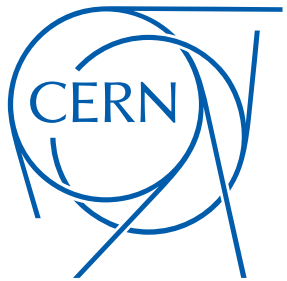
# Sherpa Status and Plans

Marek Schönherr*

CERN

Physics Event Generator Computing Workshop

26 Nov 2018




*With plenty of input from S. Höche.

# Content

1. The SHERPA Event Generator framework
2. Code structure
3. Code performance
4. Conclusion

# The SHERPA Event Generator framework

1 The SHERPA Event Generator framework

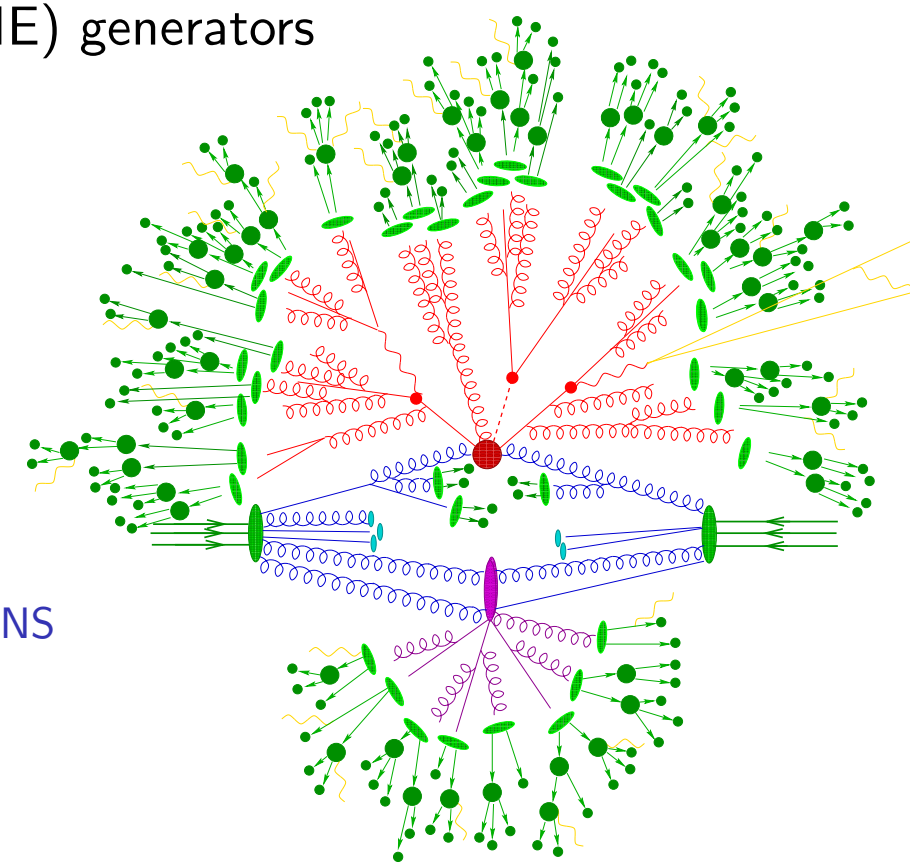
2 Code structure

3 Code performance

4 Conclusion

# The SHERPA event generator framework

- Two multi-purpose Matrix Element (ME) generators AMEGIC, COMIX
- Two Parton Shower (PS) generators CSSHOWER, DIRE
- A multiple interaction simulation à la PYTHIA AMISIC
- A cluster fragmentation module AHADIC, or interface to Lund string
- A hadron and $\tau$ decay package HADRONS
- A higher order QED generator using YFS-resummation PHOTONS
- A minimum bias simulation SHRiMPS



**Sherpa's traditional strength is the perturbative part of the event**

LO, NLO, NNLO, LOPS, NLOPS, NNLOPS, MEPS, MENLOPS, MEPS@NLO

# Available physics simulations

**Fixed order calculations**

- matrix elements only, implies fixed multiplicities
- no parton shower, no non-perturbative physics, no particle level

$\Rightarrow$ LO, NLO, NNLO

**Parton shower matched calculations**

- comb. of fixed order calc. and parton shower for one multiplicity
- particle level predictions, no multijet observables

$\Rightarrow$ LOPS, NLOPS, NNLOPS

**Multijet merged calculations**

- combination of parton shower matched calculations for increasing final state multiplicities (mostly jets)
- particle level predictions, multijet observables

$\Rightarrow$ MEPS(@LO), MEPS@NLO

# Changing scales and PDFs/$\alpha_s(m_Z)$

- on-the-fly scale and PDF variations for ME part
  $\rightarrow$ use named weights in HEPMC (available since HEPMC 2.06)
- two avenues:
  1) output weights for predetermined alternative scale, PDF, $\alpha_s(m_Z)$
     + number of weights grows linearly with requested variations
     – possible variations fixed at generation time
     $\rightarrow$ used currently by LHC experiments
  2) output coefficients
     + arbitrary a posteriori variations possible
     $\pm$ number of scales, parameters & coefficients dependent on type of event and multiplicity, e.g.

| | | |
|---|---|---|
| $V + 0j$ @ LO | **6** | (type, B, $\mathcal{O}(\alpha_s)$, $\mathcal{O}(\alpha)$, $\mu_R$, $\mu_F$) |
| $V + 4j$ @ LO | up to **37** | (as above + PDF wgt. from merging) |
| $V + 2j$ @ NLO (S) | up to **90** | (as above + comp. of S event) |
| $V + 2j$ @ NLO (H) | up to **158** | (as above + comp. of H event) |

     number much lower on avarage, as often many components zero
     – need generator-specific lib to recombine into weights
     $\rightarrow$ used in e.g. MCGRID

# Code structure

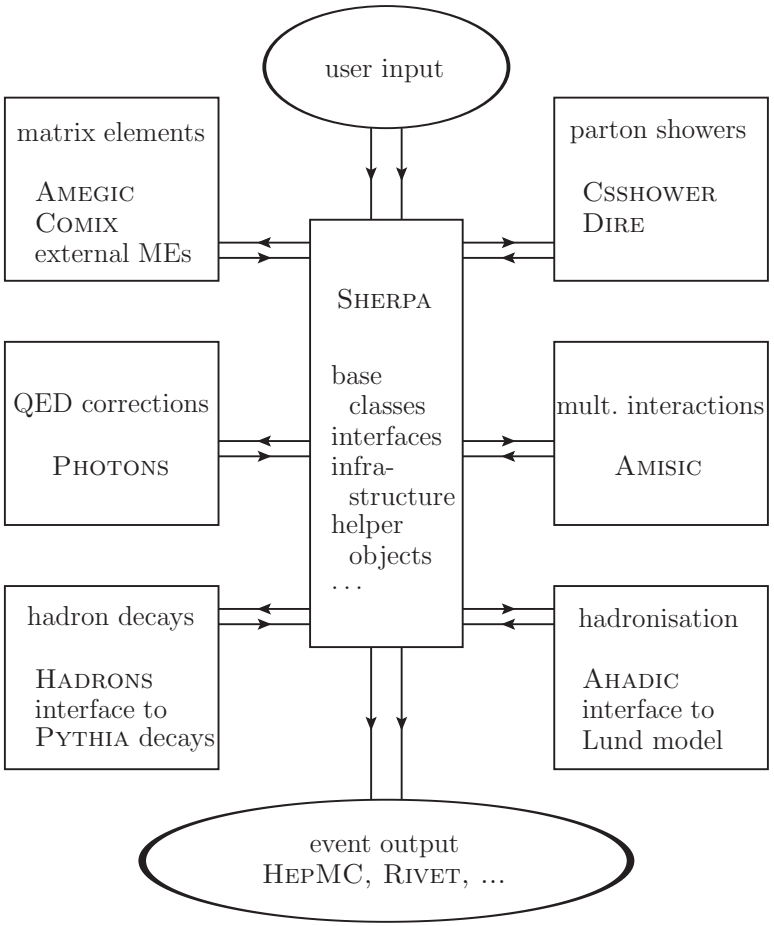
1 The SHERPA Event Generator framework

2 Code structure

3 Code performance

4 Conclusion

# Code structure




- most physics modules reside in shared libraries loaded dynamically at run time
  - base classes defining interface functionalities reside in SHERPA
  - ME, PS, PDFs, etc. loaded at run time
- same technology also used for key user definable functions:
  - scale definitions
  - phase space cuts
  - ...

→ customisable by user without needing to touch or recompile SHERPA

# Run strategy

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)
2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     $\rightarrow$ make weights as uniform as possible
   - store in database
3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

# Run strategy

needs to be done once
results platform independent

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)
2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     $\rightarrow$ make weights as uniform as possible
   - store in database
3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

# Run strategy

needs to be done once
results platform independent

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)

single core

2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     $\rightarrow$ make weights as uniform as possible
   - store in database

3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

# Run strategy

needs to be done once
results platform independent

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)

single core

2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     $\rightarrow$ make weights as uniform as possible
   - store in database

MPI parallisable
linear scaling up to few thousand cores

3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

# Run strategy

needs to be done once
results platform independent

1) **initialisation**
   - write matrix elements source code (AMEGIC)
   - write databases with list of existing processes and mapping information (AMEGIC/COMIX)

   → single core

2) **integration**
   - load matrix elements
   - determination of relative cross section of all subprocs
   - optimisation of phase space channels
     → make weights as uniform as possible
   - store in database

   → MPI parallisable
   linear scaling up to few thousand cores

3) **event generation**
   - load matrix elements
   - read phase space channel parameters
   - generate events

   → single core
   MPI parallisable but no benefits

# Code performance

1 The SHERPA Event Generator framework

2 Code structure

3 Code performance

4 Conclusion

# Code performance

## LO merging using COMIX

| Process $W^-+$ | 0j | ≤1j | ≤2j | ≤3j | ≤4j |
|---|---|---|---|---|---|
| RAM Usage | 39 MB | 44 MB | 49 MB | 64 MB | 173 MB |
| Initialization time | <1s | <1s | 3s | 22s | 7m 7s |
| Startup time | <1s | <1s | <1s | <1s | 2s |
| Integration time | 25s | 3m 19s | 34m 8s | 3h 12m | 2d 17h |
| 10k weighted evts | 3m 24s | 3m 51s | 4m 2s | 4m 4s | 4m 21s |
| 10k unweigthed evts | 3m 20s | 4m 39s | 11m 47s | 35m 54s | 4h 3m |

## NLO merging using AMEGIC+BLACKHAT/COMIX (S/H-events)

| Process $W^-+$ | 0j | ≤1j | ≤2j |
|---|---|---|---|
| RAM Usage | 51 MB | 112 MB | 572 MB |
| Initialization time | 1s | 20s | 4m 6s |
| Startup time | <1s | 2s | 18s |
| Integration time | 20m 48s | 4h 45m | 5d 23h |
| 10k weighted evts | 3m 58s | 4m 38s | 6m 48s |
| 10k unweighted evts | 4m 14s | 4h 8m | 24h 54m |

On dual 8-core Intel® Xeon® E5-2660 @ 2.20GHz using MPI parallelisation, timings quoted cumulative.

# Code performance

## LO merging using COMIX

| Process $W^-+$ | 0j | ≤1j | ≤2j | ≤3j | ≤4j |
|---|---|---|---|---|---|
| RAM Usage | 39 MB | 44 MB | 49 MB | 64 MB | 173 MB |
| Initialization time | <1s | <1s | 3s | 22s | 7m 7s |
| Startup time | <1s | <1s | <1s | <1s | 2s |
| Integration time | 25s | 3m 19s | 34m 8s | 3h 12m | 2d 17h |
| 10k weighted evts | 3m 24s | 3m 51s | 4m 2s | 4m 4s | 4m 21s |
| 10k unweigthed evts | 3m 20s | 4m 39s | 11m 47s | 35m 54s | 4h 3m |

## NLO merging using AMEGIC+BLACKHAT/COMIX (S/H-events)

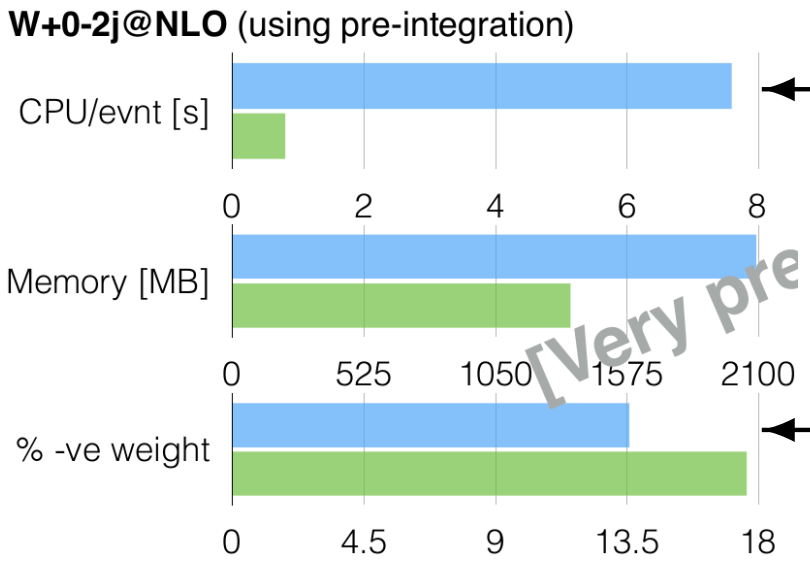
| Process $W^-+$ | 0j | ≤1j | ≤2j |
|---|---|---|---|
| RAM Usage | 51 MB | 112 MB | 572 MB |
| Initialization time | 1s | 20s | 4m 6s |
| Startup time | <1s | 2s | 18s |
| Integration time | 20m 48s | 4h 45m | 5d 23h |
| 10k weighted evts | 3m 58s | 4m 38s | 6m 48s |
| 10k unweighted evts | 4m 14s | 4h 8m | 24h 54m |

bottleneck, steps taken to reduce this look promising

factor 10 improvements seem feasible

On dual 8-core Intel® Xeon® E5-2660 @ 2.20GHz using MPI parallelisation, timings quoted cumulative.

# Code performance




Dominated by $W + 2j$ H events, which are dominated by CKKW clustering, ie. scale determination. Using $H'_{\mathrm{T}}$ approximation instead (like in MG) gives **factor 4** improvement: 2s/evt
**Available in SHERPA-2.2.x.**

Dominated by $W + 2j$ H events, can also be improved upon.

[taken from Josh's slides]

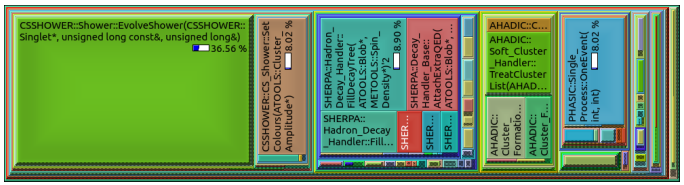
$\Rightarrow$ **more improvements possible with non-default settings**
$\rightarrow$ approximate physics, but substantially improve performance

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

**0j**

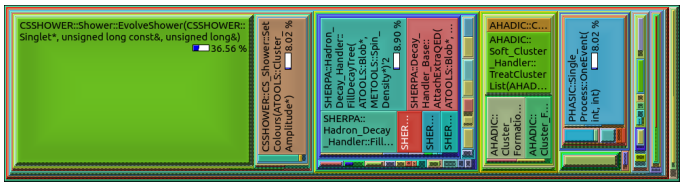


# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

**0j**



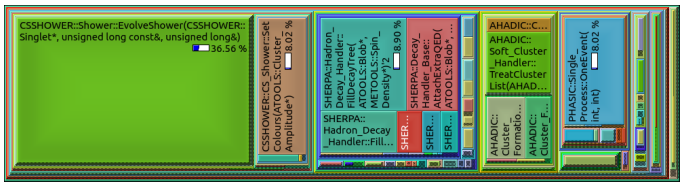

- matrix elements

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

**0j**




parton shower (incl. MI)

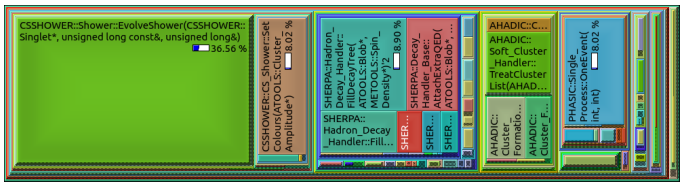
matrix elements

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

**0j**




parton shower (incl. MI)

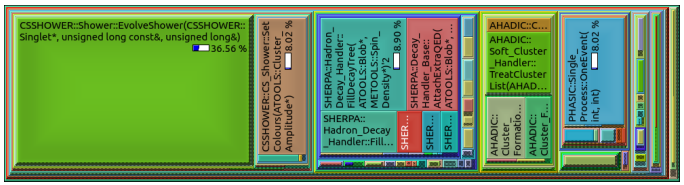
matrix elements mult. interactions

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

**0j**




parton shower (incl. MI)

hadronisation

mult. interactions

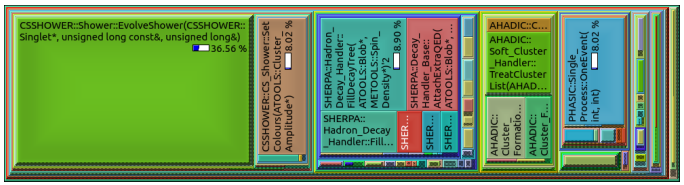
matrix elements

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

**0j**




parton shower (incl. MI)

hadron decays and QED corrs.

hadronisation

mult. interactions

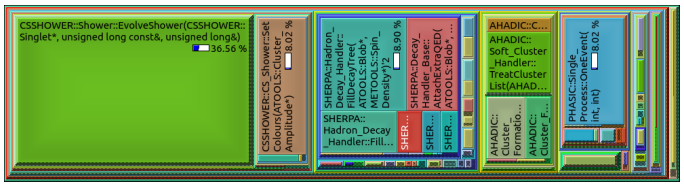
matrix elements

# Timing distribution: LO merging using Comix

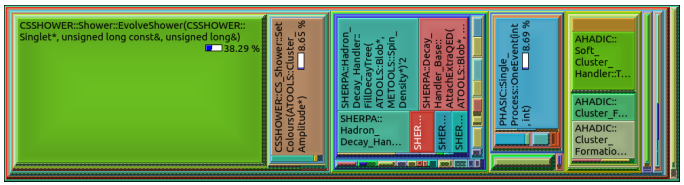
**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

**0j**




**1j**

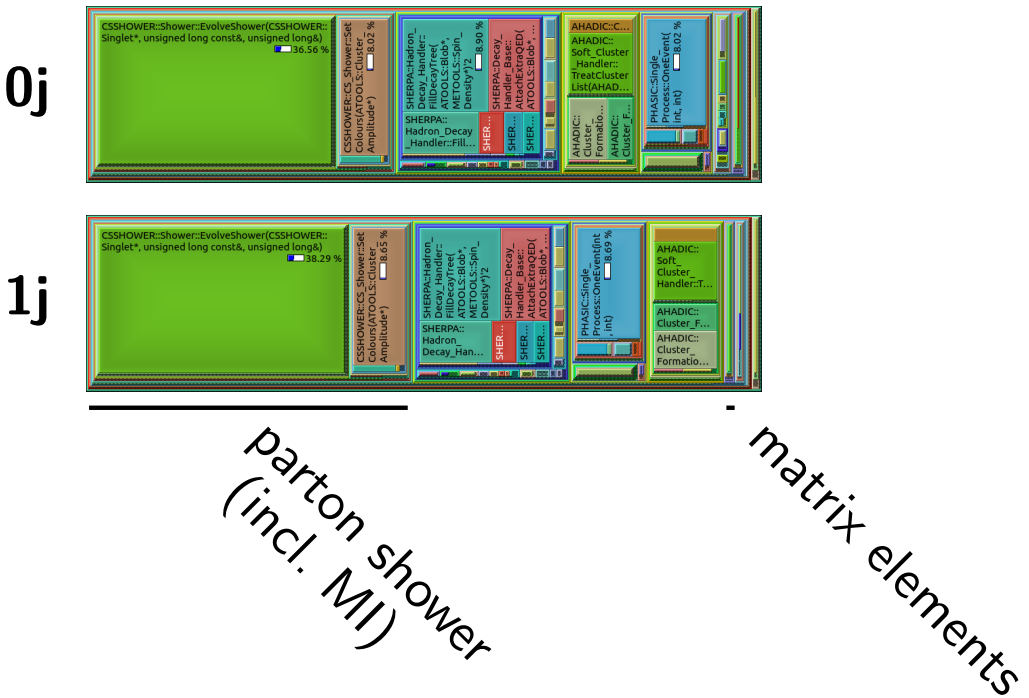



- matrix elements

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

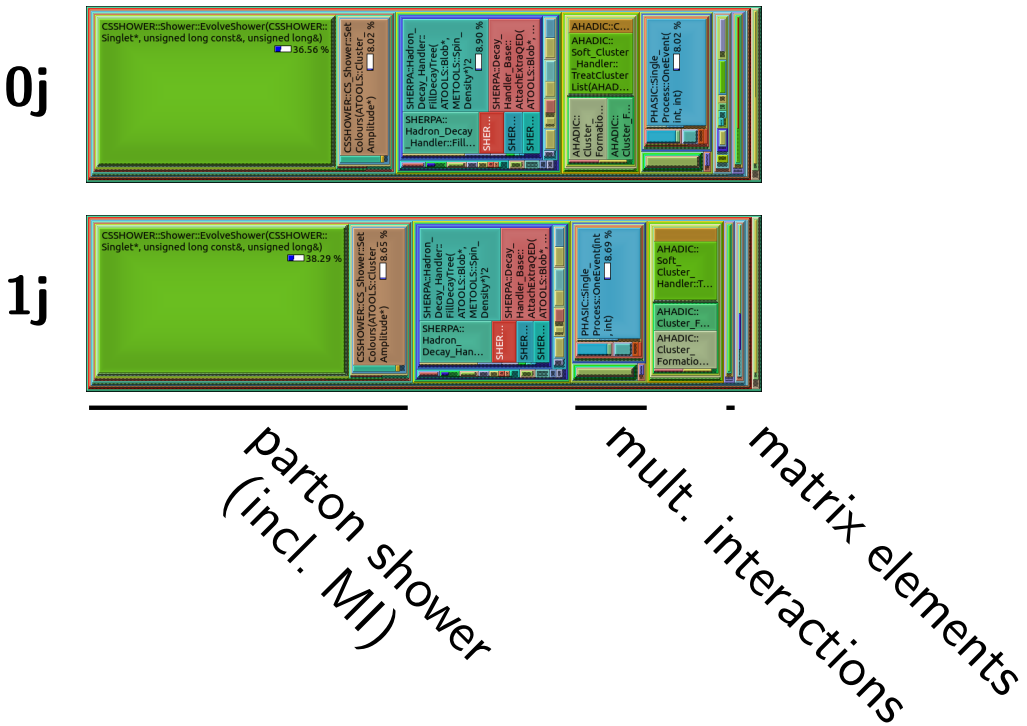
**Unweighted** W+0..≤4j@LO



# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

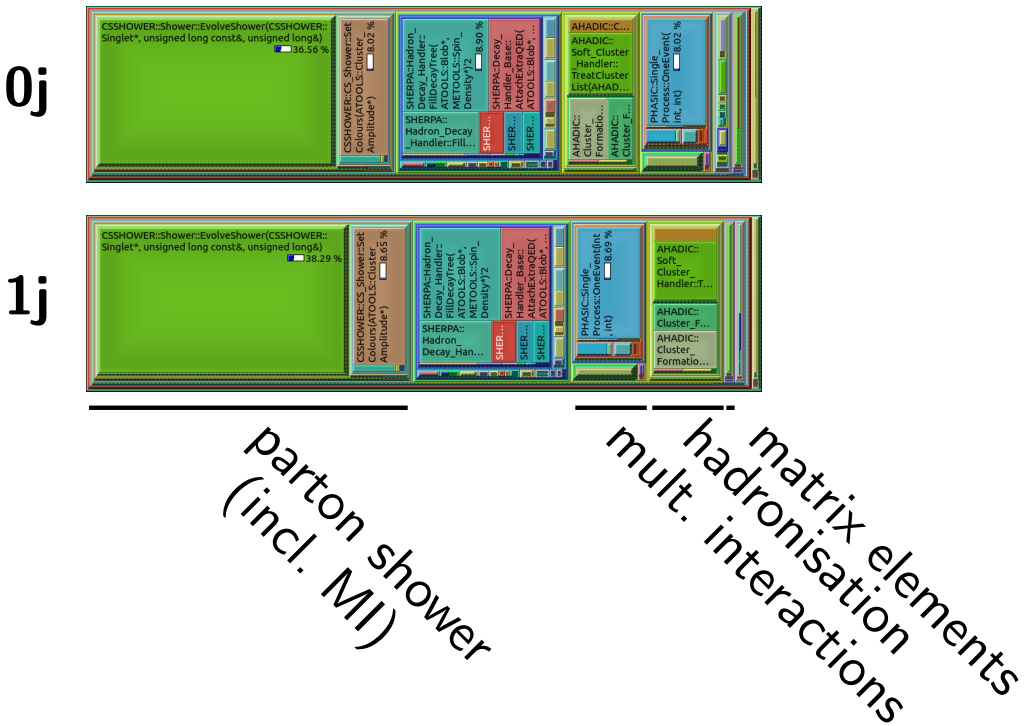
**Unweighted** W+0..≤4j@LO



# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

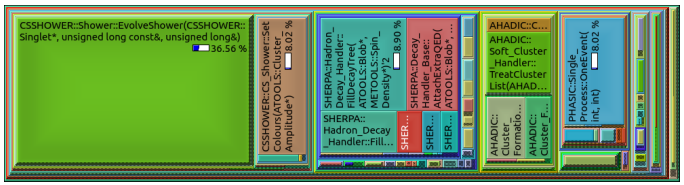


# Timing distribution: LO merging using Comix

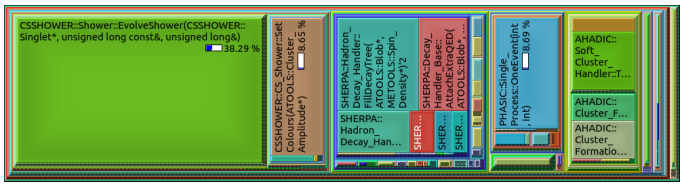
**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

**0j**




**1j**




parton shower (incl. MI)

hadron decays and QED corrs.

mult. interactions

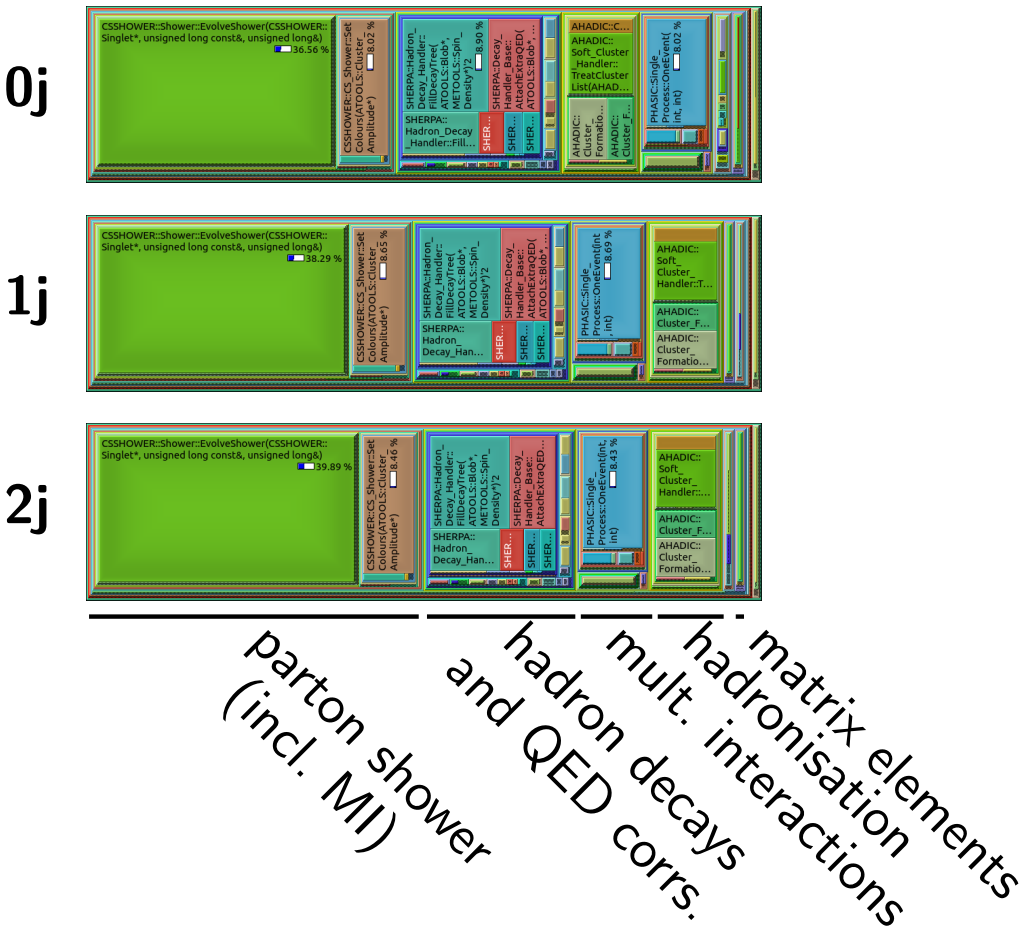
hadronisation

matrix elements

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

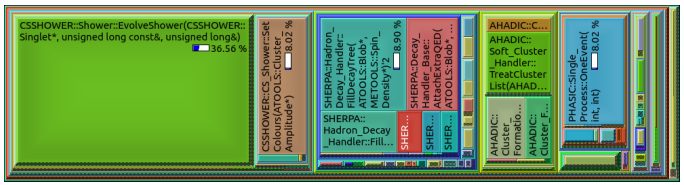


# Timing distribution: LO merging using Comix

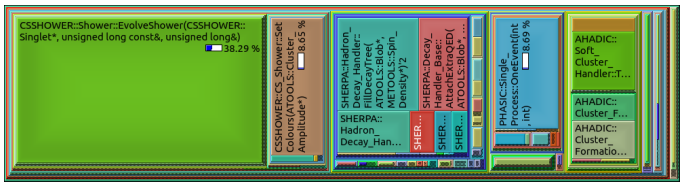
**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

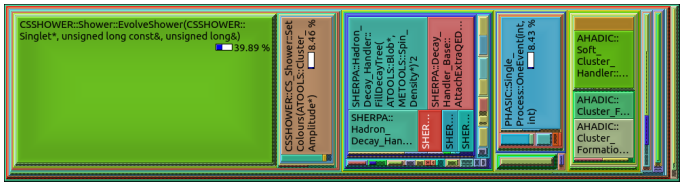
**0j**



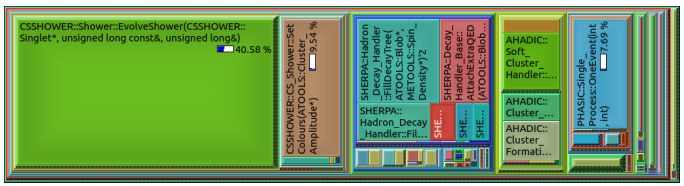

**1j**




**2j**




**3j**

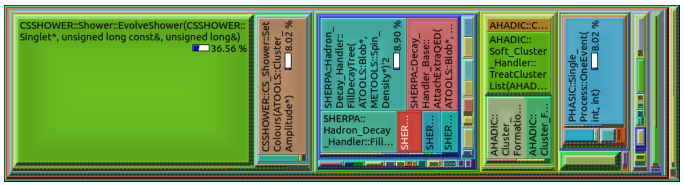


# Timing distribution: LO merging using Comix

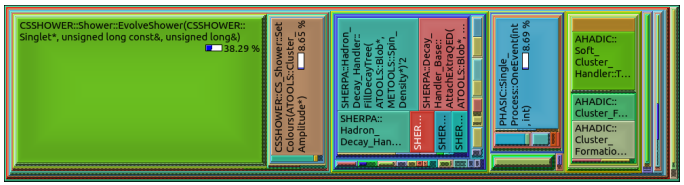
**Weighted** W+0..≤4j@LO

**Unweighted** W+0..≤4j@LO

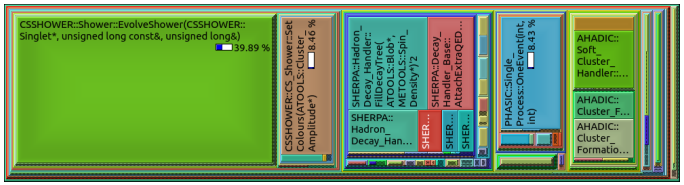
**0j**



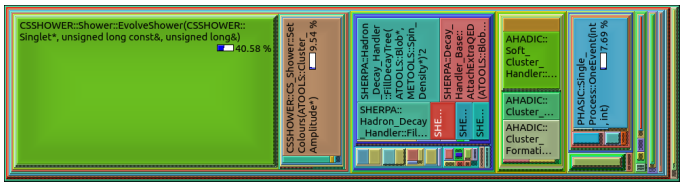

**1j**



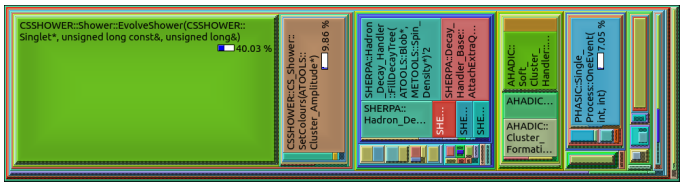

**2j**




**3j**



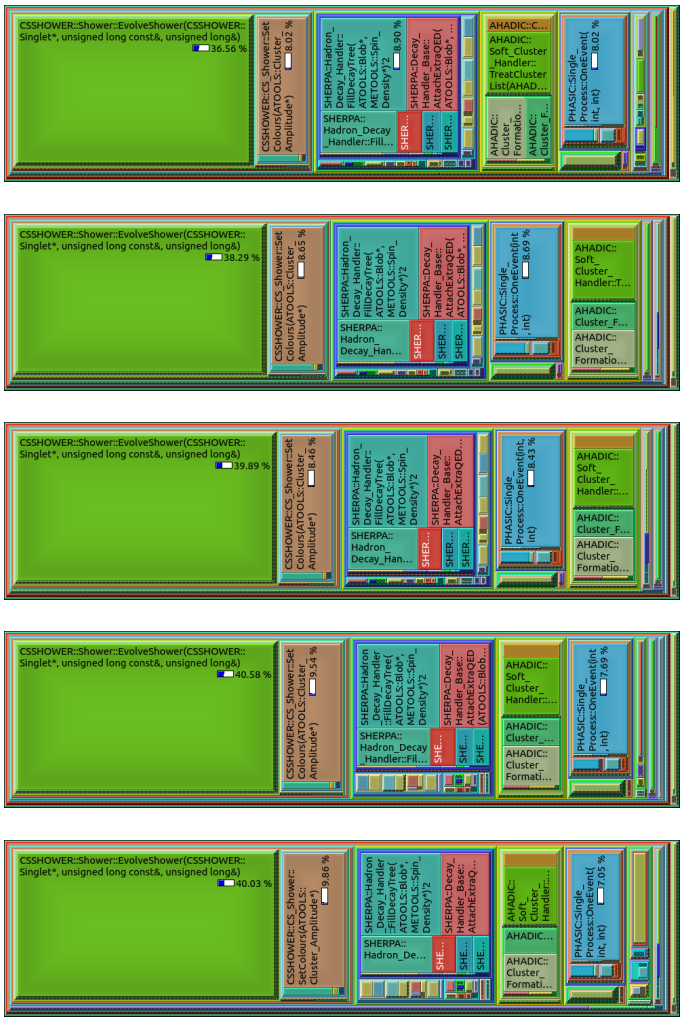

**4j**



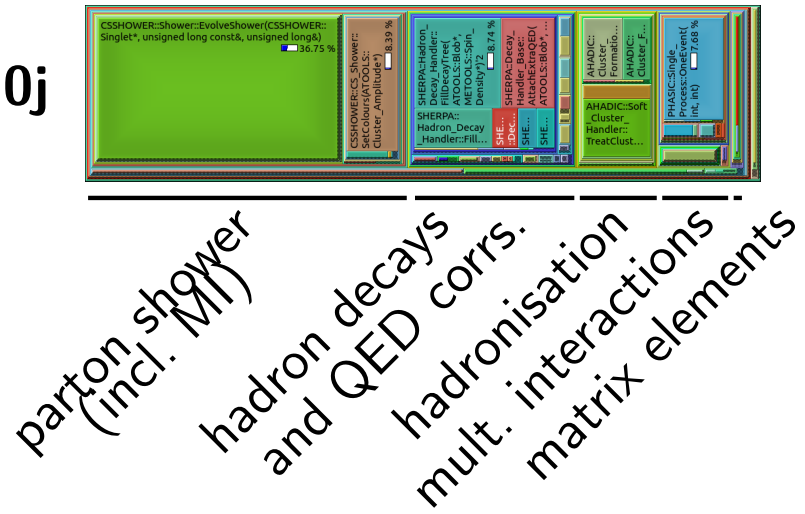

– matrix elements

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

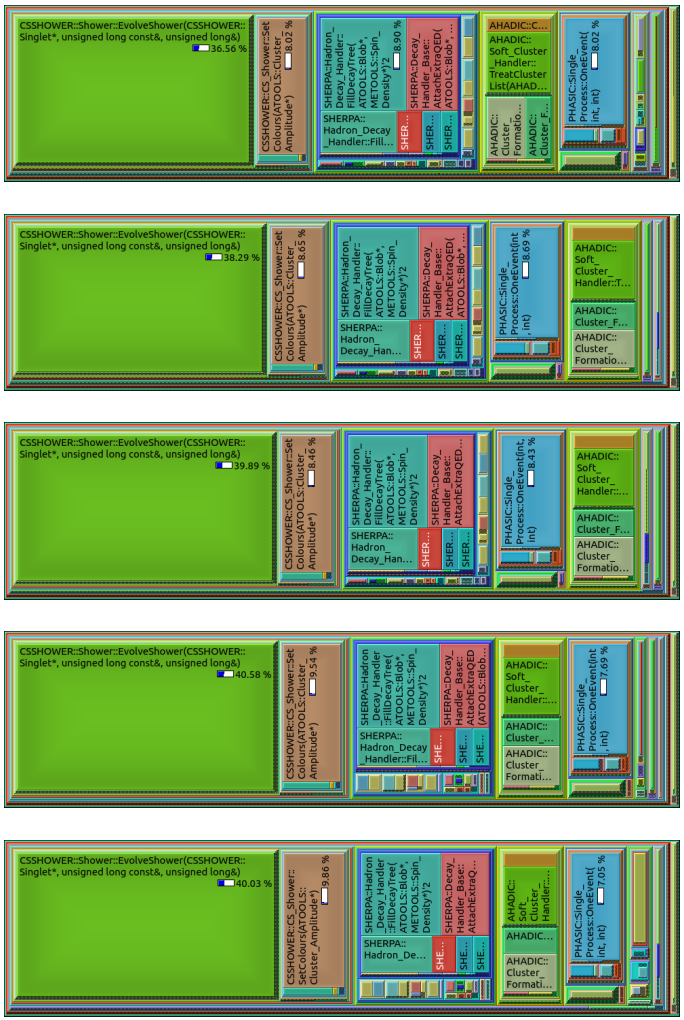



**Unweighted** W+0..≤4j@LO

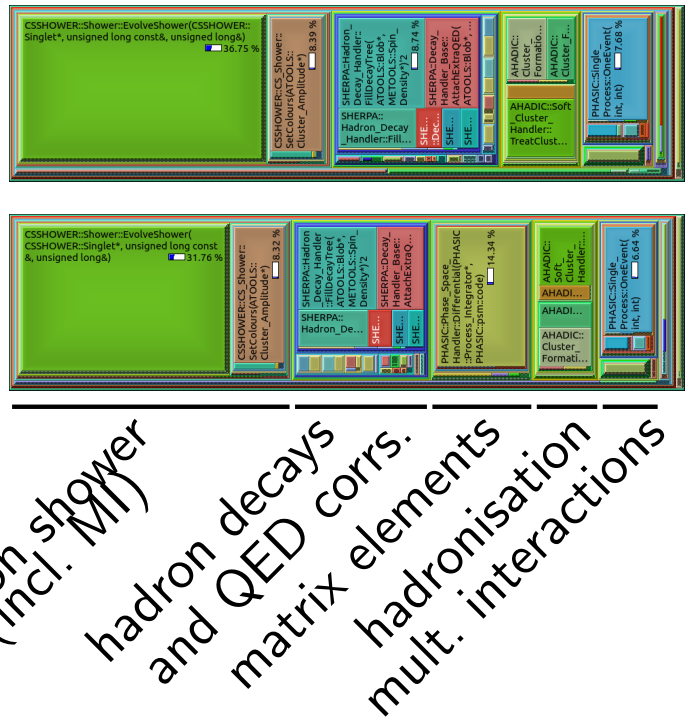


# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO




**Unweighted** W+0..≤4j@LO

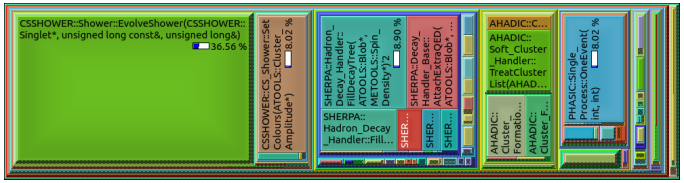


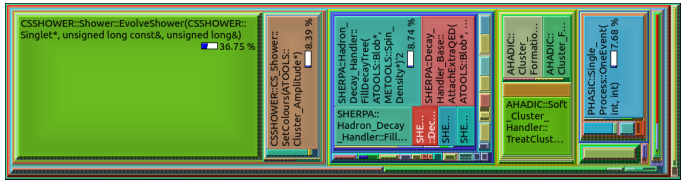
# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

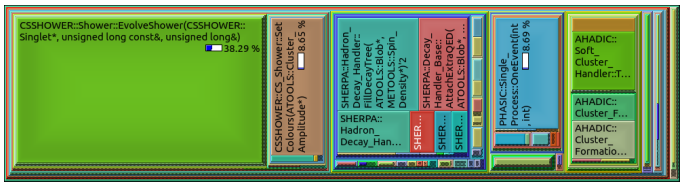
**Unweighted** W+0..≤4j@LO

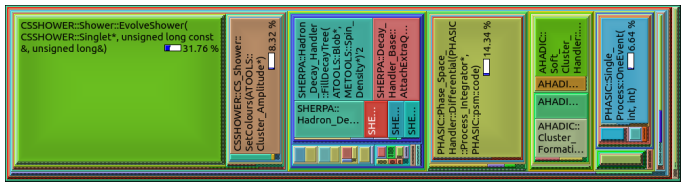
0j



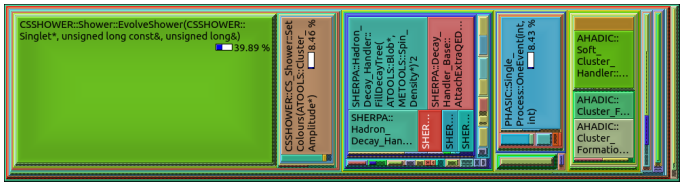



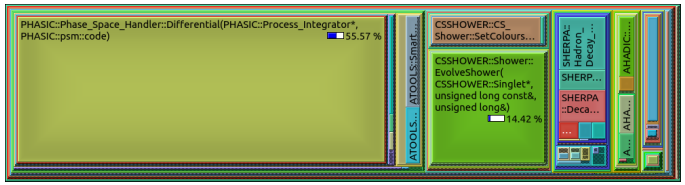

1j



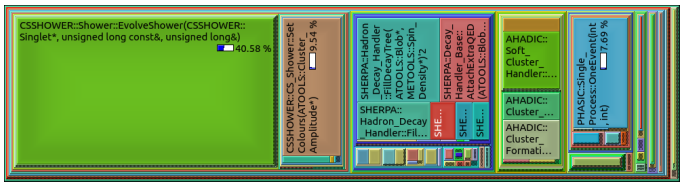




2j

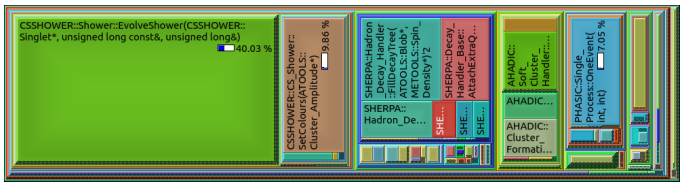



3j




4j

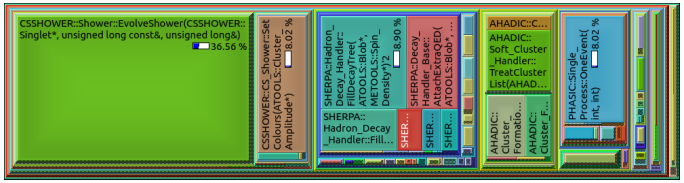



matrix elements
dominated by CKKW clustering

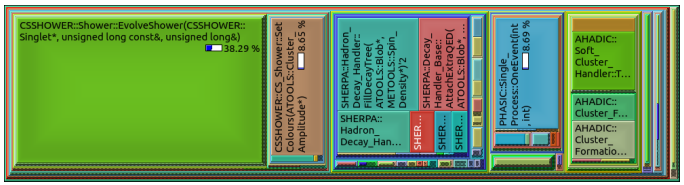
# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO

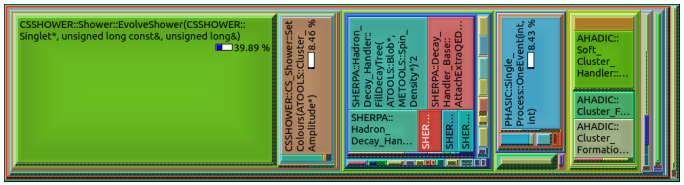
0j



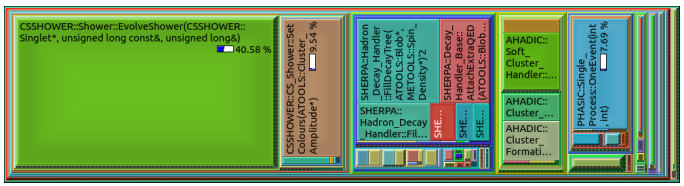

1j



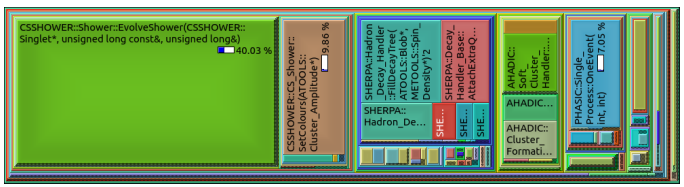

2j




3j

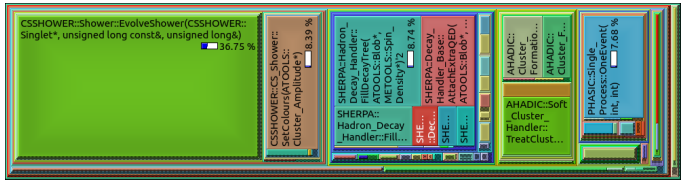



4j

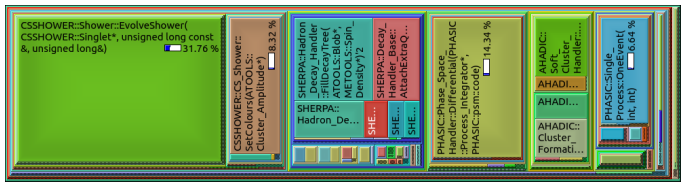



**Unweighted** W+0..≤4j@LO

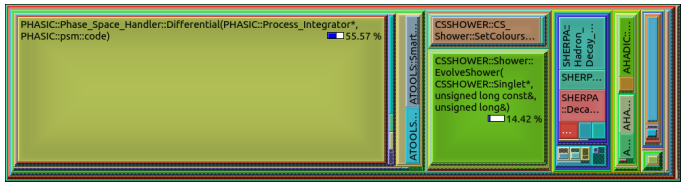
0j



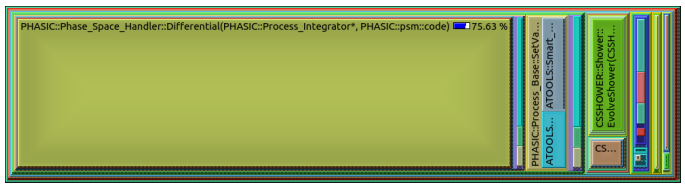

1j




2j



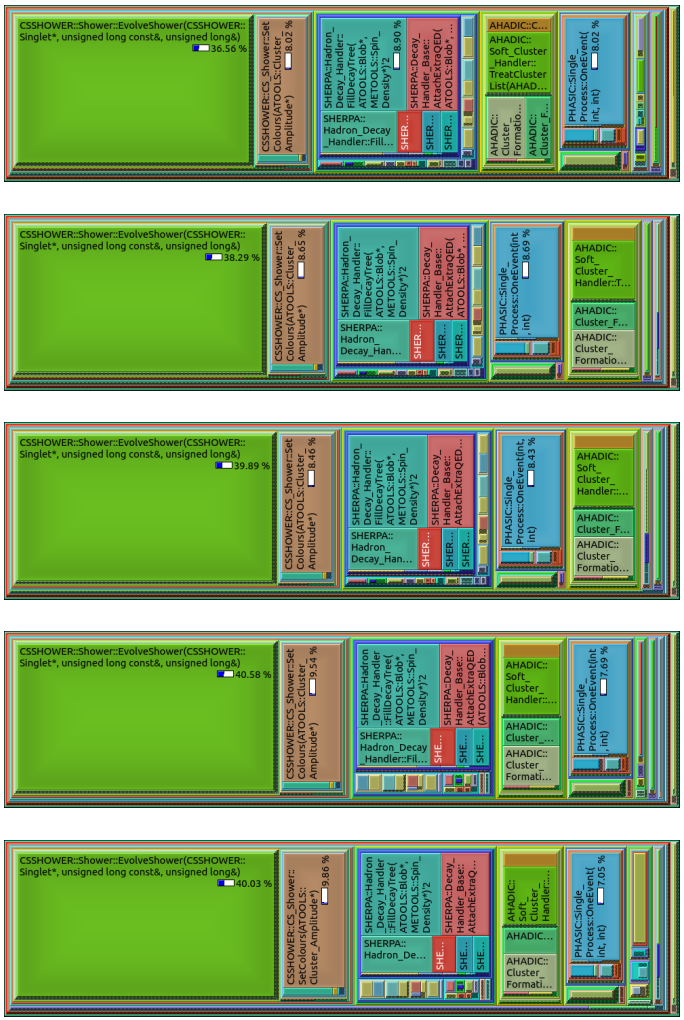

3j



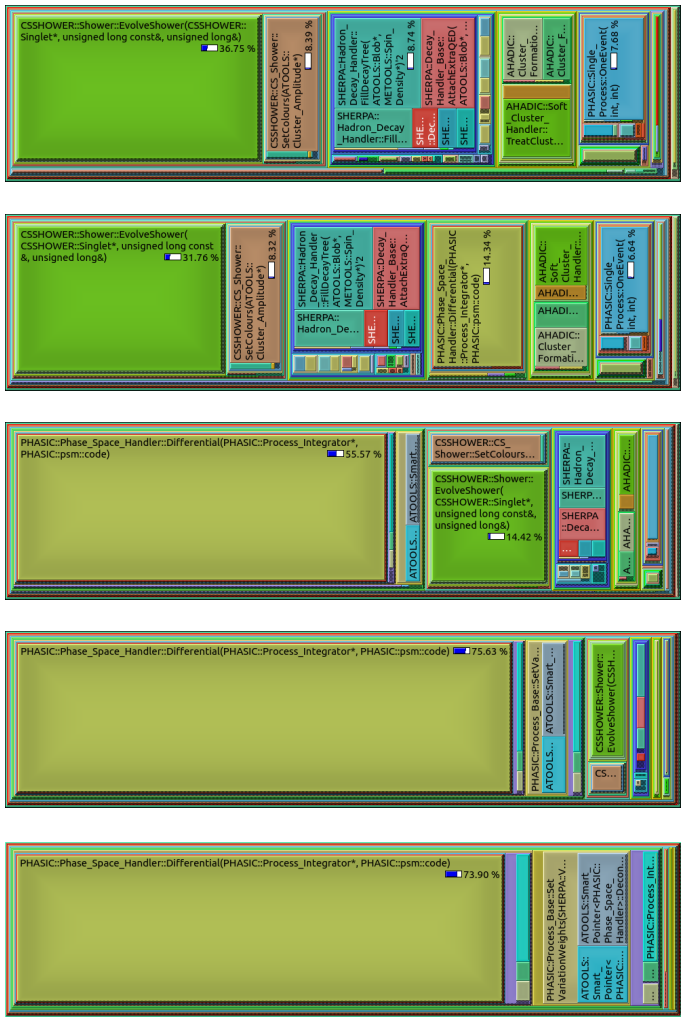

matrix elements
dominated by CKKW clustering

# Timing distribution: LO merging using Comix

**Weighted** W+0..≤4j@LO




**Unweighted** W+0..≤4j@LO



# Timing distribution: NLO merging using BlackHat

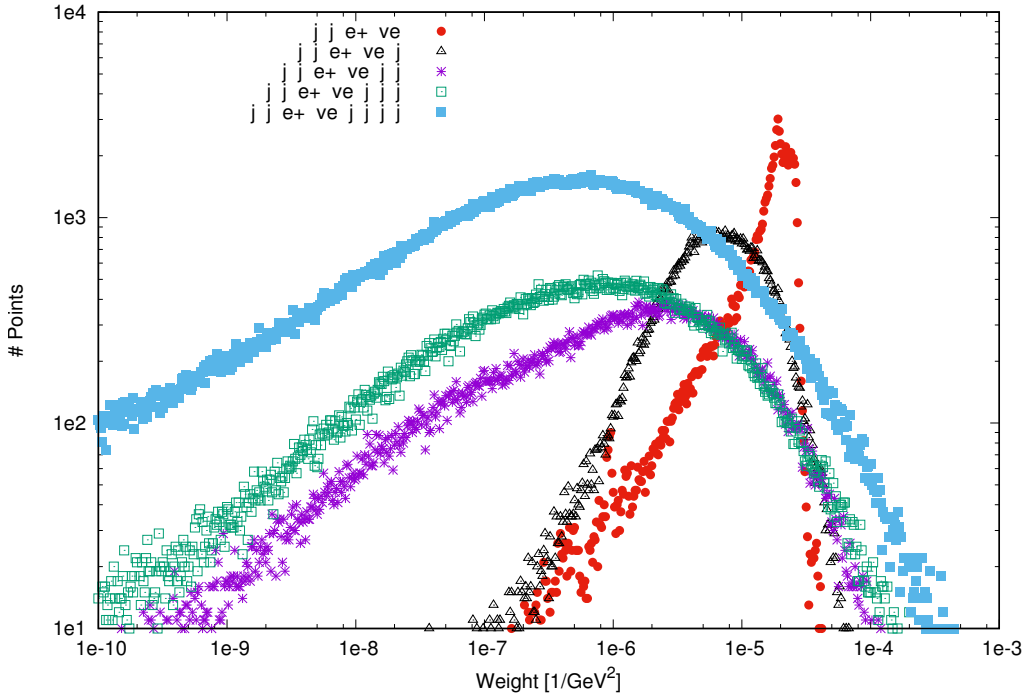
**Weighted** W+≤0..2j@NLO


**Unweighted** W+≤0..2j@NLO


NLO MEs and matching
dominated by CKKW clustering in H events,
cf. slide 12

# Weight distributions

## LO weight distribution




## Unweighting

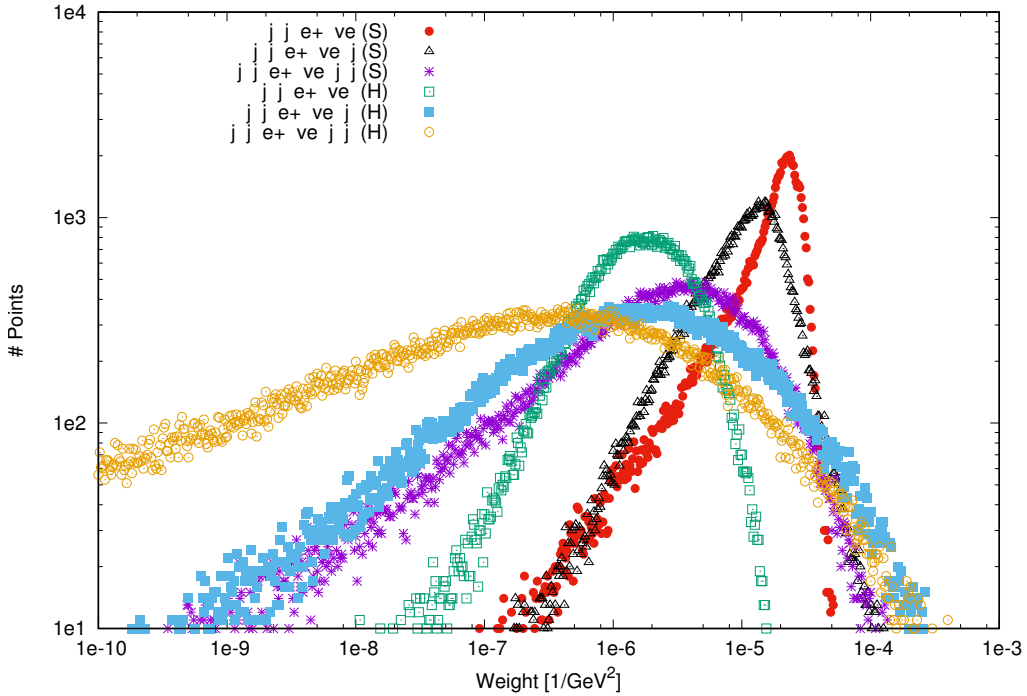
- hit-and-miss against maximum of weight distribution
- if "max" is artificially lowered (or event beyond "max" encountered)
  evts must acquire rel. weight wrt. set "max"

- higher multiplicity → wider weight distribution
  ⇒ worse unweighting efficiency
- bad unweighting efficieny at NLO dominated by H events

⇒ **performance limited by unweighting efficiency**
**need better approx. of integrand**

# Weight distributions

## NLO weight distribution




## Unweighting

- hit-and-miss against maximum of weight distribution
- if "max" is artificially lowered (or event beyond "max" encountered)
  evts must acquire rel. weight wrt. set "max"

- higher multiplicity $\rightarrow$ wider weight distribution
  $\Rightarrow$ worse unweighting efficiency
- bad unweighting efficieny at NLO dominated by H events

$\Rightarrow$ **performance limited by unweighting efficiency**
**need better approx. of integrand**

# Conclusion

**Bugfix releases of SHERPA-2.2 series**

- SHERPA-2.2.5 released Apr '18
- SHERPA-2.2.6 released shortly
  (awaiting revalidation of phase space biasing)

**Future plans:**

- NLO EW automation (fixed order)
  (approximate NLO EW corrections included since SHERPA-2.2.1)
- full $\mu_R$, $\mu_F$, $Q_{\text{cut}}$ & PDF variations in the matrix element and the parton shower for NNLO/NNLOPS in SHERPA-3.0.0
- write-out and storage of (partially un)weighted event samples
  (trade CPU costs in production with disk space)
- improvements in unweighting efficiency and neg. weight fraction

http://sherpa.hepforge.org