L1JEC explainer

Robin Aggleton (robin.aggleton@cern.ch)

Contents

Note that although you will be dealing with Stage 2 calibrations, I also include references to GCT & Stage 1(2015) trigger systems. This is because sometimes people throw around old lingo, and it's useful to know how the system evolved.

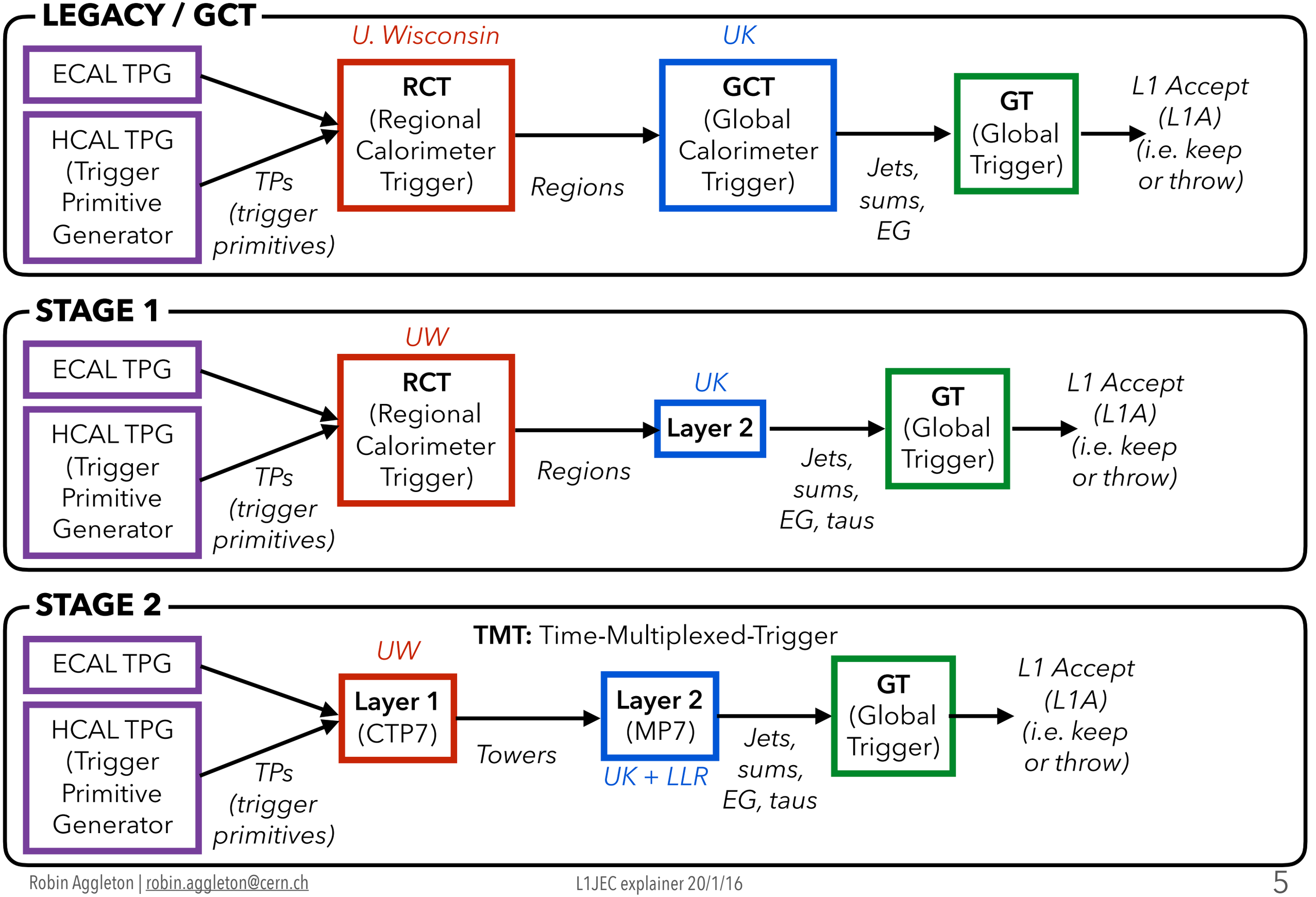
Overview

Trigger System Overview

- Historically, there have been 3 different L1 calo trigger systems in use:
  - "Legacy" / "GCT": used during Run 1, and start of Run 2
  - "Stage 1" / "interim" trigger: used for the latter part of 2015. Designed as a halfway-house between legacy and …
  - "Stage 2" / "TDR" / "2016" trigger: used for 2016 (& beyond)

- You will probably only ever touch Stage 2. But I've included brief diagrams about the other 2 so that you can understand the lingo.
- These are schematic, and massively simplified, but gives you an idea of the steps involved in making jets.

Trigger System Overview (massively simplified)



Overview

Based on:
IN2013-006
AN2015/199

- We are trying to correct the L1 jets for various losses
  - These affect the turnoff curves, rate, and efficiency
- These losses are both p_T and η dependent
- We want the jet trigger **performance to be uniform across the detector**
- We need some 'ideal' or 'reference' jet to compare a given L1 jet against
  - GenJets for MC, PF jets (with offline JEC) for data
  - Sometimes referred to as '**offline**' objects, whereas L1 is '**online**'
- Our metric is **response** $\equiv$ $\mathbf{p_T^{L1} / p_T^{Ref}}$, where p_T^{Ref} refers to the reference jet (e.g. GenJet for MC, PF jet for data)
  - Another potentially interesting quantity is Δp_T.
- This is calculated on a **jet-by-jet basis**

Overview

- To do this we need to **match L1 jets against reference jets**, in a given event (i.e. which L1 jet corresponds to which ‘reference’ jet)
- We use $\Delta R = \sqrt{(\Delta\eta^2 + \Delta\phi^2)}$ as the matching criterium.
  - For more details see later on
- Once we have our matched pairs of jets, we can perform various studies with them
  - Measure the response
  - Measure the resolution (position & energy)
  - …

Overview

- To correct our jets, we want a 'correction factor' that depends on both jet p_T and η. Let us call it $f(p_T, \eta)$, where p_T and η refer to the L1 jet quantities.
- Then each L1 jet gets its pT corrected such that it ends up with p_T^{Corr}:
  - $p_T^{Corr} = p_T \times f(p_T, \eta)$
- **Your goal: figure out $f(p_T, \eta)$**

Overview

- We could just calculate response in many p_T, η bins and use them individually BUT this would be highly subject to fluctuations
- Instead we use the same method JetMET use for their LXXX corrections
- We divide up our detector into η bins, and use a smooth function instead for each η bin. This has the advantages:
  - Removes bin-by-bin fluctuations
  - Allows us to calculate *f* for any jet p_T
- The function parameters are determined by fitting to the data or MC
- Once we have the functions, we can test them in the emulator to ensure it actually does make things better
  - Also important for testing you haven't mucked up anything!

Overview

- Quick aside: In hardware (HW), these are actually applied differently. This is because it is quicker to use a lookup table (LUT) than multiply things
- We use the function to figure out all possible corrected values, since the uncorrected values are quantised
- Note that this was the case for GCT & Stage 1, but probably not for Stage 2

Overview

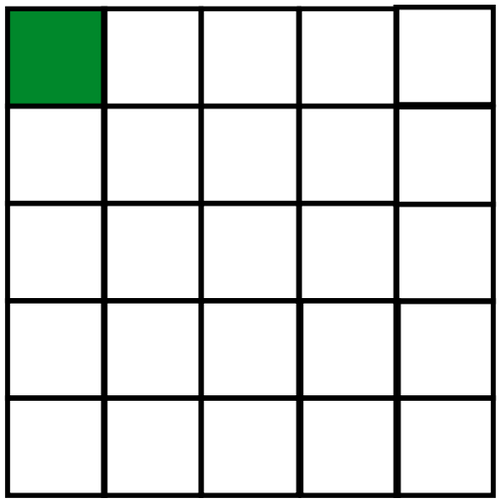
- Note that L1JEC are at the end of a very long chain:
  - ECAL/HCAL Trigger primitives (TPs) → Regions (RCT/Stage1) / Towers (Layer 1) → Jet finding (GCT/Layer 2) → L1JEC → Global Trigger (GT)
- Each preceding step has its own challenges, and possible calibrations (e.g. towers will have calibrations, TPs get updated…)
- If any upstream calibrations change, you will have to check yours are unaffected, and possibly reserve
- This means that if anything looks dodgy with the jets, you are often the first port of call (rightly or wrongly💩)
  - e.g. during low PU runs in 2015, a very large rate was observed when the jet seed threshold was reduced. Everyone screamed “L1JEC”, when intact it was discovered at the end of 2015 it was extra noise further upstream.

Jet details

Some things you should know about jets

- In an idealish world, we would be able to reconstruct all particles in an event, then **cluster** them together into jets via some algorithm (e.g. anti-kT)
  - This is what PF jets are - requires info from tracker, ECAL, HCAL, muons
- The L1 trigger doesn't have all that information, or time.
- Instead we use calorimeter information ("deposits") only
- The ECAL and HCAL have certain 'groupings' or granularity
- (See also https://twiki.cern.ch/twiki/bin/viewauth/CMS/RCTMap)

Some things you should know about jets



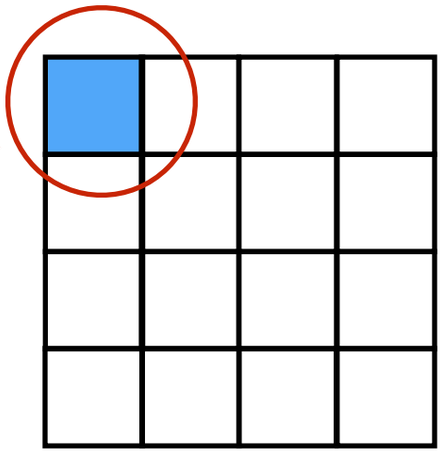
1 tower

= 5 x 5 ECAL crystals

= 1 HCAL tower

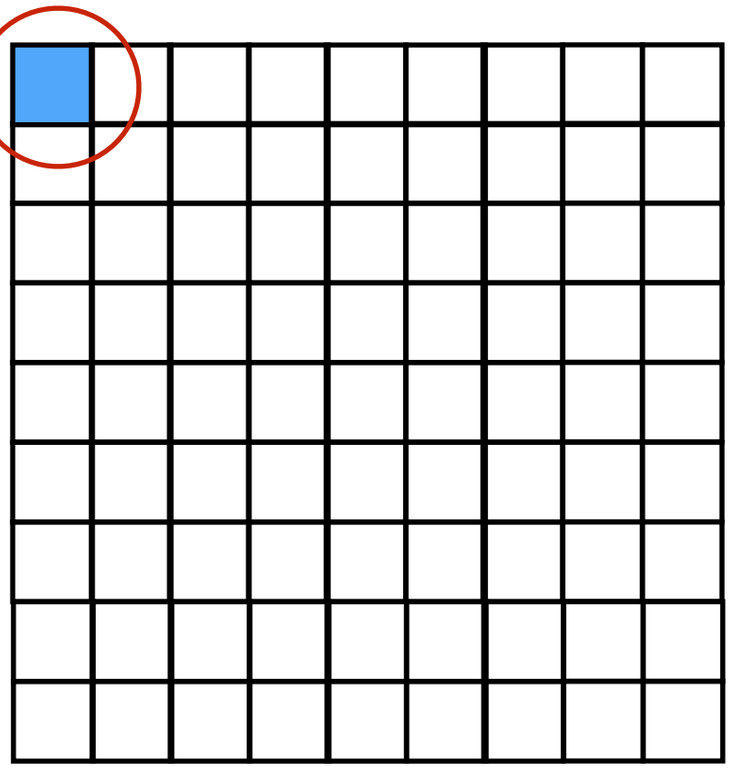
($\Delta\eta \times \Delta\phi = 0.087 \times 0.087$)

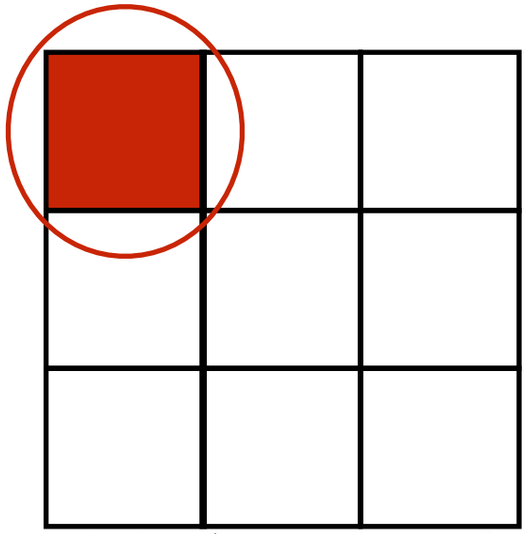
NB $\Delta\eta$ only valid up to $\eta \sim 2.1$ - gets larger at higher η



1 region = 4 x 4 towers

($\Delta\eta \times \Delta\phi = 0.348 \times 0.348$)





1 GCT/Stage 1 jet

= 3 x 3 regions

= 12 x 12 towers

($\Delta\eta \times \Delta\phi = 1.044 \times 1.044$)

1 Stage 2 jet

= 9 x 9 towers

($\Delta\eta \times \Delta\phi = 0.783 \times 0.783$)

Some things you should know about jets

- In hardware, the jet energies are represent by N number of bits
- To convert the bits to a physical energy, we need some conversion mechanism
- L1 jets use a linear scale, with a least-significant-bit (**LSB**) set to 0.5 GeV
  - So HW pT = 7 corresponds to a physical jet pT = 7 x 0.5 = 3.5 GeV
- This means that **all L1 jet energies are quantised in 0.5 GeV steps**
  - 0, 0.5, 1, 1.5, …, 0.5 x (2^N - 1)
- This is relatively nice - there shouldn't be any real issues arriving from quantisation
  - Not necessarily true with 4 GeV quantisation!
- Some studies/thoughts: https://github.com/raggleton/L1JetEnergyCorrections/blob/master/notebooks/CalibEmu.ipynb

Matching details

Done in RunMatcher[Stage2|Data]

Some thing you should know about matching

- The point of matching is to figure out which L1 jet corresponds to which reference jet
- We do this using ΔR between 2 jet collections:
  - L1 jets: no cuts on pT
  - Ref jets: typically p_T > 10 GeV to be fairly inclusive
- Some algorithm subtleties:
  - We start out by going through L1 jet (in descending pT) and look for all ref jets with ΔR < 0.4
  - If there are multiple matches we use the one closest in ΔR.
  - If we have a match, then we move onto the next L1 jet and remove that reference jet from the collection of possible matches so it can't be matched to 2 different L1 jets.
- Changing the order, or pT cut on ref jets, can change things.

Some thing you should know about matching

- For Stage 2, we require a match to be within $\Delta R < 0.4$
  - For GCT/ Stage 1 it was < 0.7 due to worse position resolution
- This is quite a large window, but means we should be relatively unbiased
- Unfortunately, PU creeps in at larger ΔR, so it's a compromise
- One could try and make ΔR pT dependent, but maybe too much faff?

- Are there better ways of matching? Or is this good enough?
  - The main issue is **information**: L1 jets don't have much, only pT, η, ϕ
  - We can't use p_T in matching, since that would bias our calibrations
  - But are there other interesting things? e.g. PU energy?

Some thing you should know about matching

- There is an important distinction between **matching to derive calibrations** and **matching to measure overall performance**
- In the former, we want a sample of good jets that deposit their energy in the calorimeter
  - e.g. jets that have a large pT muon will bias things badly - look very low energy in calorimeter but may look large in reference jet.
- In the latter, we want to understand all the jets that are coming out of our trigger, so we should be more inclusive if necessary
- You should think about what question you are trying to answer!

Some thing you should know about matching

- There are also some subtleties about matching against GenJets:
- GenJets use generator-level particles, which have no PU
- They also aren’t fully propagated in the magnetic field, so their η is at the Primary Vertex (PV)
- Whereas L1 jets are measured at the calorimeter
- If the particles are energetic enough, this is fine - they won’t bend much
- But if they are low energy, they can bend more and land outside the jet region on the calorimeter
- And if really low energy, they might never hit the calorimeter!

Correction details

Done in runCalibration.py

Lots of things you should know about corrections

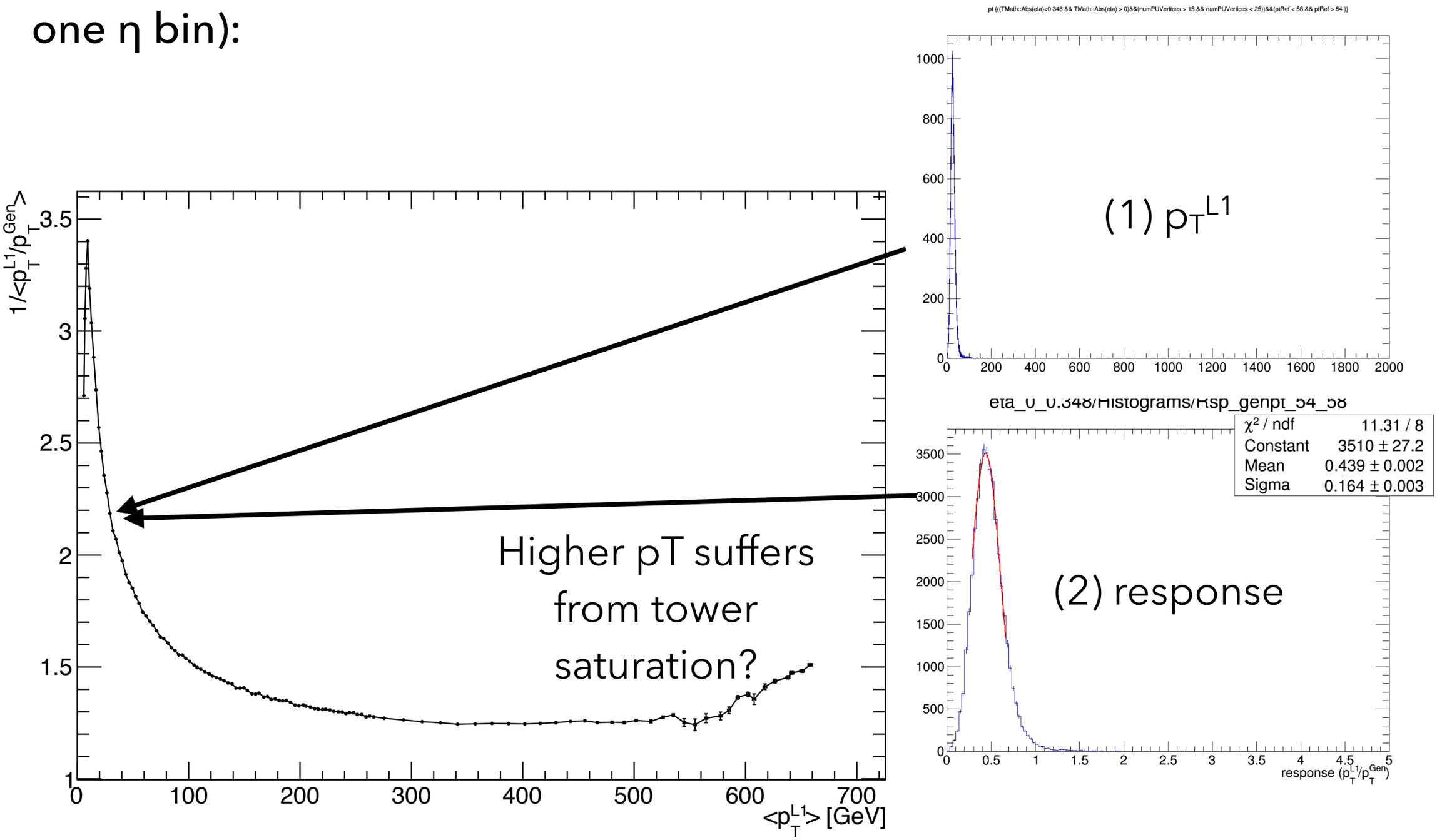
- Reminder: find $f(p_T, \eta)$
- In reality, we can separate out the η dependence by dividing our detector into bins of |η|, and find $f(p_T)$ for each.
- We split up |η| based on regions (for historical reasons), so there are 11 |η| bins currently (TODO use upgraded HF)
- We could potentially do it for each tower (of which there are 40 per ±η):
  - But variation over η isn't that fast-changing
  - It would be a bit of a nightmare - go from 11 to 40 calibration curves, each of which you would have to check though (& component plots +fits) for irregularities. Go from 512 component plots * 11 η bins → 512 * 40 ~ 20K plots
  - Not obvious it would offer much improvement

Lots of things you should know about corrections

- Let's focus on one η bin
- We divide up the possible values of p_T^{Ref} into bins, 4 GeV wide
  - binning here is also historic, and interplay between stats & capturing function change accurately
- For each bin, we make 2 histograms using out matched pairs of jets that fall into the p_T^{Ref} bin:
  1. p_T^{L1}
  2. response
- From (1) we take the raw mean, this gives us the mean p_T^{L1} for this p_T^{Ref} bin, **$\langle p_T^{L1} \rangle$**
- For (2) we fit the peak with a Gaussian, and then take the mean of that Gaussian. This is our mean response in this p_T^{Ref} bin, **<rsp>.** So the **correction factor in this bin will be 1/<rsp>**

Lots of things you should know about corrections

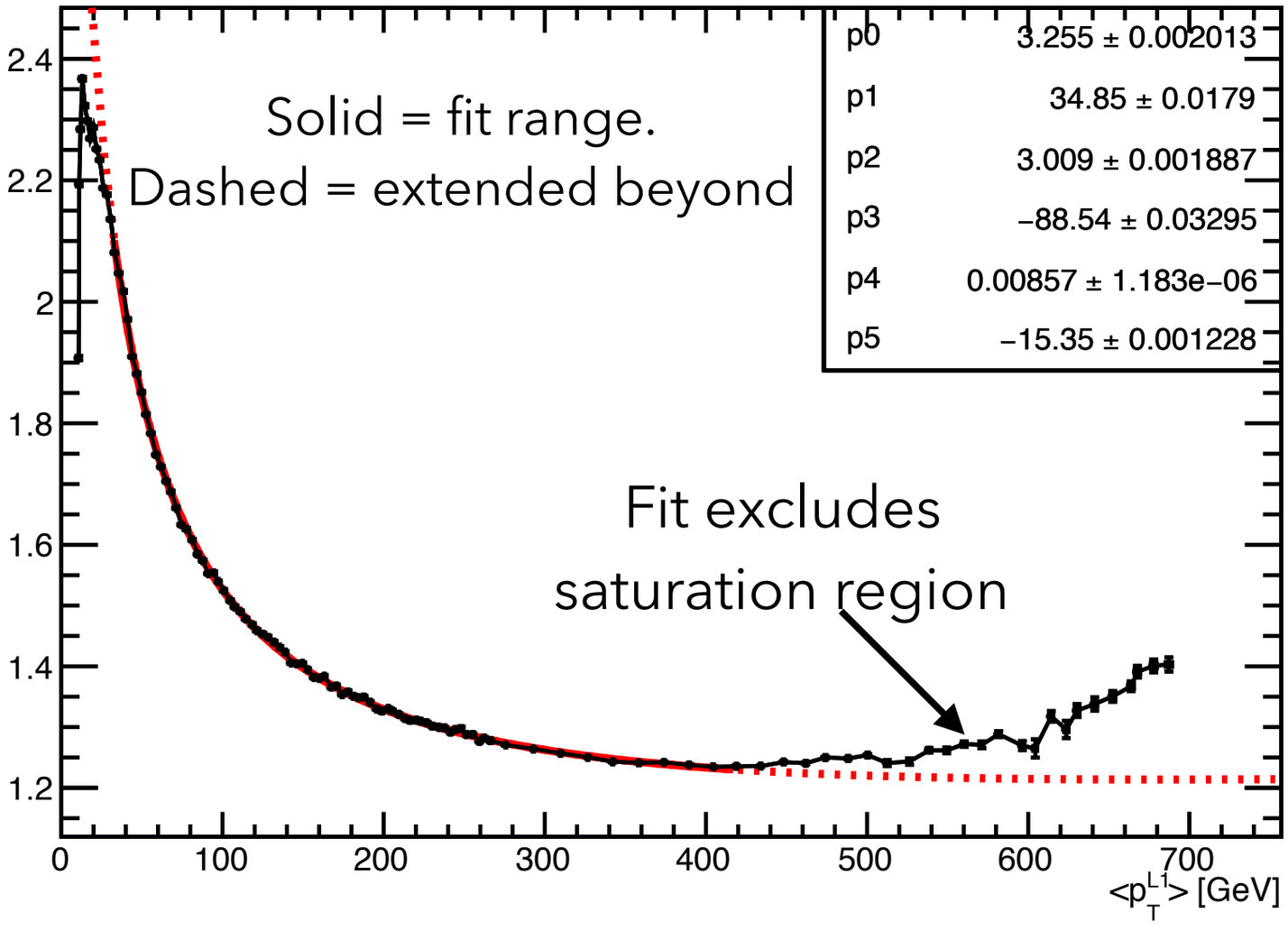
- We do this for all our p_T^{Ref} bins
- We then plot 1/<rsp> Vs $<p_T^{L1}>$ for all the p_T^{Ref} bins (still only considering one η bin):



Lots of things you should know about corrections

- We then fit the curve with the function:

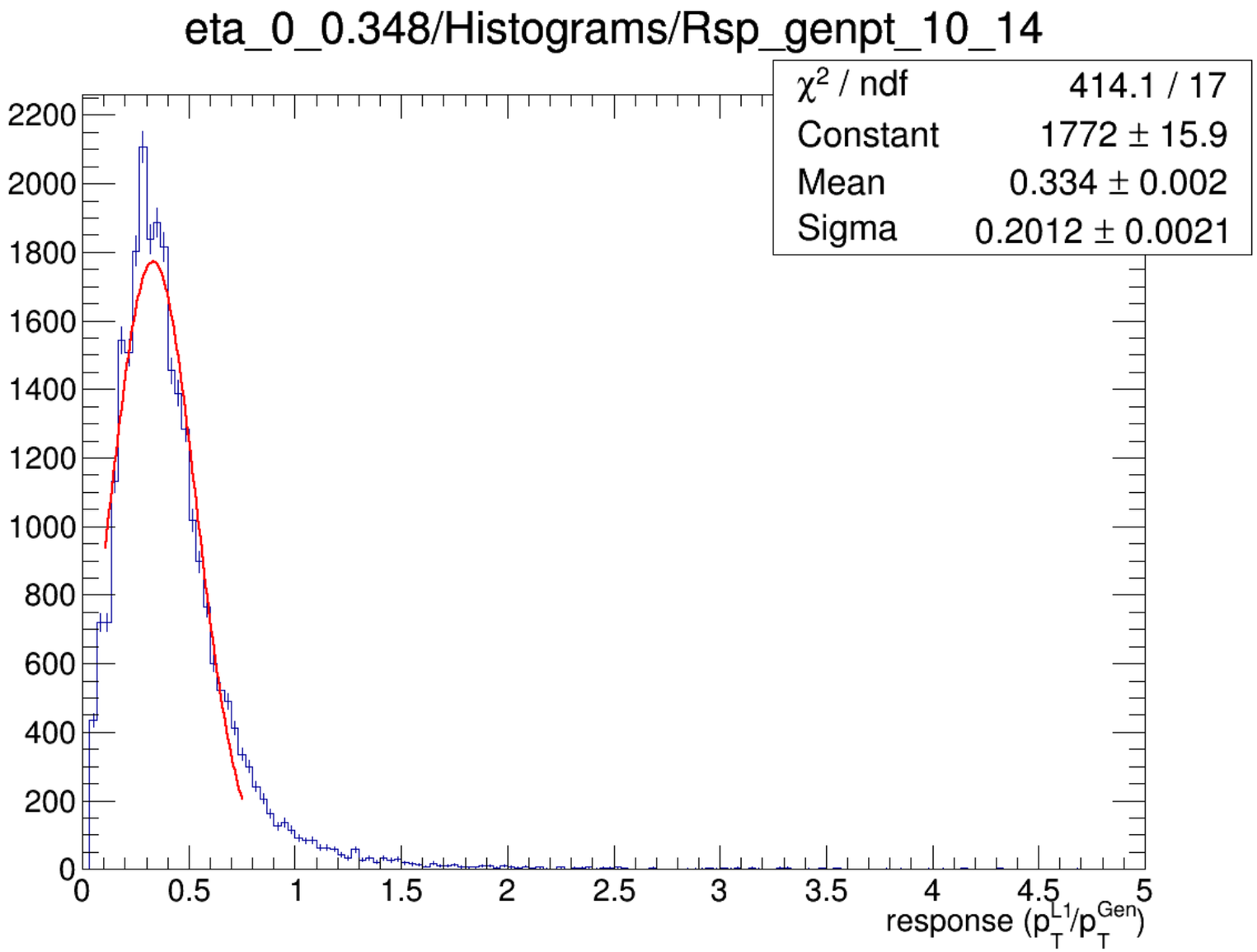
$$E_T^{L1,Corr} = E_T^{L1} \times \left(p_0 + \frac{p_1}{\left(\log_{10} E_T^{L1}\right)^2 + p_2} + p_3 \cdot \exp\left(-p_4 \left(\log_{10} E_T^{L1} - p_5\right)^2\right)\right)$$



- Do this for all η bins, and you're done!

Lots of things you should know about corrections

- It is important to check all the fits - some weird things can happen at small pT
- Newly possible: animated GIFs! (see keynote file)

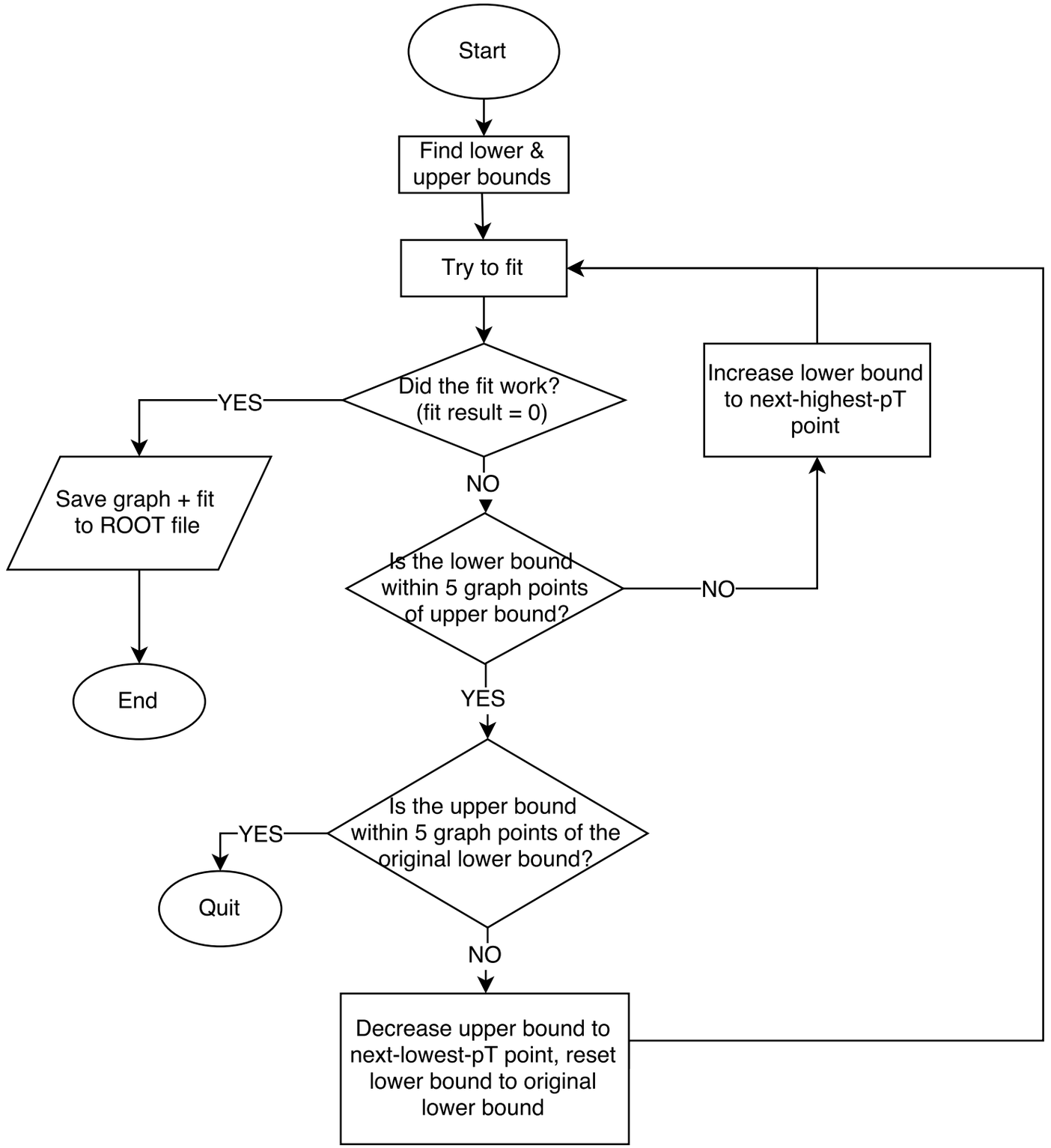


Lots of things you should know about corrections

- The correction function fit can be very sensitive to the starting parameters
  - 6 parameters → lots of parameter space!
- It is also sensitive to the fit range
- To combat this latter problem, we perform an iterative process to try increasingly smaller ranges.
- We start by finding the local maximum at low pT → lower bound for fit
  - If there isn't one, we use the point closest to 10 GeV as the lower bound
- For the upper bound, we find the local minima at higher pT → upper bound for fit
  - Avoid saturation region

Lots of things you should know about corrections

- Follows this procedure →
- If you don’t end up with a decent fit at all, you need to change your starting fit parameters.



How does it look in software

All software can be found at:
https://github.com/raggleton/L1JetEnergyCorrections

bold = program name

CMSSW

Use L1Ntuples package.

Trigger-specific

(GCT vs Stage1…)

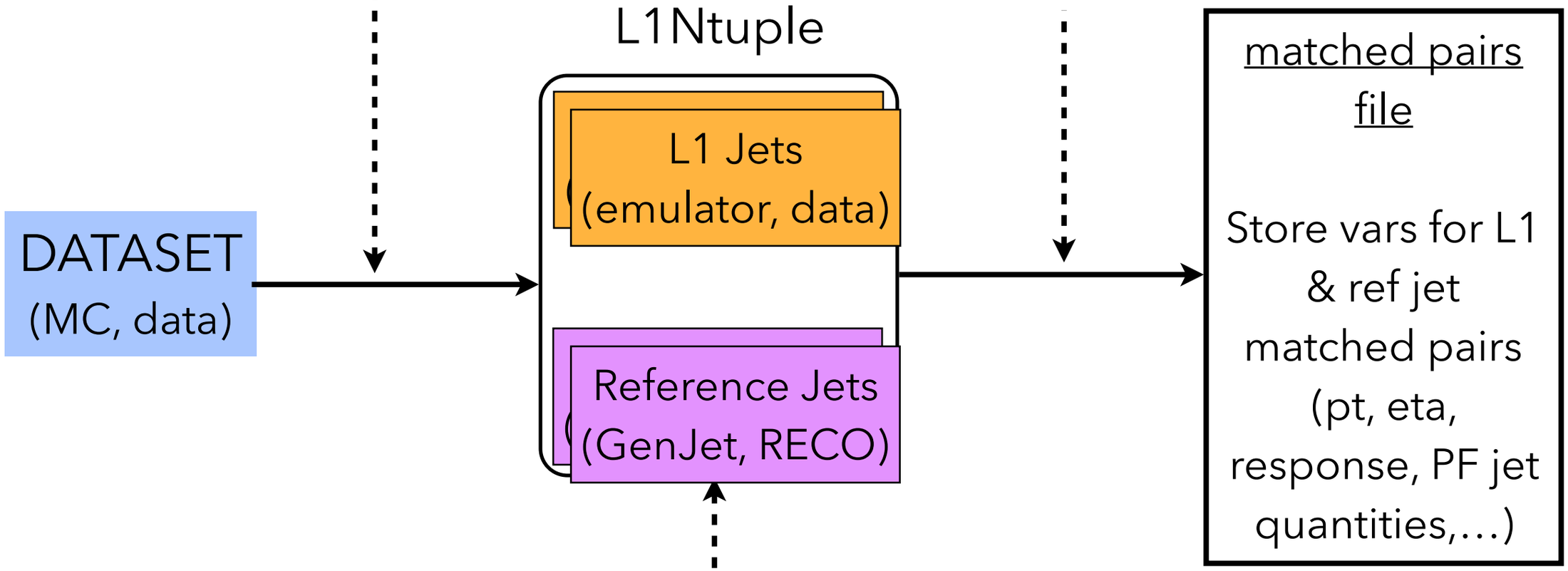
RunMatcher/RunMatcherStage2/ RunMatcherData

Does matching.

Can select L1/reference jet branches.

Also quick'n'dirty calibrate/sort/filter emulator to test calibs

In C++ for speed.

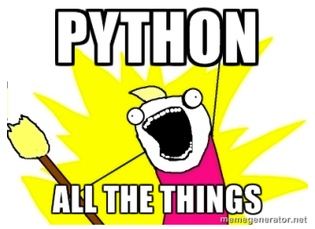


Can store several L1 & ref jet trees (L1 output, L1 internal, ak5, ak4…)

Lightweight.

Also great for exploratory analysis.

bold = program name



runCalibration.py
Make calibration curves

calibration_slides.py
for easy latex presentation

correction_LUT_plot.py
- Make LUT from correction curves.
- Output as ROOT/PyROOT/Numpy code
- Plot low pt range for checking

compareFits.py
Easily pull graphs & fits from various ROOT files & plot them together to compare e.g. for different PU bins

makeResolutionPlots.py
Make resolution plots

resolution_slides.py
for easy latex presentation

matched pairs file

checkCalibration.py
Make 1D & 2D plots of response for various bins/vars

showoffPlots.py
Prettify histograms/graphs, save as PDF, do comparisons

Typical running

- **To make a new set of corrections:**

1. Make Ntuples with reference jets included (GenJets, PF jets) **without any L1JEC** → ROOT files (~ several hours - a day)

2. Use these files do to matching (RunMatcher[Stage2|Data]) → ROOT file (~ hour)

3. Use matched ROOT file in checkCalibration.py to check things look OK pre-calibration → ROOT file with lots of histograms & graphs (~ hour)

4. Use matched ROOT file in runCalibration.py → ROOT file with all component histograms & fits (~ hour)

5. Use showoffPlots.py on the output of (3) and (4) to make pretty plots (PNGs/PDFs) to showoff to TSG (~ mins)

6. Optional: use showoffPlots.py on output of (2) to check everything OK

Typical running

- **To test a new set of corrections:**

  1. Make Ntuples with reference jets included (GenJets, PF jets) **with your new corrections** → ROOT files (~ several hours - a day)

  2. Use these files do to matching (RunMatcher[Stage2|Data]) → ROOT file (~ hour)

  3. Use matched ROOT file in checkCalibration.py to check things look OK pre-calibration → ROOT file with lots of histograms & graphs (~ hour)

  4. Use showoffPlots.py on the output of (3) to make pretty plots (PNGs/PDFs) to showoff to TSG (~ mins)

NTuple making

- Config files are in L1JetEnergyCorrections/python directory
- For MC we need to add in the GenJets - we create a copy of the L1ExtraTree, but fill it with GenJets instead (TODO use L1UpgradeTree)
  - This isn't trivial as the C++ objects are incompatible, so we have a converter plugin: **GenJetToL1Jet**
- We run over the RAW data format, since this contains all the necessary info to re-run the trigger:
  - First we "unpack" the ECAL/HCAL TPs (ecalDigis, hcalDigis modules)
  - Then we can pass them to the Layer 1 emulator to make towers
  - Then pass towers to Layer 2 to make jets, etc.
- For MC, we also store the PU info: the actual number of vertices in each event, as well as the distribution mean that the sample was generated with.

NTuple making

- There are also scripts for submitting on **CRAB3** (L1JetEnergyCorrections/crab) **[PREFERRED]** and on **HTCondor** (L1JetEnergyCorrections/condor)
- These use datasets defined in python/X_samples.py (X = data|mc) for easy swapping between different samples

Matching

- The matching program is written in C++ and requires compiling
- This is done automatically by doing `scram b -j9`
- Build options are defined in bin/BuildFile.xml (also specifies which programs get compiled)
- ΔR Matcher implemented in src/DeltaR_Matcher.cc

Batch-ify

- Unfortunately, interactively running is slow
- We can do some parallelisation - can split up task by η bin!
- All big scripts (checkCalibrations, makeResolutionPlots, runCalibration) allow you to specify 1 or more η bins to run over.
- Also scripts for running on batch systems:
  - LXBATCH **[warning: not used for a while, not as fully developed]**
  - HTCondor at Bristol
- Also batch scripts for RunMatcher (if your NTuple files are split across many files - can do matching VERY quickly)
- In times of trouble, can also make NTuples on HTCondor - but bit sketchy

Batch-ify

- For HTCondor, batch scripts are in bin/HTCondor:
  - submit_matcher_dag.py (for RunMatcher)
  - submit_checkCalib.sh (for checkCalibration.py)
  - submit_calibration_dag.sh (for runCalibration.py)
  - submit_resolution_dag.sh (for makeResolutionPlots.py)

- For PBS/Lxbatch, batch scripts are in bin/PBS:
  - macher_batch.sh (for RunMatcher)
  - check_batch.sh (for checkCalibration.py)
  - calbration_batch.sh (for runCalibration.py)
  - resolution_batch.sh (for makeResolutionPlots.py)

Computing tips

- I work on Soolin & use HTCondor because:
  - fewer users than Lxplus - (can also shout at CPU hoggers)
  - batch system much faster at processing jobs - no queue (more slots?)
  - tons of storage space on /hdfs - 100GB on AFS is a joke. (of course, there's also EOS)
  - you have local support in the form of Luke, Winnie (& me)