# nTuples for NNLO

Daniel Maître

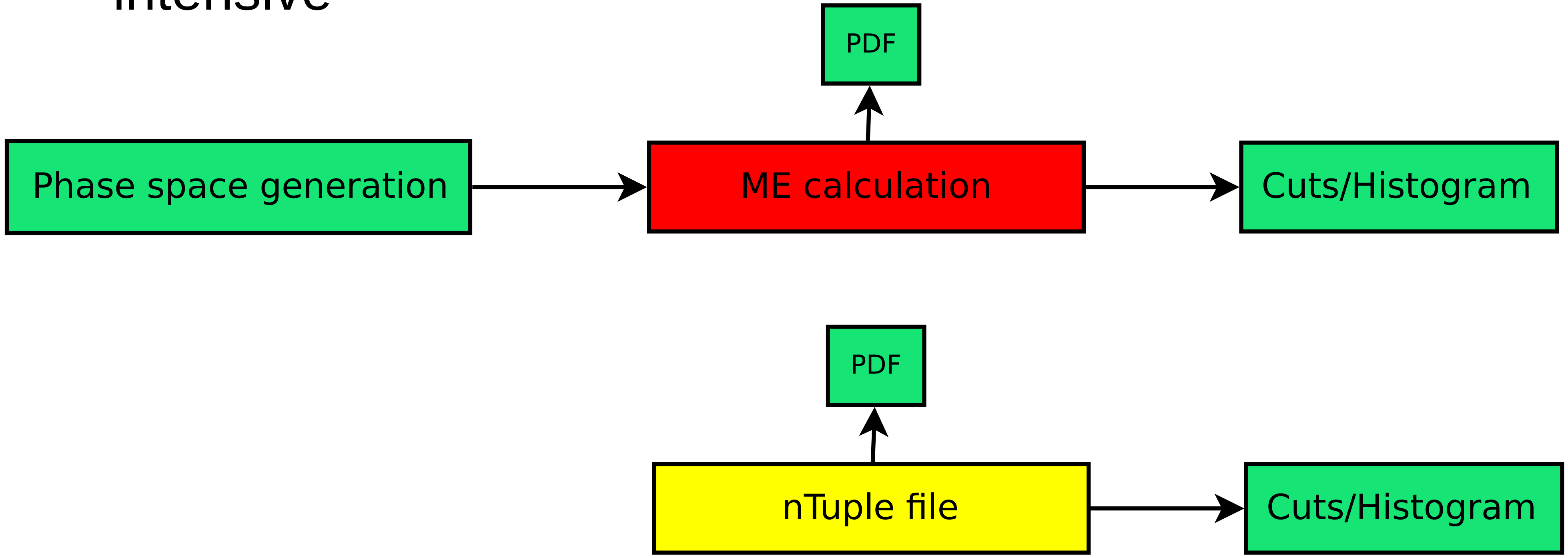
IPPP Durham

## Outline

- Why
- How

# n-Tuple files [arXiv:1310.7439]

- High multiplicity NLO calculations are computationally intensive



- Matrix elements are expensive
- Jet clustering, observables, PDF evaluation are relatively cheap
- Store matrix element, PS point and the information necessary to change scales

## NLO nTuple files [arXiv:1310.7439]

- Advantages
  - One can change the analysis cuts, add observables
  - Cheap scale variation and PDF errors (otherwise extremely expensive)
  - Easy communication between theorists and experimenters
  - No need for specific know-how of the tool which produced the NLO calculation
  - Easier to “endorse” an event file than a program
- Disadvantage
  - Large files
  - Generation cuts need to be loose enough to accommodate many analysis → efficiency cost

## nTupleReader library [arXiv:1310.7439]

- We provide a C++ library to facilitate the use of the n-Tuple files
- Allows:
  - Change of factorisation and renormalisation scales
  - Change of pdf (from LHAPDF set), including error sets
- Has a Python interface
- Template for a customised implementation
- Available on hepforge

  https://blackhat.hepforge.org/trac/wiki/BlackHatSherpaNtuples

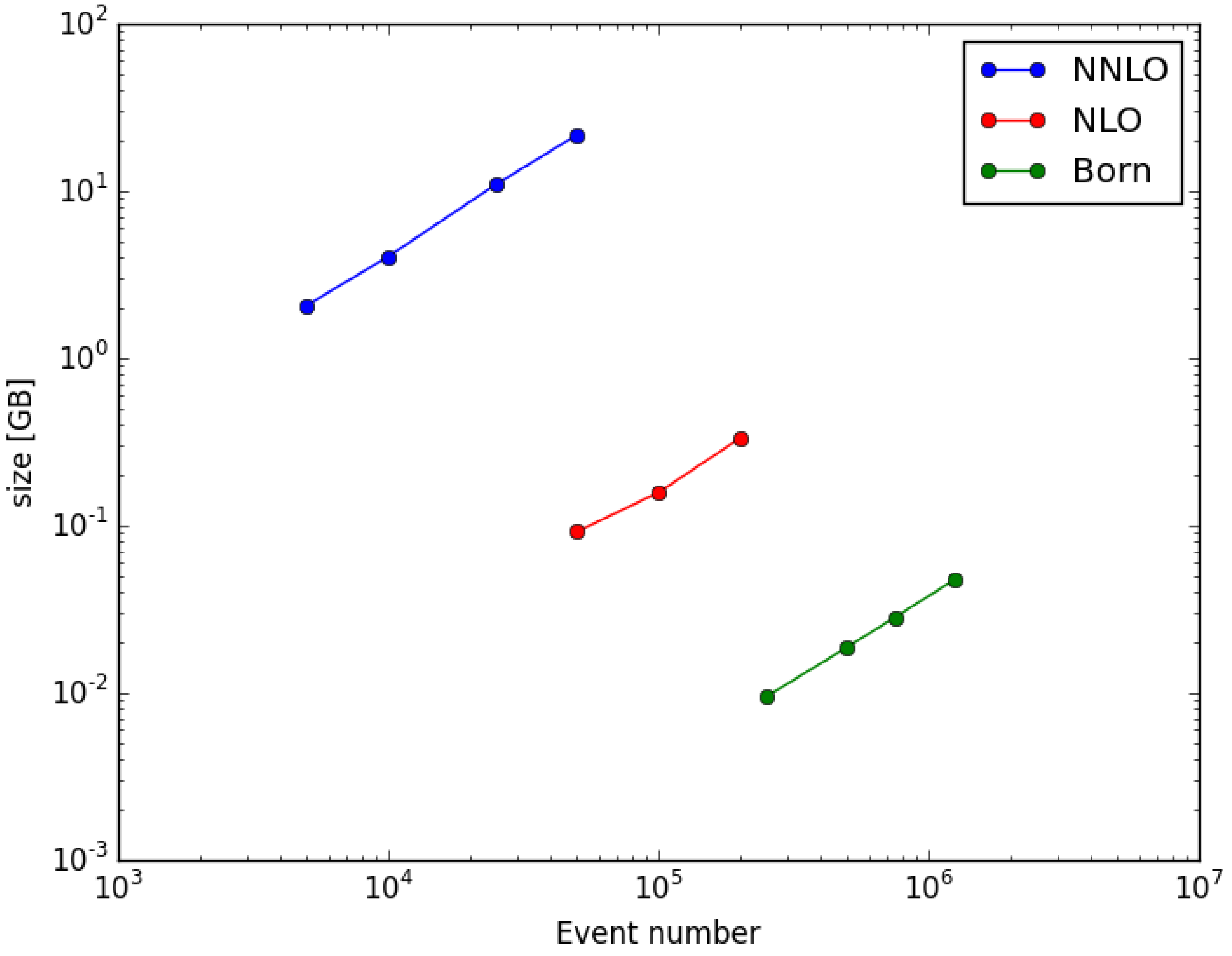
## nTuples for NNLO

- nTuples have proven useful for NLO
- Can they be as useful for NNLO?
- Same advantages and same disadvantages but amplified:
  - Programs are more complex
  - Larger files:
    - Many more pieces in the calculation
    - More logarithm coefficients
- Main question: is the size reasonable?

## Experiment in Les Houches

- Used EErad3 [Gehrmann-De Ridder, Gehrmann, Glover, Heinrich; arXiv:0710.0346] to generate NNLO ntuples for $e^+e^- \rightarrow$ 3 jets
- Storage structure very similar to NLO nTuples
- Easier than hadronic :
  - No initial state information
  - Strong coupling dependence trivial (no real need to store coefficients of logarithms)

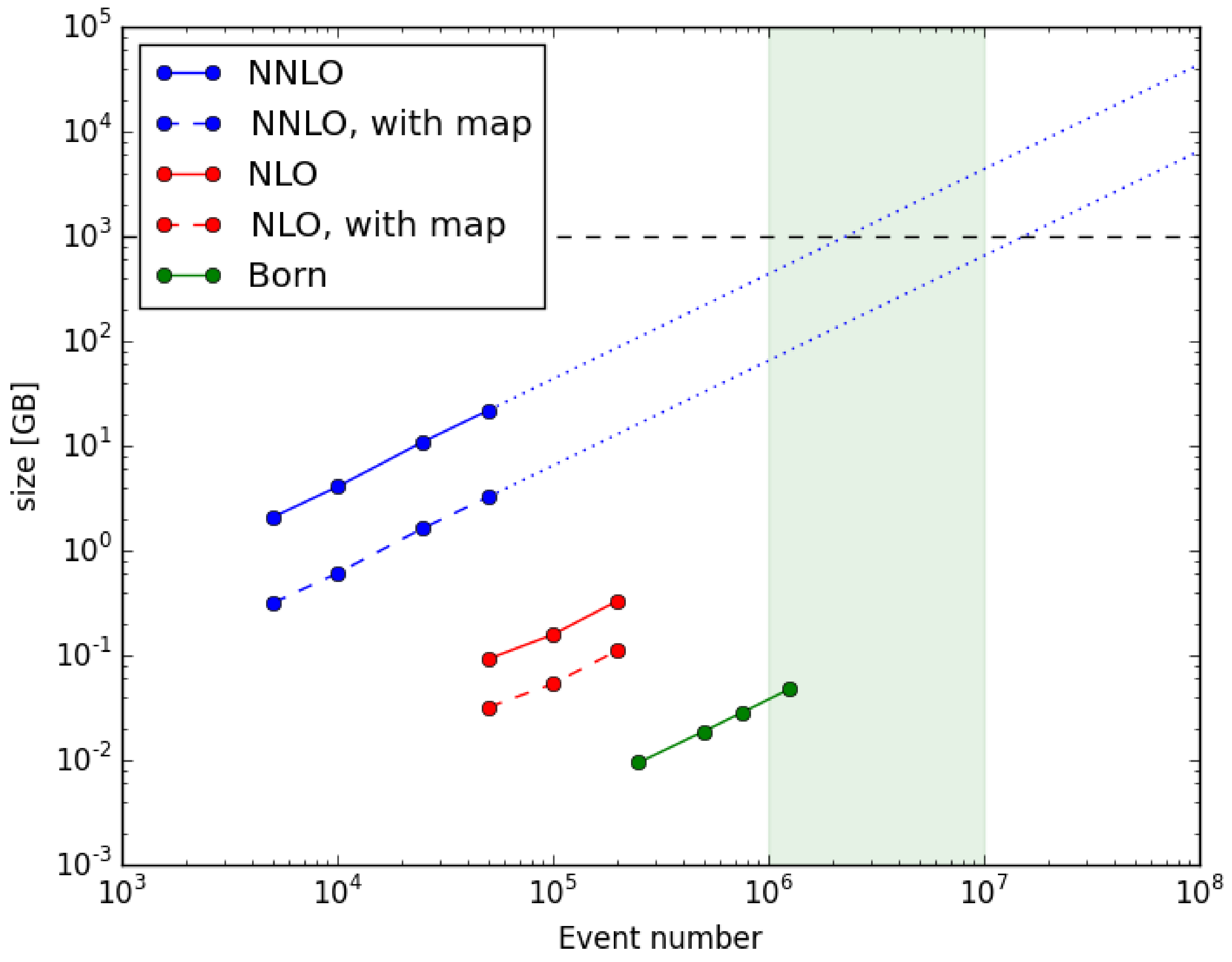
# Size vs Event number



## Using mapping information

- The most space-consuming part is the double real part
  - More final state momenta
  - Need much statistics because of subtraction terms
- For each real-real phase-space point we have many subtraction terms
- Each of them has a different set of momenta given by a $(n+2) \rightarrow n$ or $(n+1) \rightarrow n$ map
- We can save much space if we simply record the mapping that was used instead of the momenta
- The downside is that
  - there is more calculation at the moment of reading the nTuple
  - More coupling between nTuple file and code that produced it

## Extrapolated file size



## File sizes

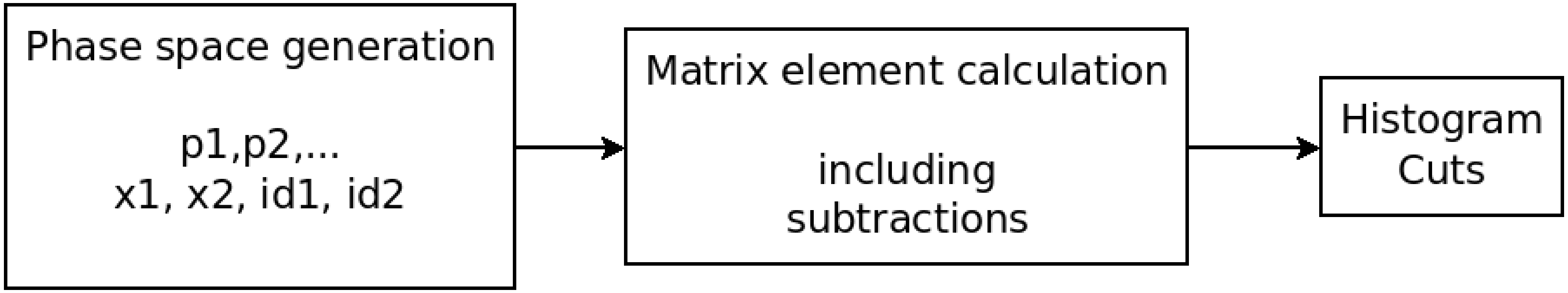
- EErad3 “true” NNLO calculation with NNLO subtraction
- Processes calculated with the $q_T$ subtraction or n-jettiness methods look more like two NLO calculations → could be more suitable for nTuple production
- Current programs are optimized for CPU usage, the trad-offs between “unweighting” and run time can be different if one wants to optimise the storage size

## Hadronic initial states

- Work in progress to extend NNLO nTuples to hadronic initial states
- Working prototype for DIS and hadron-hadron
- Currently working on hadron-hadron
- Goal: provide as general an interface as possible
  - Not every NNLO program is providing this information easily
  - Can use an approach to minimize the impact of nTuples production to the NNLO program
  - Only add a handful of lines to the program at targeted places
  - Run several instances of the program with different inputs and reverse engineer the weight
  - Trade some CPU time to same my wall clock time

# Generic NNLO program

- A typical NNLO program can be decomposed into several parts



- The histograms/cuts part is typically quite easy to identify as users of the program are expected to implement their observables/cuts
- extract momenta and initial state information by inserting a line of code in NNLO phase space generation routine
- extract weights in the histogram routine
- subtraction terms evaluated with different momenta, I can use the routine for the cuts to siphon the PS information.

# Information for nTuple

- I can get the weights from the histogram procedure, but if I want to change the PDFs and scale I need more information:
  - Coefficients of scale logarithms
  - PDF information
    - Initial state flavors
    - Momentum fractions x1, x2
    - Factorisation scale

$$\begin{aligned} \omega &= \alpha_S^n \, pdf(x_1, id_1) \, pdf(x_2, id_2) \\ &\quad \times \left(c_0 + c_1 \log(\mu^2 + \ldots)\right) \end{aligned}$$

## Initial state information

- LHAPDF is a good place to collect information:
  - Pdf information
    - Initial state flavors
    - Momentum fractions x1, x2
    - Factorisation scale
  - Often used to calculate alphas
    - Renormalisation scale

## Scale dependence

- To get the scale dependence we need a different strategy
- Run parallel instances of the program with different scale compare the weights obtained and solve for the logarithm coefficients
- The part with the most scale logarithms are not the hardest, so the overhead of running multiple instance is not too bad

# Multiple PDF terms

- There can be complications with pdfs:

$$\omega = c_1 pdf(x_1, x_2, id_1, id_2, Q) + c_2 pdf(x'_1, x'_2, id'_1, id'_2, Q)$$

- Could solve for the two coefficients with two parallel runs with different pdf scales
- Better solution is to modify the pdf function to return 0 instead of the pdf every second time

$$\begin{array}{ccccccc} \omega_1 & = & c_1 & pdf(x_1, x_2, id_1, id_2, Q) & + & c_2 & 0 \\ \omega_2 & = & c_1 & 0 & + & c_2 & pdf(x'_1, x'_2, id'_1, id'_2, Q) \end{array}$$

# Multiple PDF terms

- Another problem is that the call to LHAPDF typically returns the value for all partons
  - we can't tell which one or which is used by the NNLO program
  - We can use the same strategy and run 13 different instances

$$
\begin{aligned}
f_1(x) &= (*,0,0,0,0,0,0,0,0,0,0,0,0)\\
f_2(x) &= (0,*,0,0,0,0,0,0,0,0,0,0,0)\\
f_3(x) &= (0,0,*,0,0,0,0,0,0,0,0,0,0)\\
f_4(x) &= (0,0,0,*,0,0,0,0,0,0,0,0,0)\\
f_5(x) &= (0,0,0,0,*,0,0,0,0,0,0,0,0)\\
f_6(x) &= (0,0,0,0,0,*,0,0,0,0,0,0,0)\\
f_7(x) &= (0,0,0,0,0,0,*,0,0,0,0,0,0)\\
f_8(x) &= (0,0,0,0,0,0,0,*,0,0,0,0,0)\\
f_9(x) &= (0,0,0,0,0,0,0,0,*,0,0,0,0)\\
f_{10}(x) &= (0,0,0,0,0,0,0,0,0,*,0,0,0)\\
f_{11}(x) &= (0,0,0,0,0,0,0,0,0,0,*,0,0)\\
f_{12}(x) &= (0,0,0,0,0,0,0,0,0,0,0,*,0)\\
f_{13}(x) &= (0,0,0,0,0,0,0,0,0,0,0,0,*)
\end{aligned}
$$

## Guessing the PDF

- The individual pdfs are useful to understand what the NNLO program does but quite costly, instead one can have two runs, one with the original PDF, one with the pdf values replaced by 1

$$\begin{array}{rclcccc} \omega_1 &=& c_1 & pdf(x_1, x_2, id_1, id_2, Q) & + & c_2 & 0 \\ \omega_2 &=& c_1 & 0 & + & c_2 & pdf(x'_1, x'_2, id'_1, id'_2, Q) \\ \omega_1^{pdf=1} &=& c_1 & 1 & + & c_2 & 0 \\ \omega_2^{pdf=1} &=& c_1 & 0 & + & c_2 & 1 \end{array}$$

- We obtain the coefficient and the pdf part, which we can compare with combinations we expect

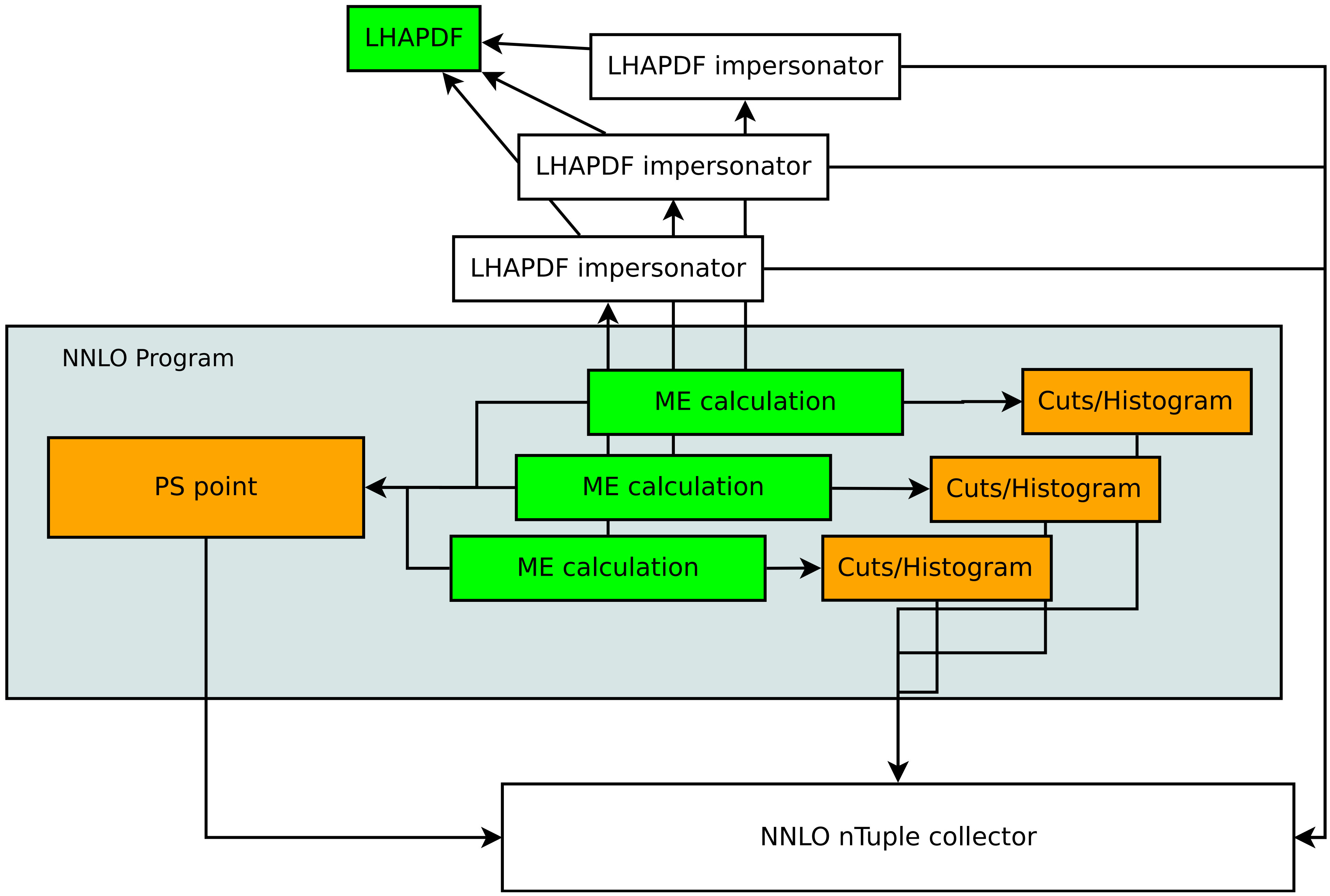
$$pdf = g(x_1)u(x_2), \left(\sum_q q(x_1)\right) g(x_2), \left(\sum_q q(x_1)\bar{q}(x_2)\right), \ldots$$

- If the combination is not found, we can investigate with the previous technique.

## Guessing coupling constant

- In most cases we know the power of the coupling constant from the part (born, real, double real) of the NNLO program we are running
- if not we can use two parallel instances of the program, one normal and one with the coupling constant set to 1 and compare the ratio

# Implementation



# LHAPDF impersonator

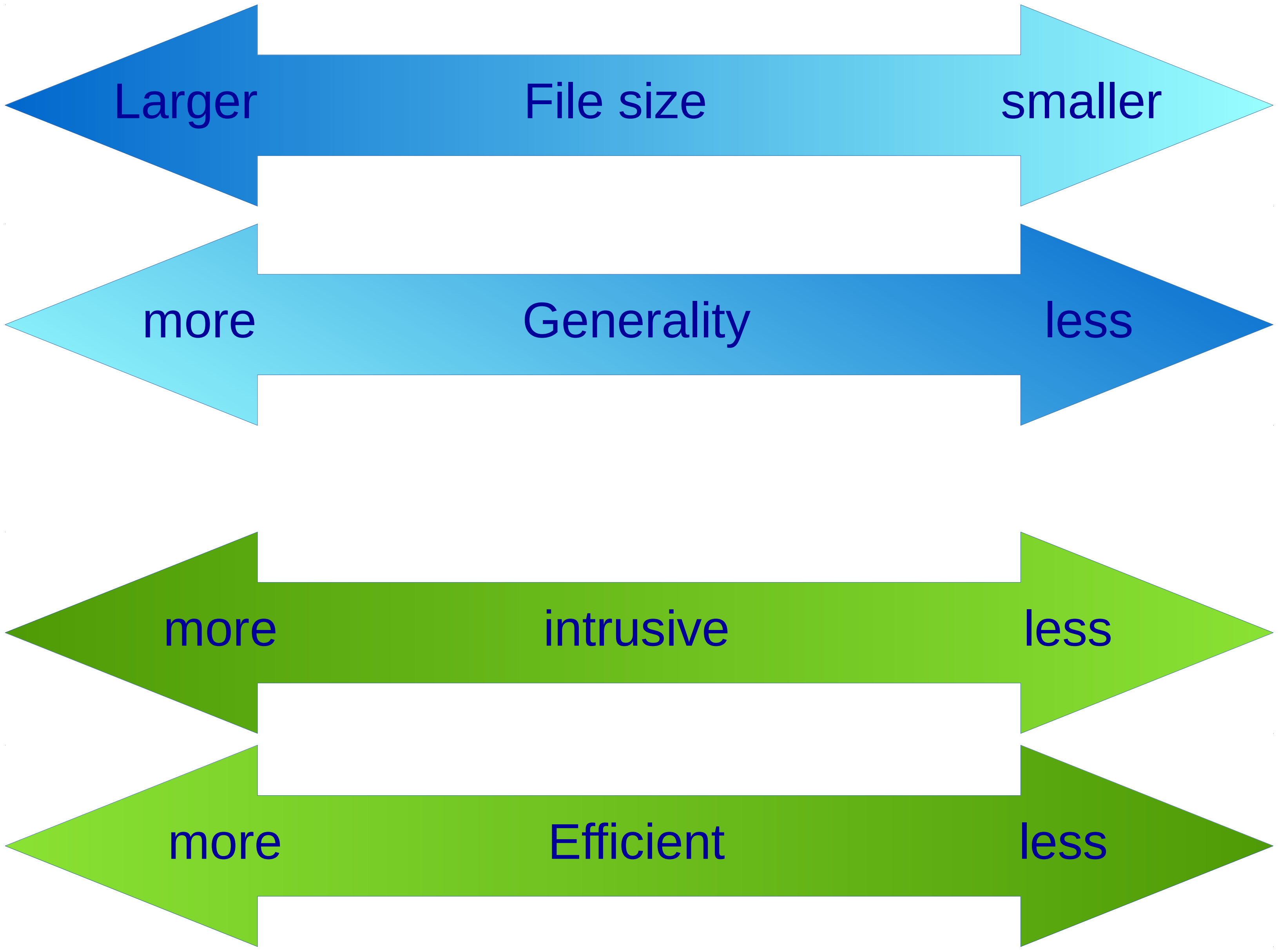
- Allows to alternate vanishing and real pdf to disentangle different pdf terms
- Allows to filter specific initial state
- Reports pdf arguments to the nTuple collector
- Allows to set alphas values to one
- Reports
- Is implemented using a hacking technique so there is no need to modify neither the NNLO program nor the LHAPDF code

## Additional difficulties

- Caching can cause confusion when the order of pdf, alphas, cuts and histograms are perturbed.
- Need to avoid grid adaptation to ensure synchronisation of the threads

# NNLO nTuple files

- Trade offs



## Conclusion

- nTuples could be a viable option for NNLO
- nTuples can be extracted with minimal changes to code
- Extraction can be more efficient with more intrusive modifications
- Storage can be made more efficient at the cost of read-time speed or changes to the original code
- nTuple files can help disseminate NNLO results