JuliaHEP 2025

HepMC3.jl - Julia Interfaces to HepMC3 Event Record Library

https://github.com/JuliaHEP/HepMC3.jl

Divyansh-Goyal
Guru-Gobind-Singh-Indraprastha-University
28 July 2025

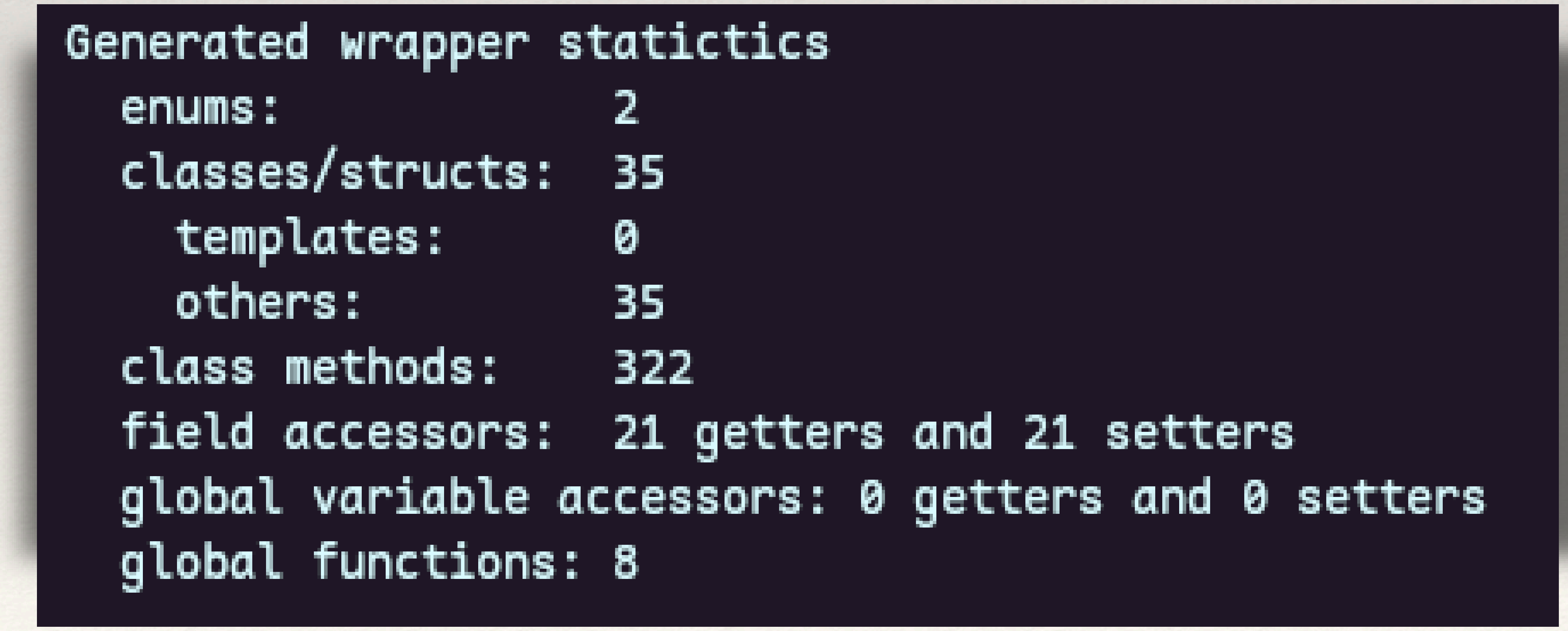
Mentor: Graeme-A-Stewart

Motivation → Why HepMC3 Fits

- **Julia is a priori a good programming language candidate for HEP**
- It combines **high-level expressibility** for scientific computational problems together with **high-performance** execution, avoiding the two language problem
- One essential aspect is to improve **interoperability** with existing C++ libraries in HEP
- HepMC3 files are a defacto standard for event generator outputs. Julia support to read these files would be a valuable addition to the JuliaHEP toolbox.

Julia wrappers to HepMC3

- Similarly to Python, to call C++ from Julia you need to write (better generate) wrappers for each method you want to offer to Julia
- Using the **CxxWrap.jl** package
  - The user needs to write a small code (in C++) to wrap each class and method (similar to pybind11 or Boost.Python)
  - The package **WrapIt** developed by Philippe Gras makes use of LLVM libraries to generate the wrappers automatically 😀



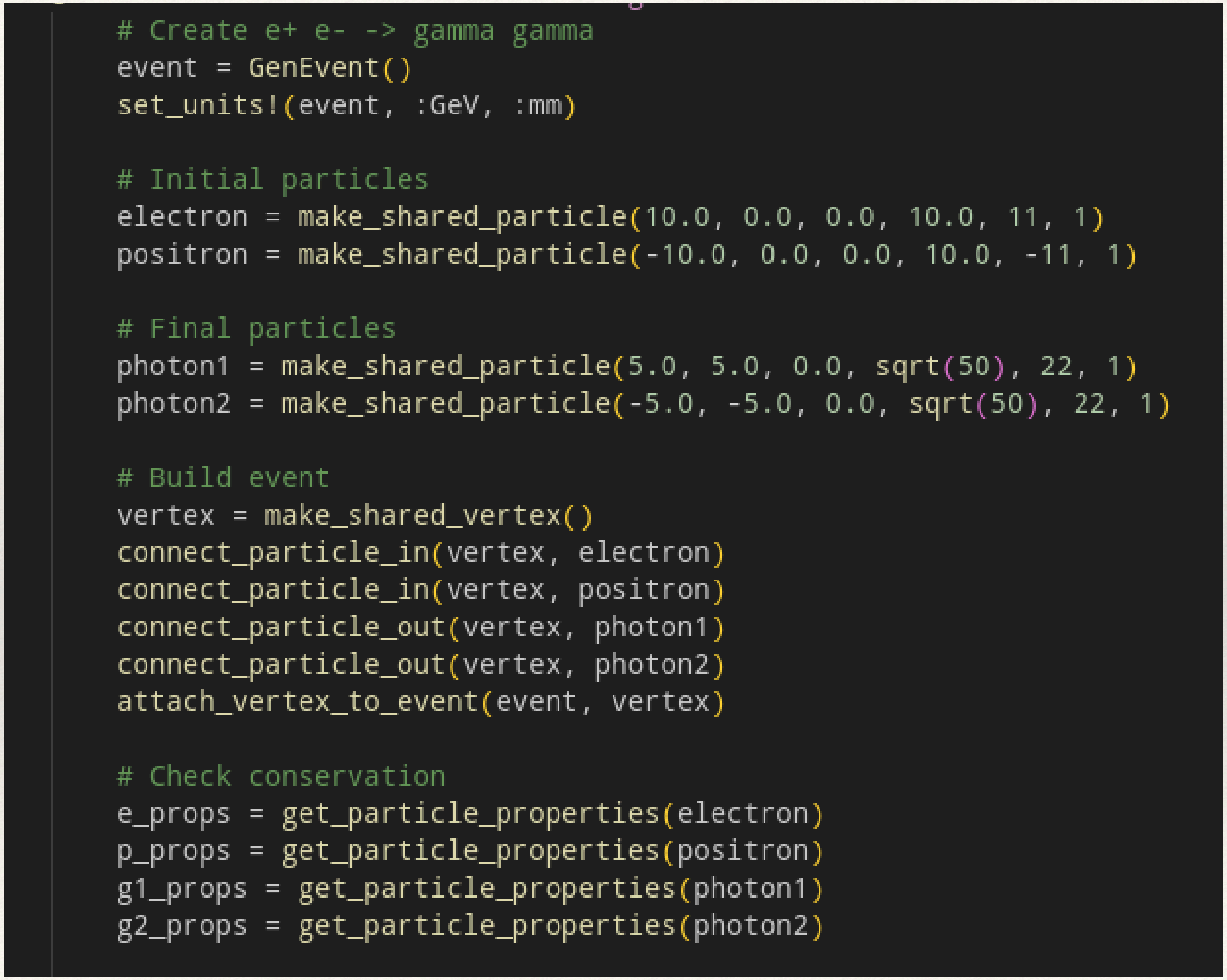
```
Generated wrapper statictics
  enums:             2
  classes/structs:   35
    templates:       0
    others:          35
  class methods:     322
  field accessors:   21 getters and 21 setters
  global variable accessors: 0 getters and 0 setters
  global functions: 8
```

HepMC3.jl: Basic Interface

- ❖ All HepMC3 functions maintain descriptive names - make_shared_particle(), set_units!(), get_particle_properties() providing clear, Julia-style API, also easy for someone familiar with HepMC3 to use

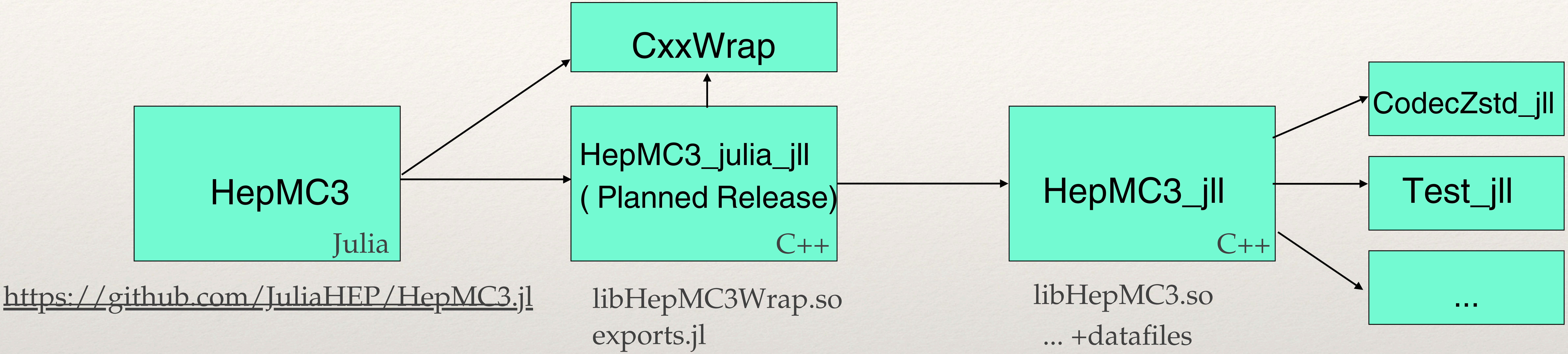
- ❖ Direct C++ object manipulation through shared pointers - Functions return Ptr{Nothing} handles for efficient memory management and performance

- ❖ Sometimes native Julia types require extraction - Particle properties accessed via get_particle_properties(particle_ptr) returning named tuples with physics data



```
# Create e+ e- -> gamma gamma
event = GenEvent()
set_units!(event, :GeV, :mm)

# Initial particles
electron = make_shared_particle(10.0, 0.0, 0.0, 10.0, 11, 1)
positron = make_shared_particle(-10.0, 0.0, 0.0, 10.0, -11, 1)

# Final particles
photon1 = make_shared_particle(5.0, 5.0, 0.0, sqrt(50), 22, 1)
photon2 = make_shared_particle(-5.0, -5.0, 0.0, sqrt(50), 22, 1)

# Build event
vertex = make_shared_vertex()
connect_particle_in(vertex, electron)
connect_particle_in(vertex, positron)
connect_particle_out(vertex, photon1)
connect_particle_out(vertex, photon2)
attach_vertex_to_event(event, vertex)

# Check conservation
e_props = get_particle_properties(electron)
p_props = get_particle_properties(positron)
g1_props = get_particle_properties(photon1)
g2_props = get_particle_properties(photon2)
```

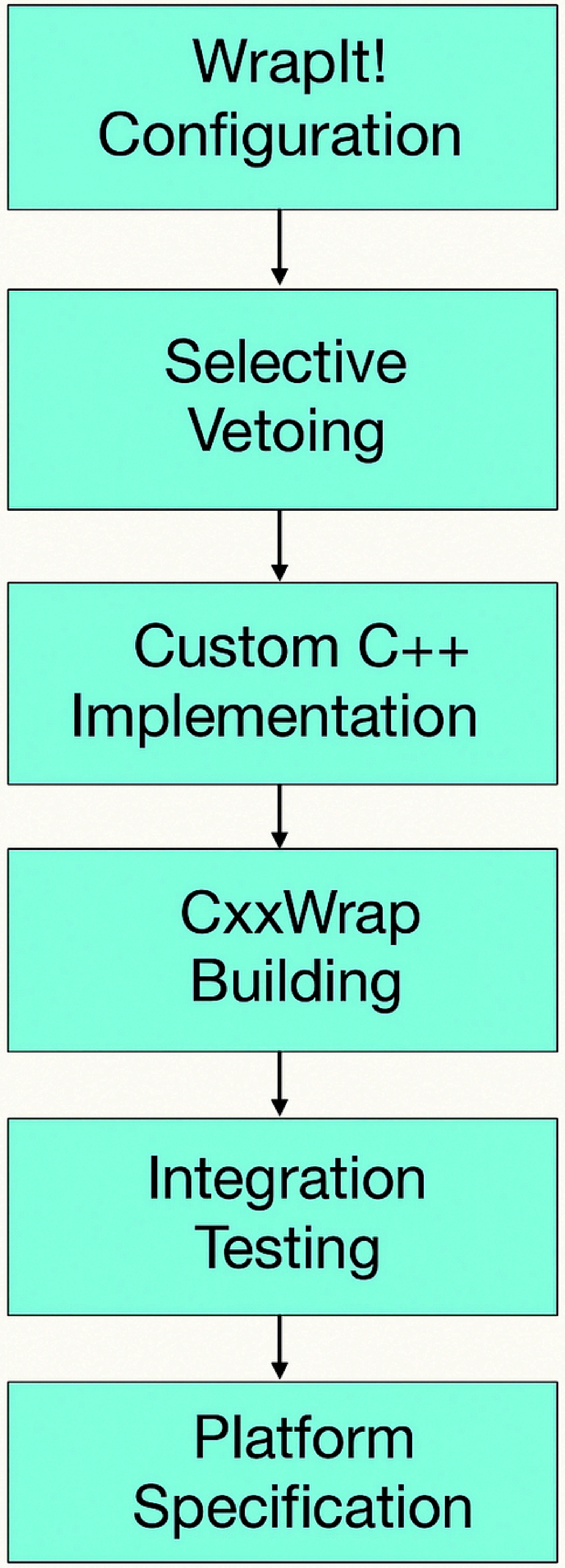
Package Structure



- The package HepMC3.jl is a Julia package (platform dependent with custom C++ bindings)
- The binary libraries (platform dependent) for HepMC3 and the custom wrapper library are built using manual C++ implementation and linked with the HepMC3_jll package binaries from Julia infrastructure (GitHub)

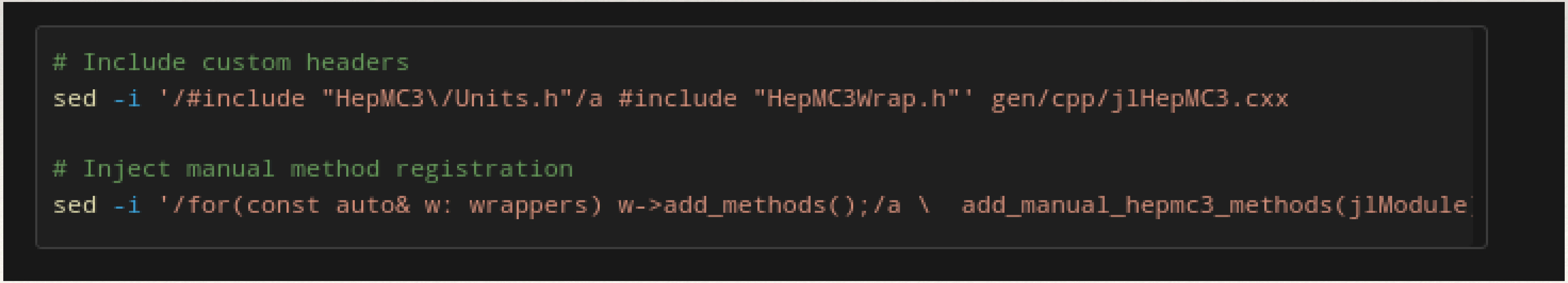
HepMC3.jl: Implementation Workflow

- **Veto File Creation** - Exclude problematic C++ constructs (std::shared_ptr, std::vector<GenParticlePtr>, complex containers) from auto-generation
- **WrapIt! Configuration** - Auto-generate Julia bindings from HepMC3 headers with vetoed functions excluded (GenEvent, GenParticle, FourVector classes)
- **Custom C++ Implementation** - Write dual-type wrapper functions (particles_size vs particles_size_raw) for shared_ptr/raw pointer compatibility
- **Library Rebuilding** - Compile hybrid libHepMC3Wrap.so with automatic CxxWrap binding generation for new manual C++ functions
- **Julia Interface Layer** - Create method dispatch routing GenEvent objects to _raw functions, Ptr{Nothing} to shared_ptr functions
- **Integration Testing** - Validate JetReconstruction.jl pipeline (100 events) and comprehensive test suite (156 tests passing)
- **Platform Specification** - Verified on x86-64 Linux systems with Ubuntu/Arch compatibility



HepMC3.jl: Problems Faced & Solutions

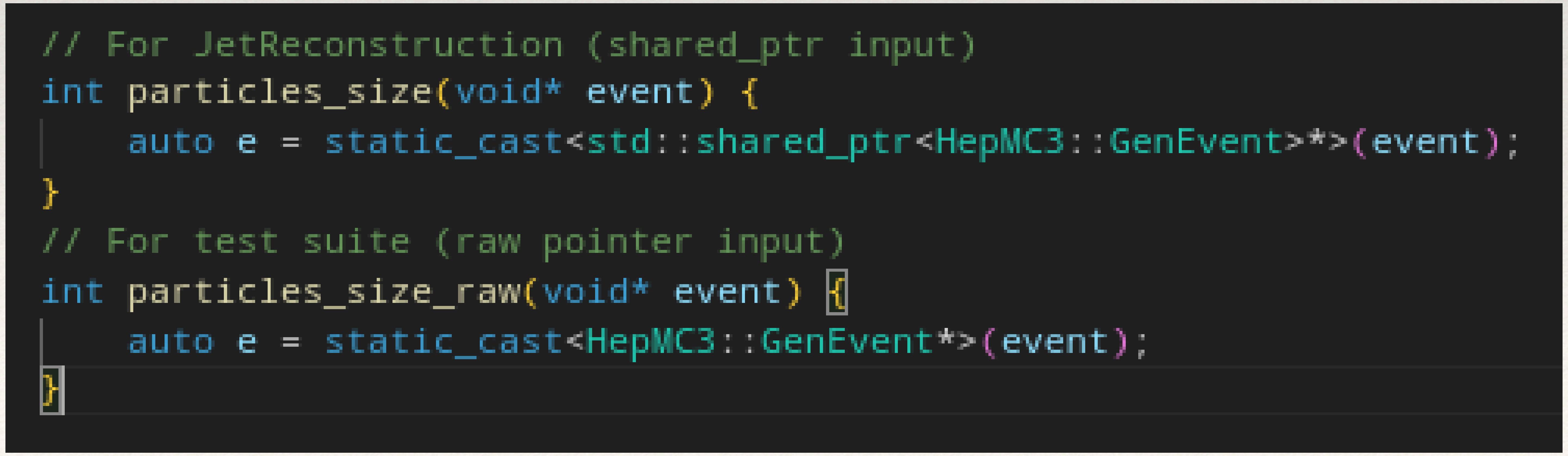
- **Problem 1: Single JLCXX Module Limitation**
  - **Challenge** - CxxWrap libraries can only define one jlcxx module per shared library, preventing separate manual wrapper modules
  - **WrapIt! Limitation** - Auto-generated code creates the primary jlcxx module, blocking additional manual module definitions
  - **Solution Strategy** - Let WrapIt! generate the main jlcxx module, then inject custom functions via add_manual_hepmc3_methods()
  - **Implementation** - Add custom C++ wrapper functions to existing module through library source patching
  - **Rebuild the library** - Now, rebuild the library with the applied patch.



```
# Include custom headers
sed -i '/#include "HepMC3\/Units.h"/a #include "HepMC3Wrap.h"' gen/cpp/jlHepMC3.cxx

# Inject manual method registration
sed -i '/for(const auto& w: wrappers) w->add_methods();/a \  add_manual_hepmc3_methods(jlModule
```

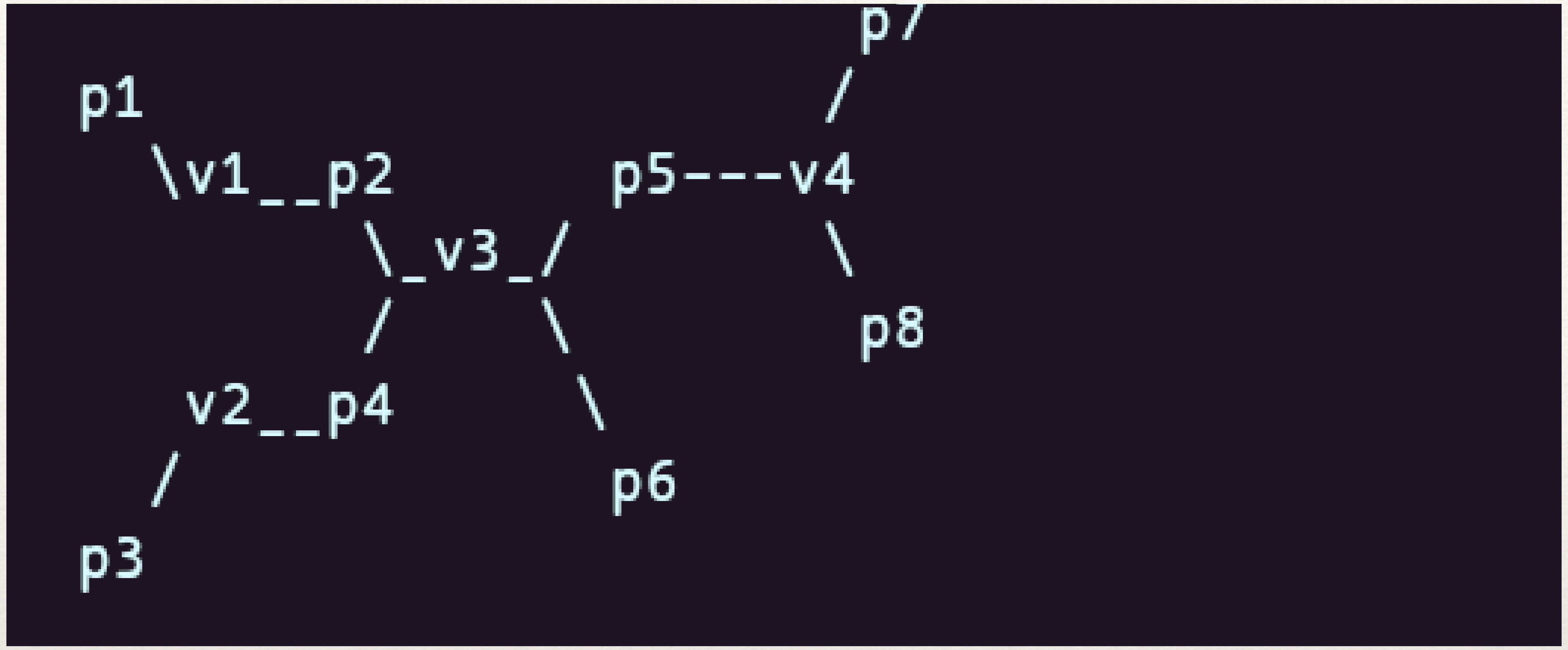
HepMC3.jl: Problems Faced & Solutions

- **Problem 2: Dual Pointer Type Compatibility Crisis**
  - **Challenge** - JetReconstruction.jl pipeline expected shared_ptr<GenEvent>* input while test suite used GenEvent.cpp_object (raw GenEvent* pointers)
  - **Memory Corruption** - Type casting mismatches caused crashes (std::bad_alloc), garbage return values (particles_size() = -1), and segmentation faults
  - **Incompatible Use Cases** - Single function implementations couldn't serve both:
    - File reader → Ptr{Nothing} → shared_ptr functions (JetReconstruction path)
    - GenEvent objects → .cpp_object → raw pointer functions (test path)
  - **Failed Attempts -** Changing input casting broke one pipeline while fixing the other, creating zero-sum compatibility issue
  - **Solution Architecture** -Dual C++ function implementation

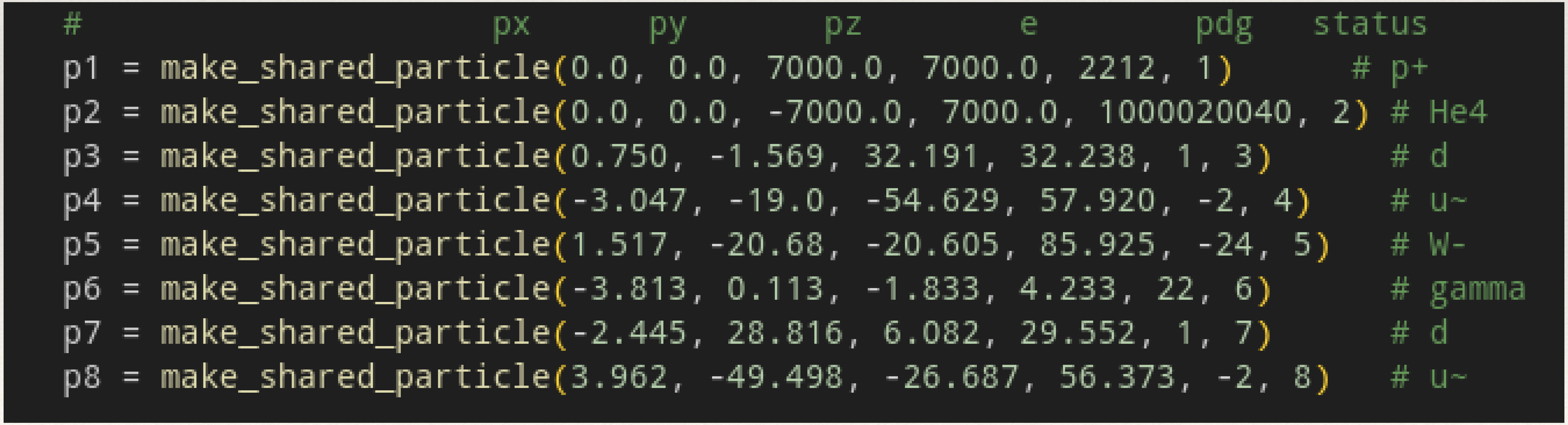


```cpp
// For JetReconstruction (shared_ptr input)
int particles_size(void* event) {
    auto e = static_cast<std::shared_ptr<HepMC3::GenEvent>*>(event);
}
// For test suite (raw pointer input)
int particles_size_raw(void* event) {
    auto e = static_cast<HepMC3::GenEvent*>(event);
}
```

HepMC3.jl: basic tree example success



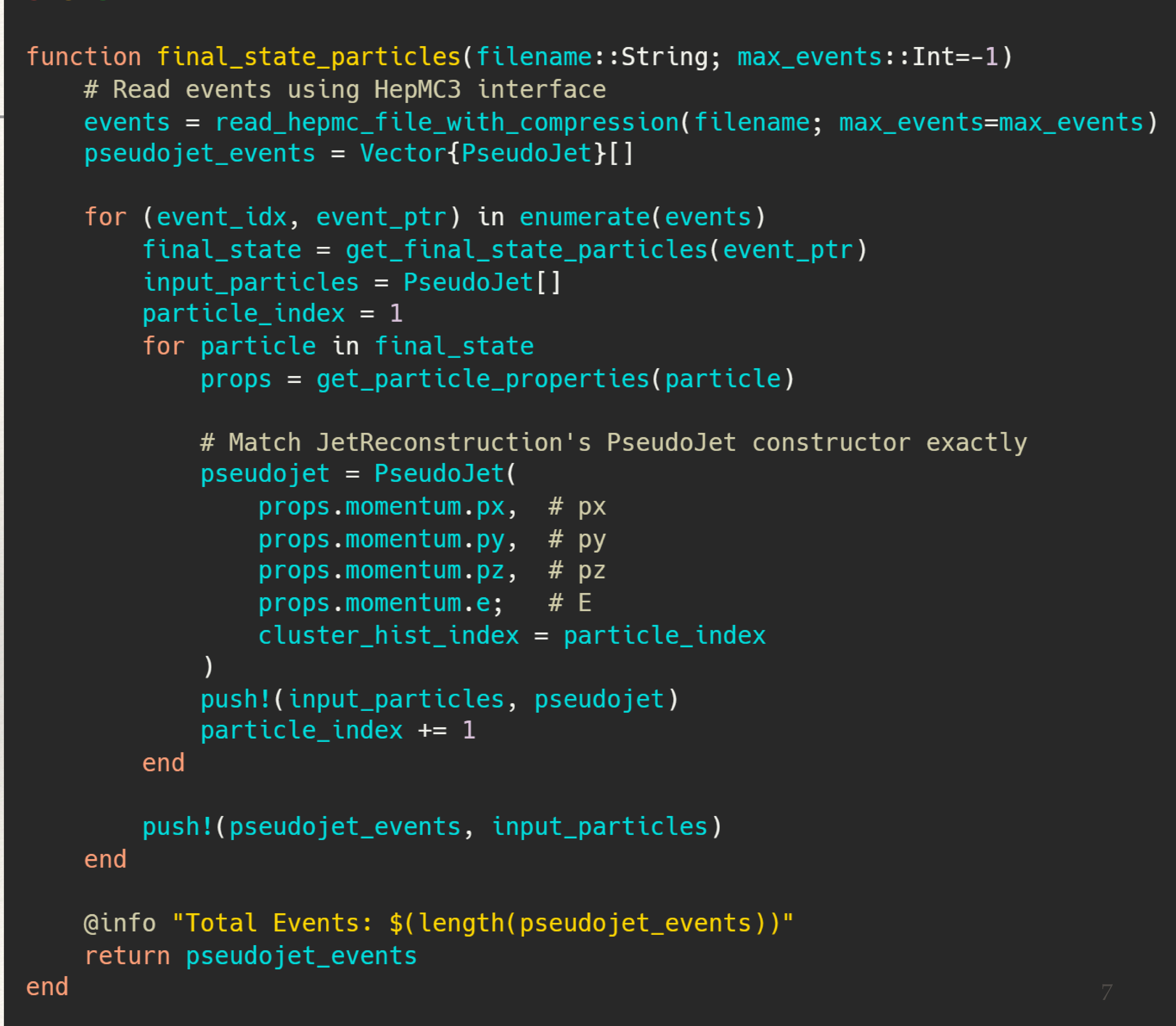
```
                      p7
p1                   /
  \v1__p2      p5---v4
        \_v3_/       \
        /    \        p8
   v2__p4     \
  /            p6
p3
```



```
#                            px        py          pz          e          pdg     status
p1 = make_shared_particle(0.0, 0.0, 7000.0, 7000.0, 2212, 1)      # p+
p2 = make_shared_particle(0.0, 0.0, -7000.0, 7000.0, 1000020040, 2) # He4
p3 = make_shared_particle(0.750, -1.569, 32.191, 32.238, 1, 3)      # d
p4 = make_shared_particle(-3.047, -19.0, -54.629, 57.920, -2, 4)    # u~
p5 = make_shared_particle(1.517, -20.68, -20.605, 85.925, -24, 5)   # W-
p6 = make_shared_particle(-3.813, 0.113, -1.833, 4.233, 22, 6)      # gamma
p7 = make_shared_particle(-2.445, 28.816, 6.082, 29.552, 1, 7)      # d
p8 = make_shared_particle(3.962, -49.498, -26.687, 56.373, -2, 8)   # u~
```

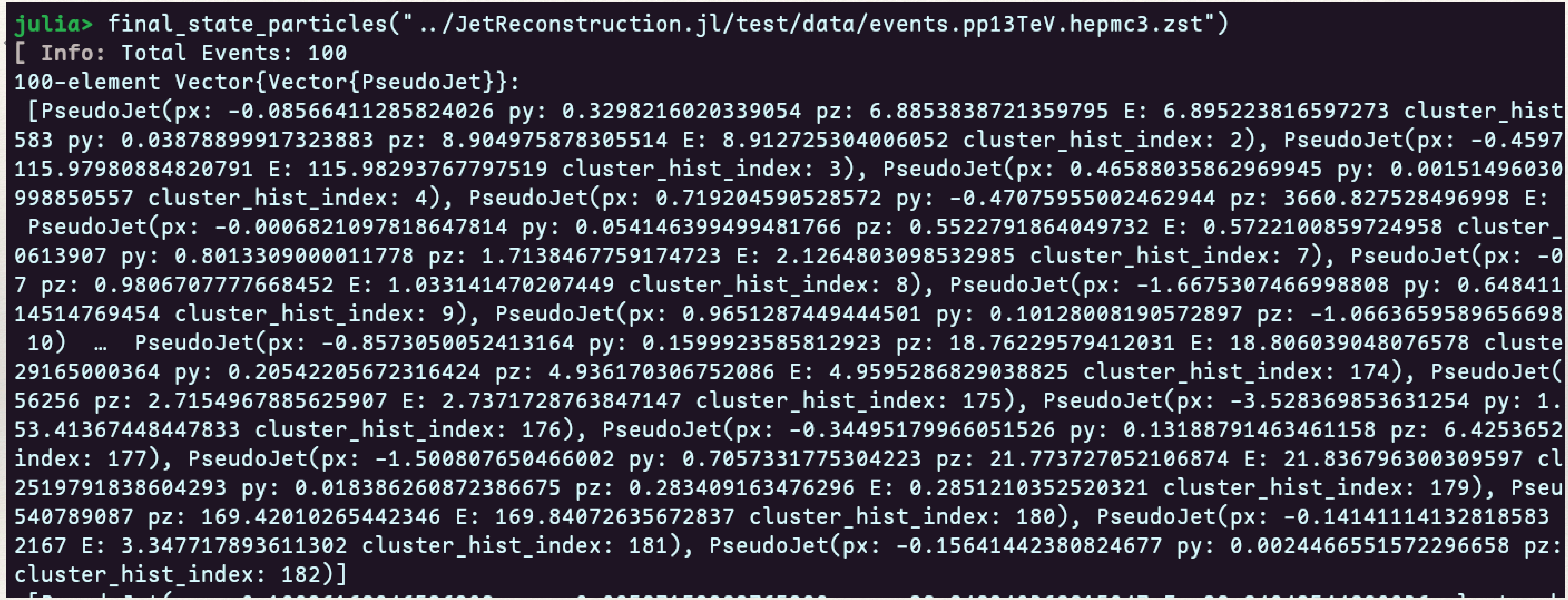
HepMC3.jl:

final state particles for Jet Reconstruction



```julia
function final_state_particles(filename::String; max_events::Int=-1)
    # Read events using HepMC3 interface
    events = read_hepmc_file_with_compression(filename; max_events=max_events)
    pseudojet_events = Vector{PseudoJet}[]

    for (event_idx, event_ptr) in enumerate(events)
        final_state = get_final_state_particles(event_ptr)
        input_particles = PseudoJet[]
        particle_index = 1
        for particle in final_state
            props = get_particle_properties(particle)

            # Match JetReconstruction's PseudoJet constructor exactly
            pseudojet = PseudoJet(
                props.momentum.px,  # px
                props.momentum.py,  # py
                props.momentum.pz,  # pz
                props.momentum.e;   # E
                cluster_hist_index = particle_index
            )
            push!(input_particles, pseudojet)
            particle_index += 1
        end

        push!(pseudojet_events, input_particles)
    end

    @info "Total Events: $(length(pseudojet_events))"
    return pseudojet_events
end
```

HepMC3.jl: final state particles for JetReconstruction



```
julia> final_state_particles("../JetReconstruction.jl/test/data/events.pp13TeV.hepmc3.zst")
[ Info: Total Events: 100
100-element Vector{Vector{PseudoJet}}:
 [PseudoJet(px: -0.0856641128582 4026 py: 0.3298216020339054 pz: 6.8853838721359795 E: 6.89522381659727 3 cluster_hist
583 py: 0.0387889917323883 pz: 8.904975878305514 E: 8.912725304006052 cluster_hist_index: 2), PseudoJet(px: -0.4597
115.97980884820791 E: 115.98293767797519 cluster_hist_index: 3), PseudoJet(px: 0.4658803586296994 5 py: 0.00151496030
998850557 cluster_hist_index: 4), PseudoJet(px: 0.719204590528572 py: -0.4707595500246294 4 pz: 3660.827528496998 E:
 PseudoJet(px: -0.0006821097818647814 py: 0.0541463994948176 6 pz: 0.552279186404973 2 E: 0.572210085972495 8 cluster_
0613907 py: 0.801330900001177 8 pz: 1.713846775917472 3 E: 2.126480309853298 5 cluster_hist_index: 7), PseudoJet(px: -0
7 pz: 0.980670777766845 2 E: 1.033141470207449 cluster_hist_index: 8), PseudoJet(px: -1.667530746699880 8 py: 0.648411
14514769454 cluster_hist_index: 9), PseudoJet(px: 0.965128744944450 1 py: 0.101280081905728 97 pz: -1.066365958965669 8
 10)  …  PseudoJet(px: -0.857305005241316 4 py: 0.159992358581292 3 pz: 18.76229579412031 E: 18.806039048076578 cluste
29165000364 py: 0.205422056723164 24 pz: 4.936170306752086 E: 4.959528682903882 5 cluster_hist_index: 174), PseudoJet(
56256 pz: 2.715496788562590 7 E: 2.737172876384714 7 cluster_hist_index: 175), PseudoJet(px: -3.528369853631254 py: 1.
53.41367448447833 cluster_hist_index: 176), PseudoJet(px: -0.344951799660515 26 py: 0.131887914634611 58 pz: 6.4253652
index: 177), PseudoJet(px: -1.500807650466002 py: 0.705733177530422 3 pz: 21.77372705210687 4 E: 21.836796300309597 cl
251979183860429 3 py: 0.018386260872386675 pz: 0.283409163476296 E: 0.285121035252032 1 cluster_hist_index: 179), Pseu
540789087 pz: 169.42010265442346 E: 169.84072635672837 cluster_hist_index: 180), PseudoJet(px: -0.141411141328185 83
2167 E: 3.347717893611302 cluster_hist_index: 181), PseudoJet(px: -0.156414423808246 77 py: 0.002446655157229 6658 pz:
cluster_hist_index: 182)]
```

Conclusions

- **HepMC3.jl** is in early development and accomodate plans with comprehensive functionality for high-energy physics event processing

- The package can be a very useful addition to the Julia HEP ecosystem enabling seamless integration with **JetReconstruction.jl** and physics analysis workflows

- Julia **BinaryBuilder** and **CxxWrap** are powerful tools to streamline installation and deployment of complex C++ physics libraries like HepMC3

- Dual pointer architecture proved essential for HepMC3 compatibility - measured zero performance overhead between **shared_ptr** and **raw pointer** paths

If you want to try… (on x86 linux)

1. install Julia version > 1.9
   - just download the binary (`https://julialang.org/downloads`) and `untar` it
   - include in PATH the `julia-1.9.3/bin directory`
2. clone HepMC3.jl for the examples (Make sure to also clone and install JetReconstruction.jl)
   - `git clone` ~~`https://github.com/JuliaHEP/HepMC3.jl.git`~~
   - `cd HepMC3.jl`
3. install locally all the needed packages and dependencies by the examples
   - `julia --project=. -e 'import Pkg; Pkg.instantiate()’`
4. run an example (e.g. Jet Reconstruction event structs)
   - `julia --project=. -i examples/test_jetreconstruction_pipeline.jl`