leptons[1].momentum()).mass(), event.weight());
    _h_zpt->fill(
      (leptons[0].momentum()+leptons[1].momentum()).pT(), event.weight());
  }
}
```

Finalisation of histograms

```
void finalize() {

  scale(_h_nparticles, crossSection()/picobarn/sumOfWeights());
  scale(_h_zmass, crossSection()/picobarn/sumOfWeights());
  scale(_h_zpt, crossSection()/picobarn/sumOfWeights());

}
```

Conclusions

- Particle level analyses are the most future-proof form of preserving measurements
- Rivet provides a framework for particle-level analyses in elegant way
- Huge library of implementations of analyses from LEP, Tevatron, LHC, HERA, RHIC, SPS, . . .
  - ATLAS analysers: Need your help to keep this as complete as possible!

Outlook: Didn't cover aspects relevant for some of you:

- ATLAS analysers: `Rivet_i` interface in Athena
- BSM aficionados: Triggers/efficiencies/smearing for reco-level objects
- N(N)LO calculators: dealing with correlated events (e.g. subtraction)
- Dark Matter searchers: Sorry, probably not very useful for your work.

Questions?