Using and debugging HepMC content for Tauola++ and Photos++ projects

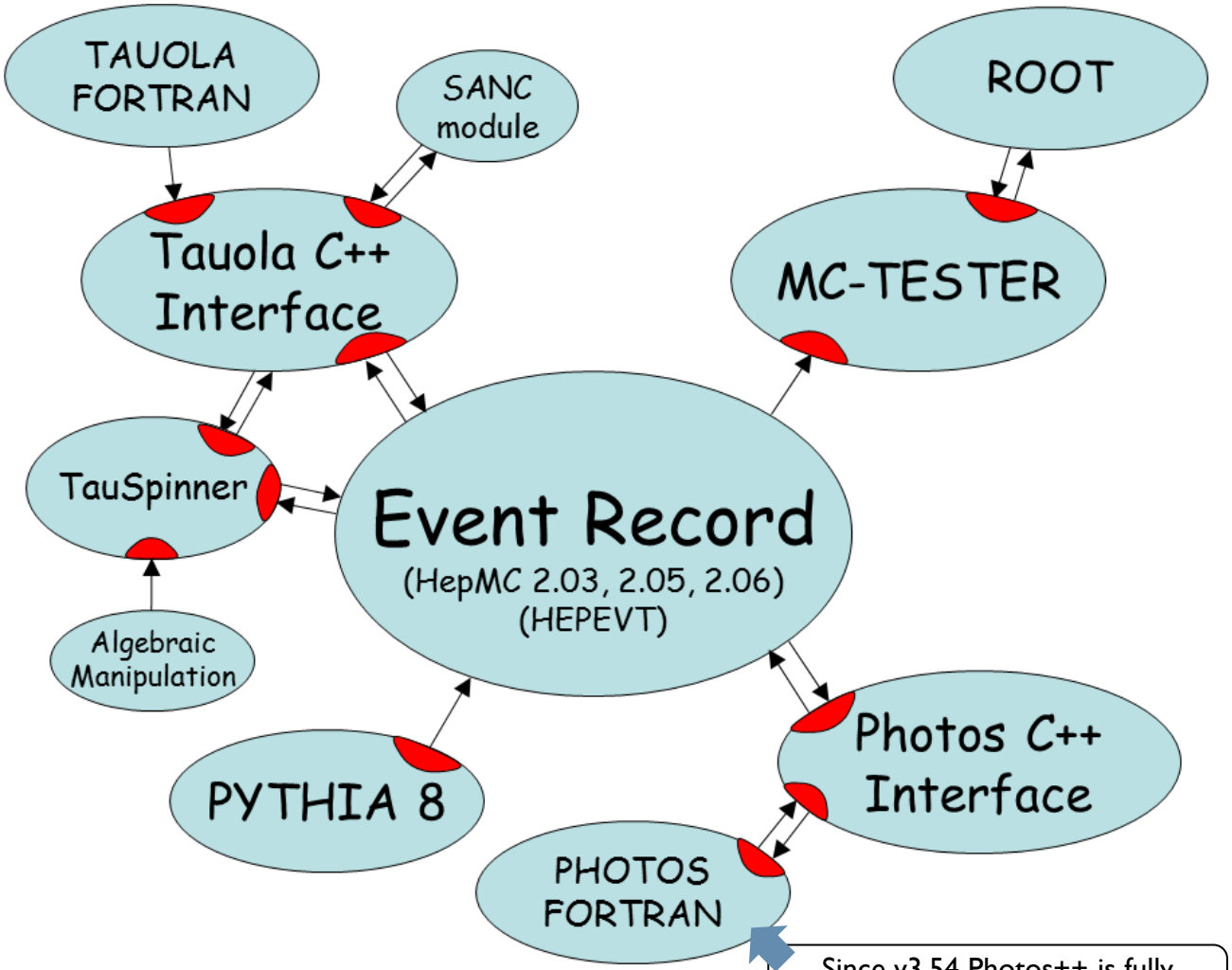
Tomasz Przedziński
Jagiellonian University, Kraków, Poland

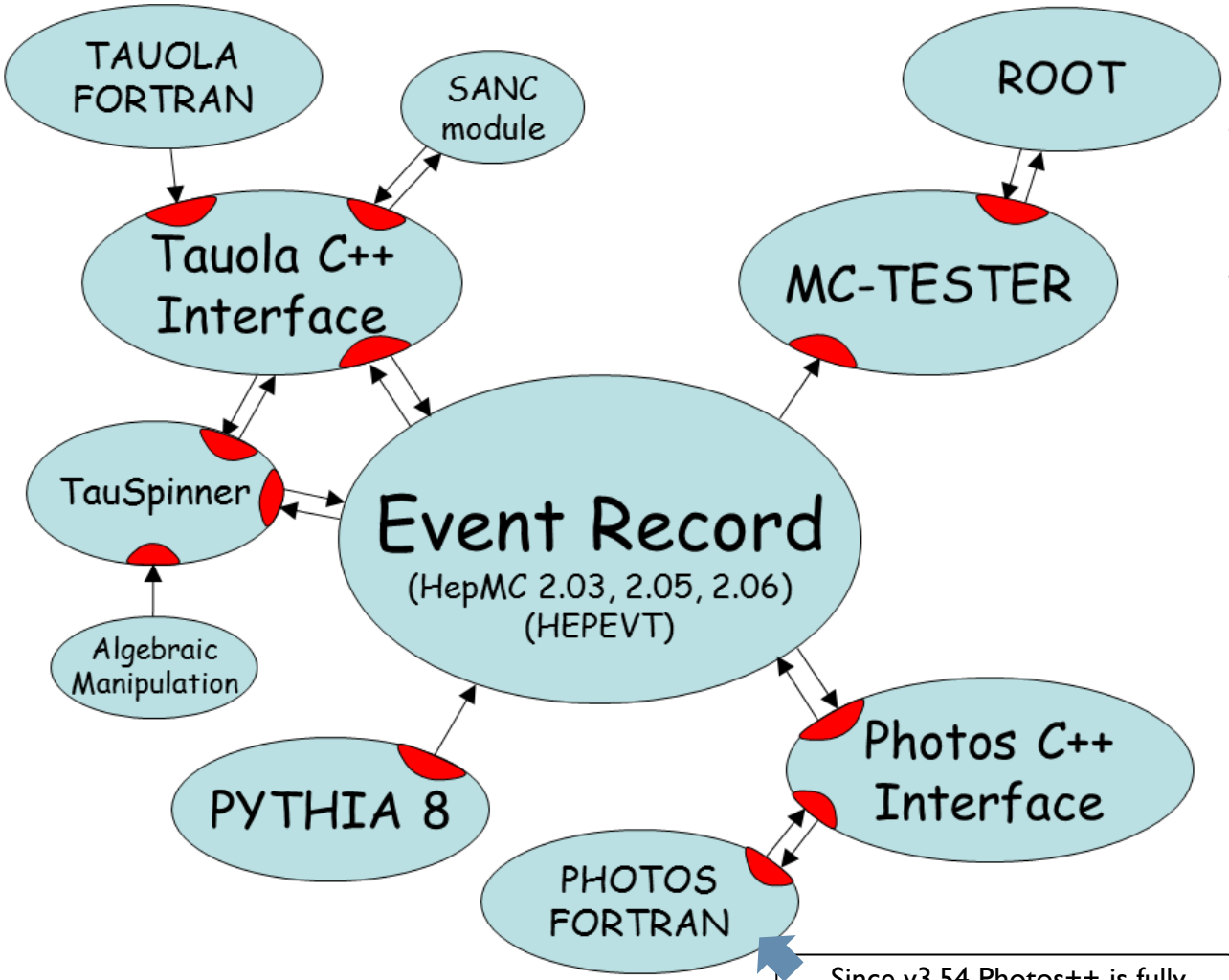
Plan:

- HepMC as an infrastructure for communication
- Use of HepMC in our projects
- Debugging HepMC content: issues and suggestions
- Questions regarding the plans for HepMC 3.0
- Summary
- Proposal: database of unusal events contents
- New functionality proposal: vertex versioning

HepMC as an infrastructure for communication

- Most simulations use an event record in a chain of analysis where next tool uses output from the previous one.
- If at any step of the analysis event record content is flawed/misunderstood, next steps may be impossible to perform. Or worse: they will be performed with faulty results.
- We often find problems with HepMC content:
  - non-standard status codes (history entries, etc.)
  - non-tree structure (decays with more than two daughters, loops)
  - 4-momentum non-conservation
  - unphysical particles in decay trees (e.g. pomerons)
  - and many more
- Knowing that such exceptions can occur (and often must, because of physics extensions) in the event record, we have to prepare special arrangement for each of such problems (which do not destroy other cases as well).
- Often we have no influence over on what events our tools are used – we cannot force experiments (and authors of other MC tools) to use any standard. When such problems occur (at unexpected moment), we have to extend our applications.
- Our main concern with HepMC is to identify problems with communication; we need tools for debugging HepMC content.

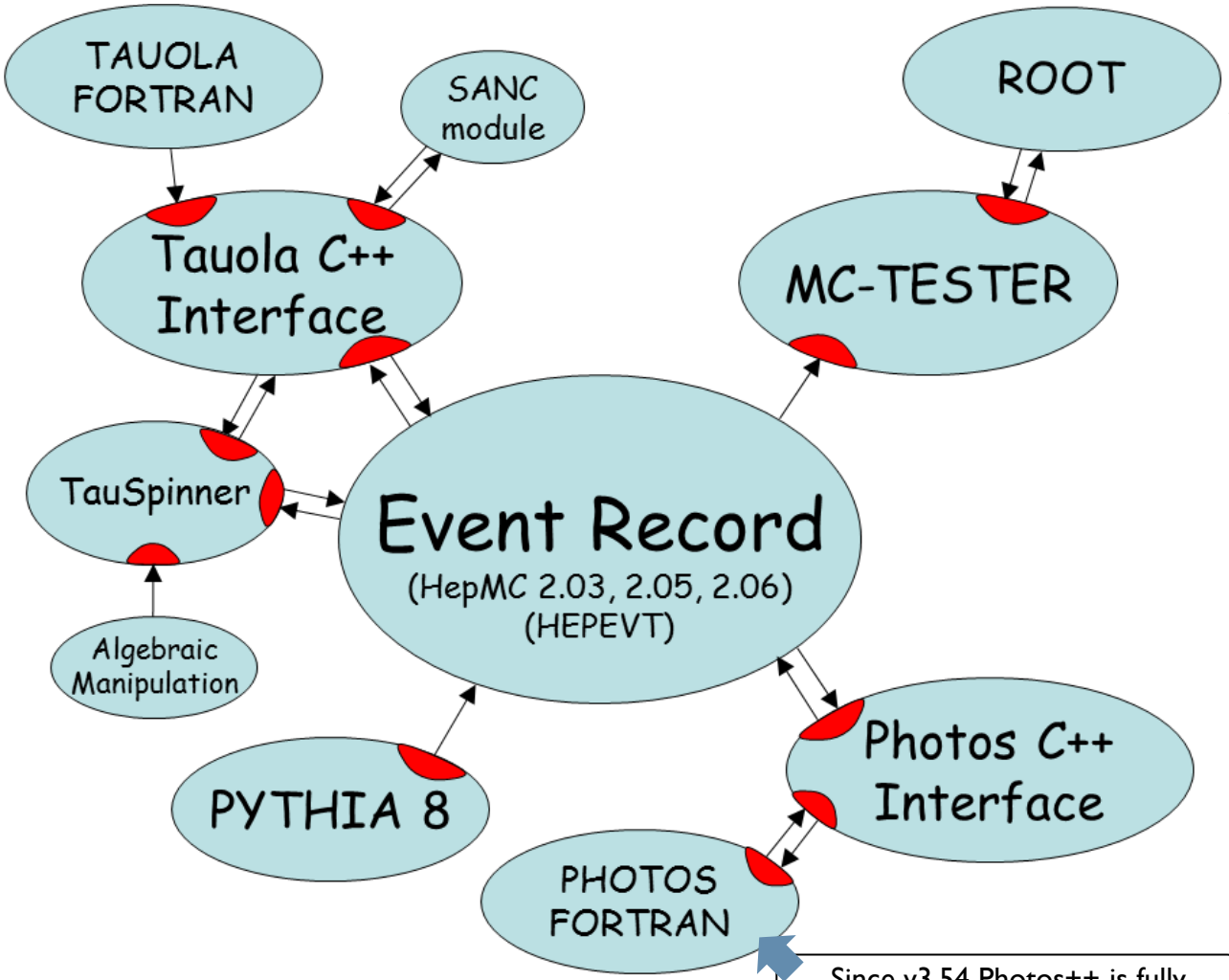

TAUOLA FORTRAN
SANC module
ROOT
Tauola C++ Interface
MC-TESTER
TauSpinner
Event Record
(HepMC 2.03, 2.05, 2.06)
(HEPEVT)
Algebraic Manipulation
PYTHIA 8
Photos C++ Interface
PHOTOS FORTRAN
Since v3.54 Photos++ is fully in C++ and has its own namespace



All these programs require that HepMC structure is a tree, which often isn't true.

If it's not, we have to reconstruct appropriate branching.

Worst-case scenario is if there's a loop in a critical place in the tree…



Tauola++ modifies branching at end of the tree (relatively easy), but uses two earlier branchings to calculate matrix elements.

Photos++ modifies single branching and (only in simple way) consecutive decays. Uses earlier branchings only in NLO matrix calculations.

Use of HepMC in our projects

- Common algorithms for these projects:
  - Find mothers, grandmothers and stable daughters of a particle
  - Find first/last version of a particle ($Z_0 \to Z_0 \to Z_0 \to \tau^+\tau^-$)
  - Add/modify particles and vertices
  - Backup existing particles ("history entries")
- Event content validation:
  - Validate tree structure (or create proper tree branching on the fly)
  - Check (or restore) 4-momentum conservation in vertex
  - Check 4-momentum conservation in the whole event
  - Check differences before/after processing by MC tool

Debugging HepMC content: Issues and suggestions

Debugging HepMC content: issues and suggestions
Printout precision

- When we get bug report from experiments, usually the first bit of feedback is the log of the program run. This log usually has several HepMC events printed out using `GenEvent::print()`. `IO_GenEvent` has `precision(int)` function, but `GenEvent::print()` uses `GenParticle::operator<<`, which always uses `ostr.precision(2)`.
- This precision, useful when looking at the tree structure, in many cases, especially when debugging MC generators, is not enough.
- Since `GenEvent::print()` is the most useful printout for debugging for us as well, one of the first "hacks" we've made years ago was to change this precision to 8 digits.

```
           581       22 +3.69841247e-01,+2.94039669e+00,+8.37444307e+00,+8.88335590e+00   1
GenVertex:     -438 ID:    0 (X,cT):0
 I: 1       578       15 -7.34781838e+00,-1.17229385e+00,+5.56212273e+01,+5.61448424e+01  23      -438
 O: 1       582       15 -6.37072096e+00,-1.01373109e+00,+4.84170416e+01,+4.88772068e+01   2      -742
Vertex:     -742 ID:    0 (X,cT)=-8.23e-02,-1.31e-02,+6.26e-01,+6.32e-01
 I: 1       582       15 -6.37072096e+00,-1.01373109e+00,+4.84170416e+01,+4.88772068e+01   2      -742
 O: 3     10001       16 -1.48669913e+00,-3.79751911e-01,+1.38697502e+01,+1.39543741e+01   1
          10002       13 -3.44810969e+00,-2.21272813e-01,+2.76722167e+01,+2.78872939e+01   1
          10003      -14 -1.43594786e+00,-4.12712039e-01,+6.87534667e+00,+7.03581334e+00   1
Vertex:     -743 ID:    0 (X,cT)=+2.28e-01,-6.60e-02,+6.12e-01,+6.62e-01
 I: 1       580      -15 +4.60252655e+00,-1.33490511e+00,+1.23751204e+01,+1.33890448e+01   2      -743
 O: 3     10004      -16 +2.46164145e+00,-1.71538384e-01,+7.55554335e+00,+7.94829789e+00   1
          10005      -11 +1.19530419e+00,-1.01178470e+00,+3.05986075e+00,+3.43732575e+00   1
```

Debugging HepMC content: issues and suggestions
Printout precision

- This solution works for us, but it still does not help us analyze user feedback.
- Having `GenEvent::print(int precision=2)` would be useful (and backward-compatible with the default use of this function) even if not exactly straightforward to do because of the use of `GenParticle::operator<<`
- We could then encourage experiments to send us feedback with `GenEvent::print(8)` while they could still use `GenEvent::print()` in their logs to conserve disk space.

Debugging HepMC content: issues and suggestions
Momentum conservation checks

- Some time ago I wrote a ticket about misleading name of the function `GenVertex::check_momentum_conservation()`. There exists `GenParticle.momentum()` which is of `FourVector` class, but `check_momentum_conservation()` checks only 3-momentum conservation.
- For theoretical physicist, word "momentum" in almost any context, means "4-momentum". This is worth mentioning, because this mismatch in convention made us confused and made us redo a lot of tests that were relying on this method to check 4-momentum conservation.
- It has been resolved in HepMC by indicating in the comment of this function, that it's a 3-momentum check. However, we sill only use 4-momentum conservation checks.
- HepMC does not have a function to do it; we had to write our own.

Debugging HepMC content: issues and suggestions
Vertex and event momentum conservation checks

- In terms of vertex 4-momentum conservation check: it would be useful if `GenVertex` could have a printout option that shows in/out vertex sums and difference between these sums.
- For the most-precise algorithms the result of the momentum conservation check is not enough. We have to know exactly which components of the 4-vector are not conserved.

- In terms of event 4-momentum conservation check: a function which checks difference between all stable products and the incoming beams would be useful as well.
- This is a typical test for MC generators (from HEPEVT times) often also performed before detector simulation. This test serves as the most-generic indicator of a bug and we often use it to pick unusual events that may not be correctly handled by our tools.

Debugging HepMC content: issues and suggestions
Small I/O issues

- Cutting out a sub-tree of an event is one of the common things we do (to save space) when writing events to disk.
- However, when doing so (saving, for example: $X, Y \rightarrow Z_0 \rightarrow \tau^+\tau^-$) we've found out that saving `production_vertex()` and `end_vertex()` of Z_0 skips particles X,Y. Creating new vertex with X,Y as outgoing particles is needed to save them to output file.
  - I don't know if it's an intended feature (I couldn't find it in documentation). It might be a bug because in such cases one would expect that full content of `GenEvent` (displayed by `GenEvent::print()`) is written to a file.
  - On a side note: declaring X and Y as incoming beams does not help.

- When merging multiple HepMC files into one, we find headers/footers to be problematic. However, it's a standard to include them in `IO_GenEvent` file and that is ok.
  - For this purpose, we wrote `omit_header` and `omit_footer` for `IO_GenEvent`.

Debugging HepMC content: issues and suggestions
Database of debug events

- In collaboration with theorists and experimentalists, we constantly find many particular cases of formally faulty HepMC events to which we have to adapt our algorithms.
- This is a burden to our projects but we understand that in many cases it cannot be avoided due to the physics content outside of HepMC design.
- At present it is routine activity of ours to find and debug such events (see previous slides).
- From this, we have created a set (>15) of debugging HepMC files
  - We run every new release of our code against these files to check if these events are still processed correctly after new patches.
  - We rely on 1 - 10 events tweaking our test programs to recreate the bug.
  - Right now, we keep these events for our internal use. Maybe this could change?

Debugging HepMC content: issues and suggestions
Database of debug events

- As many of these events have non-trivial structures, maybe a library of such events could be useful as a part of HepMC content debugging?
  - such events would be associated with the HepMC project, not just specific MC tool, making it available to others
- This would allow MC generators developers to benefit from the experience of people who already encountered (and have noticed, which in many cases is the main problem) such events.
  - developers could test their software against such events and prepare a fix in case they are not handled correctly
  - this would prevent the bugs from occurring in later steps of development or deployment of the project, which might be crucial to the project maintenance
- Since this might improve communication between experiments, MC tools and MC generators, experiments may be willing to agree to make such samples of events public.

Summary

- Our algorithms for fixing/debugging input:
  - output precision hack
  - tree structure validation
  - filters for unsupported/unphysical particles and self-decays
  - in/out vertex sum; vertex 4-momentum conservation check
  - 4-momentum event conservation checks (used also after processing by our tool)
  - several algorithms to correct vertex 4-momentum non-conservation
  - using PDG mass or `generated_mass` to correct particle 4-momentum
  - database of peculiar events that we use (constantly) to verify our changes
- Our algorithms regarding HepMC content:
  - unit fix (parts of the input event can have different unit than others! Momentum non-conservation checks easily pick it up, but we still need to fix it to continue)
  - creating history entries (currently: copying particles with `status=3`)
  - redecaying particles (currently: storing new decays in separate HepMC events)
  - store sub-trees for further analysis (to conserve disk space)
  - `IO_GenEvent::omit_header`, `IO_GenEvent::omit_footer`

Questions

- Are there any plans of introducing `HepMC::IO_GenEvent` compression?
  - We often find .gz files to be 3x smaller than plaintext files generated by `HepMC::IO_GenEvent`. However, using gzip in this way has no benefit because we still need disk space for compression/decompression.
  - A native .gz support for HepMC I/O could be beneficial.
- Do you have any feedback on how often `HepMC::IO_GenEvent` files are used or do experiments / MC tools developers prefer other HepMC output standards?
- Since HepMC is used as means of communication between MC programs, are there any plans for introducing any content validation?
  - status code standard
  - charge conservation
  - flavour conservation
  - Input from Les Houches Accords format compliance; check that all information is passed

Bonus slides: New functionality useful for MC generators developers

New functionality proposal
Vertex versioning

- We often compare events before/after processing by our tools – both to verify the changes and to keep a history of modifications.
- ATLAS, for example, uses "history entries" (particles with `status=3`) to save information about the original particles that are later modified by Photos++.
- Such history entries are useful to compare events before/after modification, but this solution works only if vertices remain unchanged.
- My proposition: vertex (and particles?) versioning:
  - By default, all vertices have "version 1". Creating a copy of a vertex can introduce version change, i.e. two vertices of the same number exist in the event but with different version number. User can access (or check existence of) different versions of a vertex through proper accessors.
  - When printing out, saving to file or iterating over particles either all or only one selected version is used. Saving just one version introduces backward-compatibility and negates any additional memory use when dumping events to files.

New functionality proposal
Vertex versioning

- This proposal solves the problem of history entries and allows to have a history of changes of vertices as well (number of vertices in each version can vary).
- Since many MC tools are used in chain, this solution allows to save the result of processing by each tool as a new version. A history of changes is immediately available and versions can be easily compared with each other.
- While this can be done by copying `GenEvent` at each step, this is not a memory-efficient or disk space-efficient solution. Versioning would reduce the memory used by saving only vertices that have changed.
- For experiments that do not need it, this solution would not introduce any disk space cost (if needed, information about version does not have to be saved and versioning can be dropped when saving final/selected version to a file)
- `short int` is enough to save version number. Again, when saving to file, "version 1" vertices don't have to include version number at all, which is both backward-compatible and does not increase disk space.