Pacific Northwest
NATIONAL LABORATORY

julia in HEP Analyses

Jan Strube, PNNL / University of Oregon

Marcel Stanitzki, DESY

U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

Pacific Northwest NATIONAL LABORATORY

Overview

- The goal of this talk is to demonstrate how tasks in high energy physics can be solved using the Julia programming language. It is understood that **there is more than one way to do it**.
  - For a list of case studies, https://juliacomputing.com/case-studies/
- I will intersperse the introduction of language features with use cases.
- Outline
  - Interactivity and ease of use
  - Implementation of a convex hull for 3D shower shapes in a calorimeter.
  - Interoperability with Python and C++
  - Event shapes in Julia
  - Multi-threaded processing and parallelism
- Material is based on our paper and the corresponding code examples.

Pacific Northwest NATIONAL LABORATORY

Introduction

Starting from a personal story.

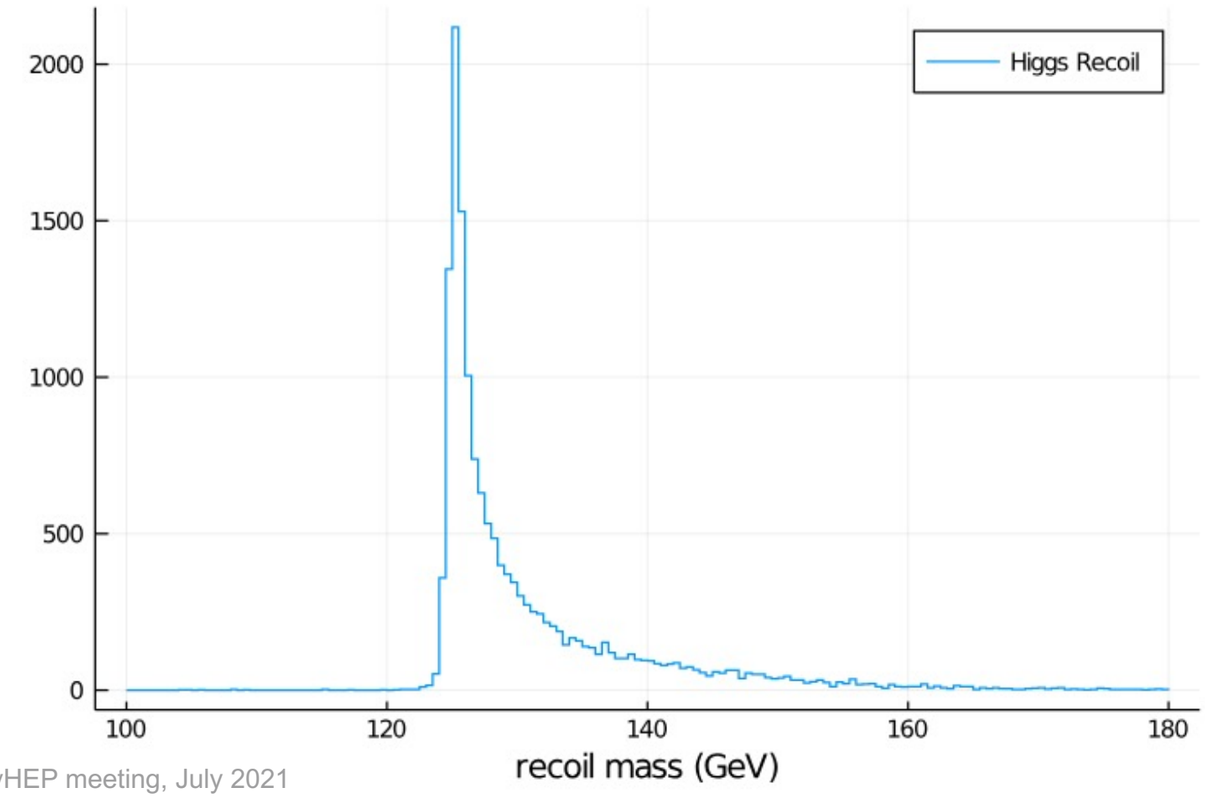
- I wanted to study ILC calorimeter clusters containing thousands of hits. What if we treat them like a rigid body?
- I had a student with a strong background in computational methods and an interest in visualizations.
- I had large amounts of data in Linear Collider I/O (LCIO) format.
- I wanted to try out concepts like fractal dimensions and other algorithms to characterize the shapes of clusters from different particles.
- My options were
  - C++ → steep learning curve for the student
  - Java → would have been fine for that project, but shrinking role in HEP
  - Python → probably would have been OK, but not great for numerical algorithms
  - ✓ Julia → has been used for numerical algorithms, good support for linear algebra

The "ju" in jupyter is for Julia

- Excellent for teaching
- Interactive data exploration is much more efficient
- Support for Jupyter notebooks is becoming ubiquitous
- Julia works seamlessly on jupyterhub servers, e.g. at DESY, or SWAN

```
In [8]: plot(
            recoilMassList,
            seriestype=:stephist,
            bins=100:0.5:180,
            label="Higgs Recoil",
            xlabel="recoil mass (GeV)"
        )
```

Out[8]:



Pacific Northwest NATIONAL LABORATORY

Julia has excellent support for packages

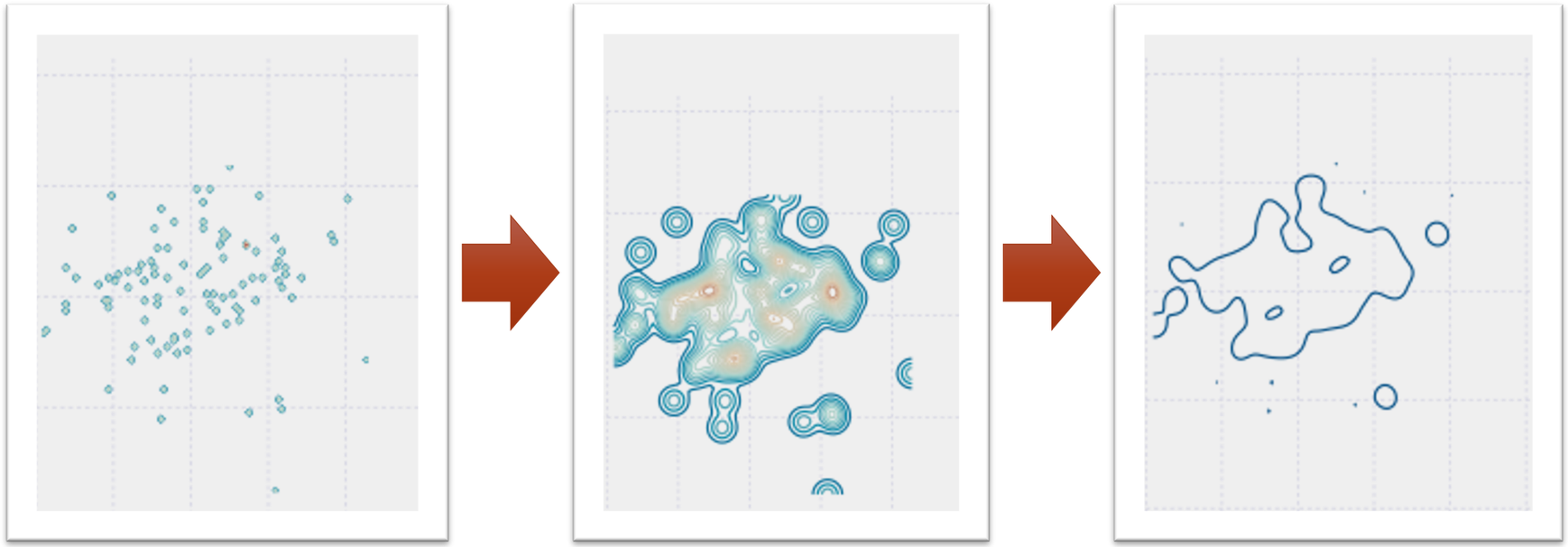
- The command prompt (REPL) has a package mode
  - Type "]" to switch to package mode, "Backspace" to go back to julia mode
- Julia packages can be registered in the central registry
  - Or in any other registry you want to set up
  - "add <package name>" in package mode
  - "add https://github.com/package" for unregistered packages
  - Supports installing different tags / branches directly
- Julia has native support for tests
  - "test <package>"
- Package updates run all tests by default
- (Many) Core language features are implemented in Julia
  - No conflicts between numpy <=> tensorflow <=> pytorch …
  - Packages are much more composable than Python

Use case: Analysis of calorimeter clusters

The project was exploratory in nature. Interactivity is a plus. Strong support for linear algebra is a requirement.

- Requirement: Productivity
  - Summer student, no prior knowledge of ILC physics
  - Had taken a CS class where they had used Java. Algorithms should be straightforward to translate from class.
- Requirement: Maturity
  - I did not want to be the only point of contact in case of issues
  - Focus should be development of novel algorithms, not installation or compilation of libraries
- Goal: Improve the analysis of ILC calorimeter clusters
  - Cluster based particle ID --> improved reconstruction using shape templates
- Seemed like an ideal use case for trying out Julia

From clusters to volumes



Hit energies are binned according to position

Resulting density grid is Gaussian blurred

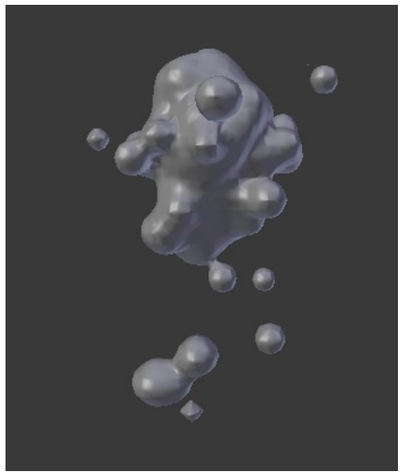
Isosurface is generated using Marching Cubes

(border is expanded as needed)

Pacific Northwest NATIONAL LABORATORY

Calorimeter clusters as rigid bodies

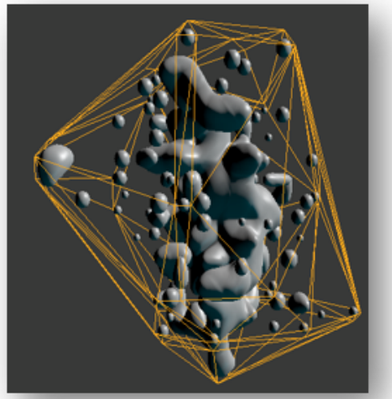
Pion (50 GeV)

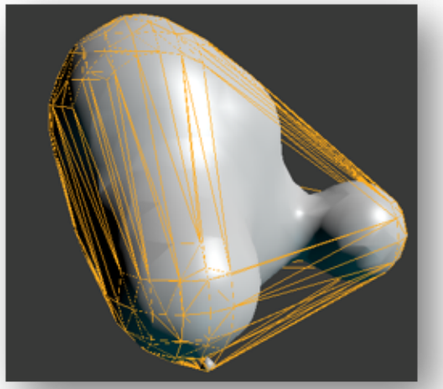


Electron (50 GeV)

Muon (50 GeV)

Convex hull





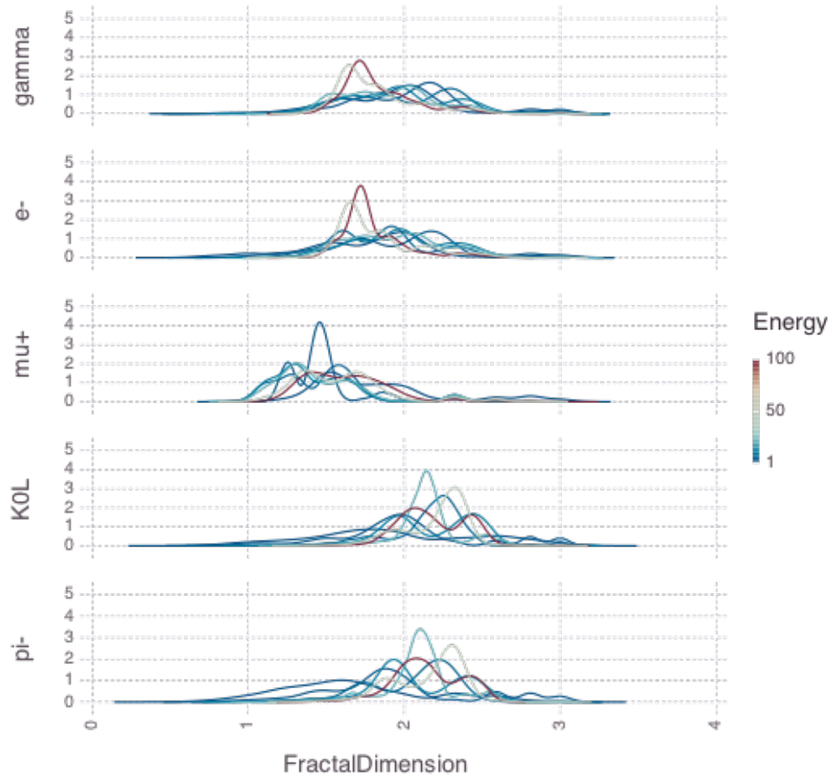
Convex hull allows defining additional parameters on the cluster, such as the ratio of empty to filled volume, i.e., sparsity of the cluster.

Visualizations in blender

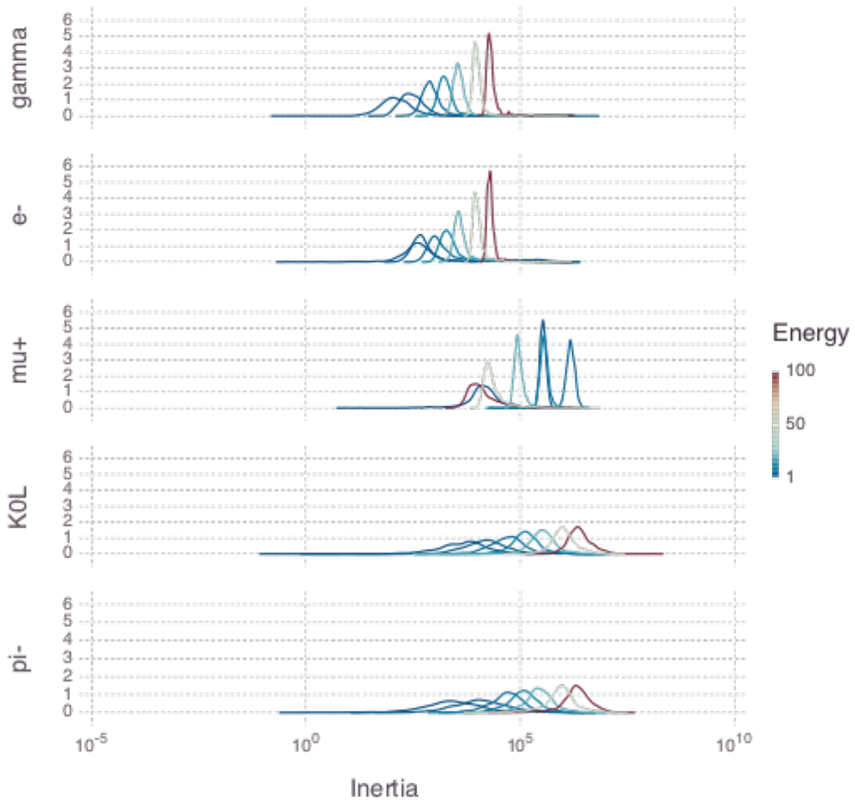
https://github.com/ShnitzelKiller/MeshTools.jl

Discriminating calorimeter shower shapes of different particle types

Fractal Dimension



Moment of Inertia



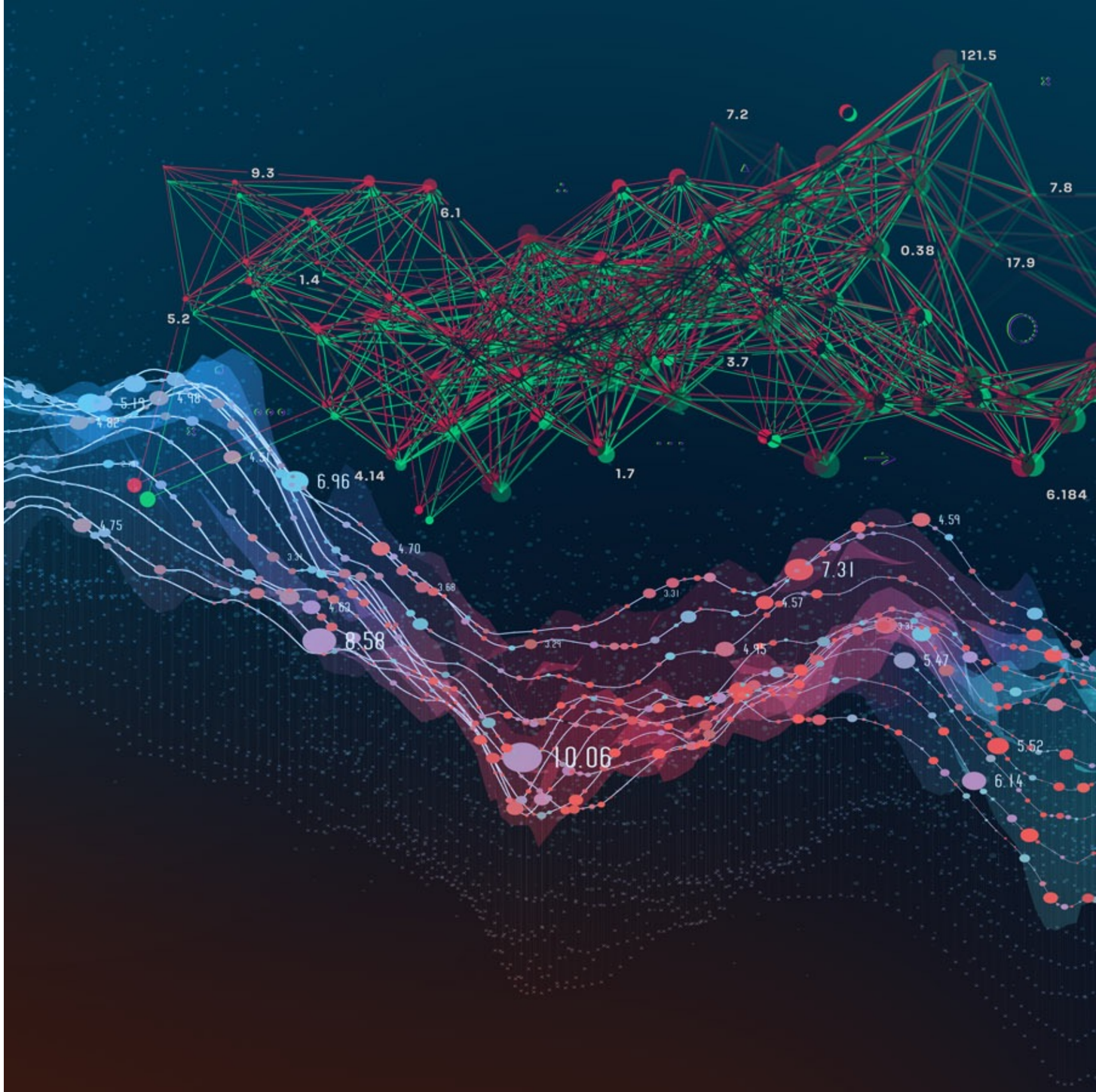
Fractal dimensions had been used for particle identification. Our implementation followed the paper.

Using rigid body dynamics to separate shower shapes adds discriminatory power.

Pacific Northwest
NATIONAL LABORATORY

Language interoperability

I don't want to re-write everything from scratch



Pacific Northwest NATIONAL LABORATORY

Julia integrates well with C++

- HEP has developed a lot of irreplaceable C++ code
  - Language interoperability is a strong requirement
- Julia has support libraries for Boost.python-like wrappers
  - Write a small wrapper class in C++ that exports classes and methods to Julia
  - Requires a bit of boilerplate code, but allows fine-grained control
  - For example, convenience methods to avoid type-casting
    - ✓ **C++ LCIO**: `Track t = static_cast<Track>(collection.getElementAt(iterator));`
    - ✓ **LCIO.jl**: `track = collection[iterator]`
- LCIO.jl: interface to the LCIO file format
- FastJet.jl: wrapper around FastJet and plugins
- See https://github.com/JuliaHEP for more

Pacific Northwest NATIONAL LABORATORY

Julia integrates well with Python

Julia starts being used as the backend for python packages:

- https://github.com/SciML/diffeqpy: For solving differential equation
- https://github.com/MilesCranmer/PySR: Symbolic regression

Julia can use existing Python packages

- PyPlot.jl: Matplotlib integration
- UpROOT.jl: uproot wrapper
  - Using VectorOfVectors instead of awkward-array
  - UnROOT.jl for a Julia implementation of uproot

Algorithms in Julia: Fox-Wolfram moments

- Well-known analysis technique in HEP
  - Observables for the Analysis of Event Shapes in e+e Annihilation and Other Processes

$$H_l \equiv \left(\frac{4\pi}{2l+1}\right) \sum_{m=-l}^{+l} \left| \sum_i Y_l^m(\Omega_i) \frac{|\vec{p}_i|}{\sqrt{s}} \right|^2$$

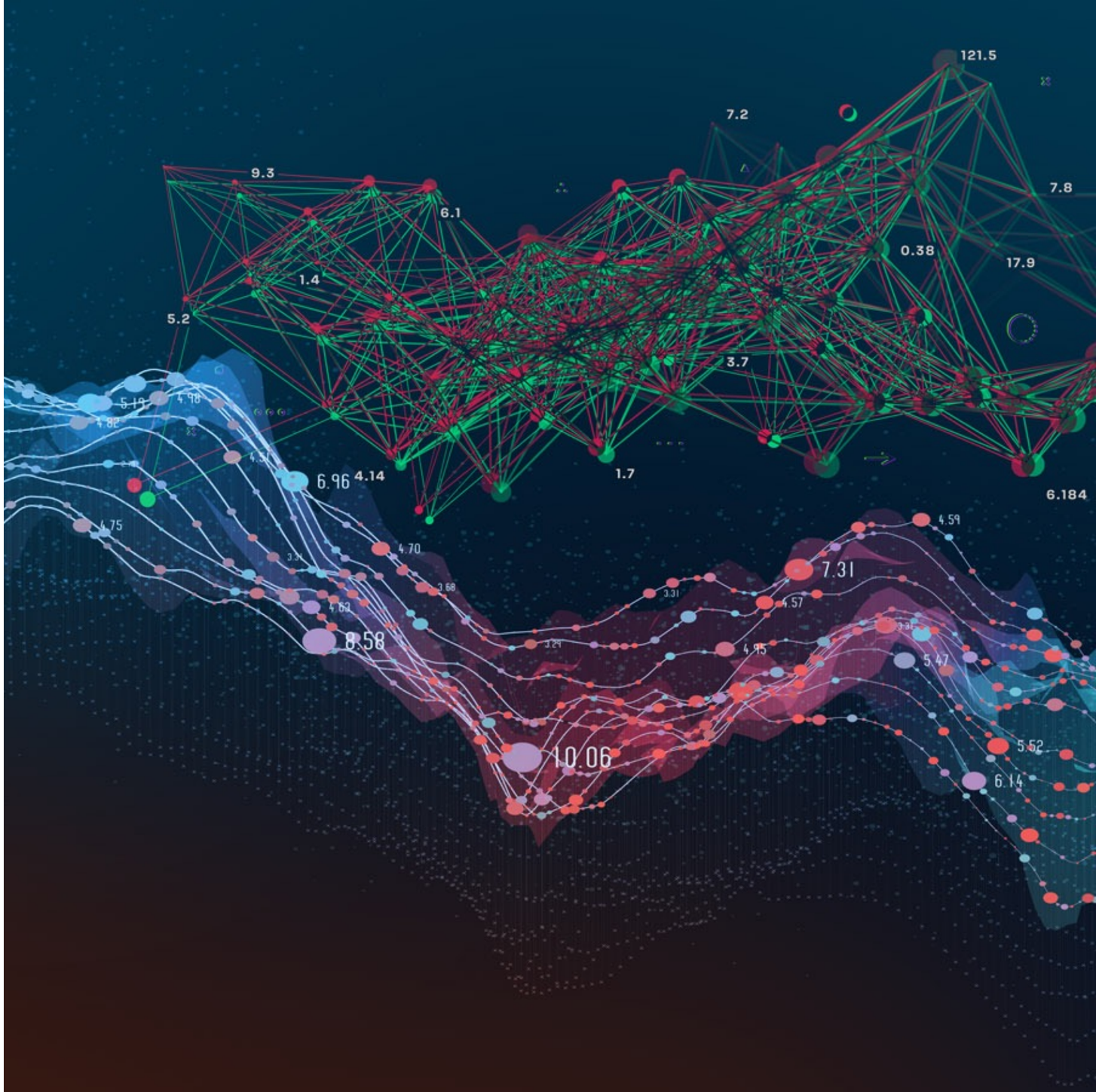
$$= \sum_{i,j} \frac{|\vec{p}_i||\vec{p}_j|}{s} P_l(\cos\varphi_{ij}),$$

- Implementation exists in pythia (PYFOWO, in FORTRAN77)
  - Translate to C++, Julia, Python, and benchmark the results
- Straightforward translation (aim for 15 minutes or less for reproducible research)
  - **Julia: 287 events / s**
  - C++: 182 events / s
  - Python: 0.54 events / s
  - Python + numba: 204 events / s
    - ✓ Comes with all the pros and cons of using a C++ compiler

Pacific Northwest
NATIONAL LABORATORY

Parallelism

Because it's 2021



Pacific Northwest NATIONAL LABORATORY

Support for parallel and concurrent programs

- Green threads: Go-like concurrency model
  - Schedule thousands of tasks onto the available threads
  - Communicate via `Channels`
- Multi-threading
  - `Threads.@threads for i = 1:100 a[i] = Threads.threadid() end`
  - runs `nThreads` iterations of the loop in parallel, shared memory parallelism
- Multi-processing
  - Each process runs in its own context, a la Python
  - Same Node / multiple node processing (`Distributed` processing)
- GPU programming
  - Support for NVIDIA CUDA / Intel GPUs with oneAPI / AMD GPUs
  - See JuliaGPU for more information

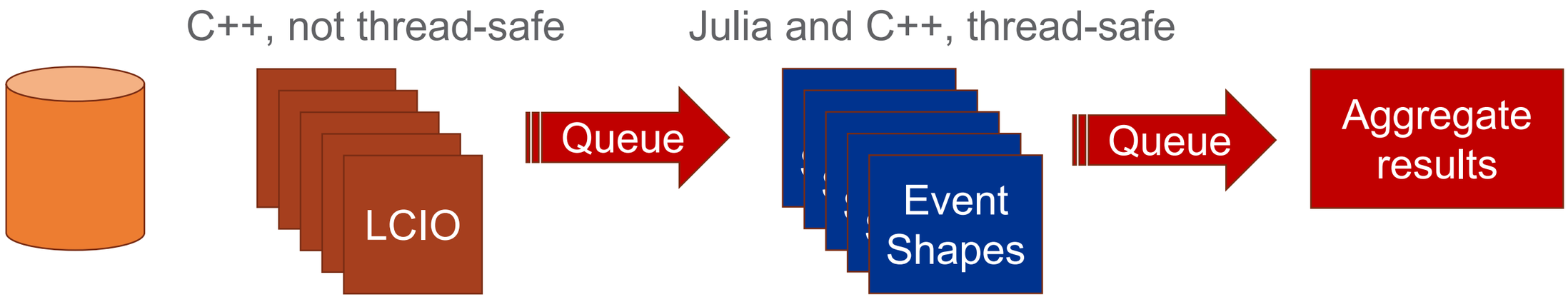
Example: Parallel processing of LCIO files

Goal: Demonstrate speed-up with different levels of parallelism with Julia

- Reading files concurrently
  - Many C++ libraries are not thread-safe.
  - We wrap each instance in a separate process. Communicate via channels.
  - Each file is read sequentially, multiple files read at the same time.

- Processing events in parallel
  - Events are pulled from the event queue and processed
  - Fox-Wolfram, implemented natively
  - Fastjet, using the C++ bindings

Putting it all together



One hard drive

Many readers, each uses their own process space

One queue

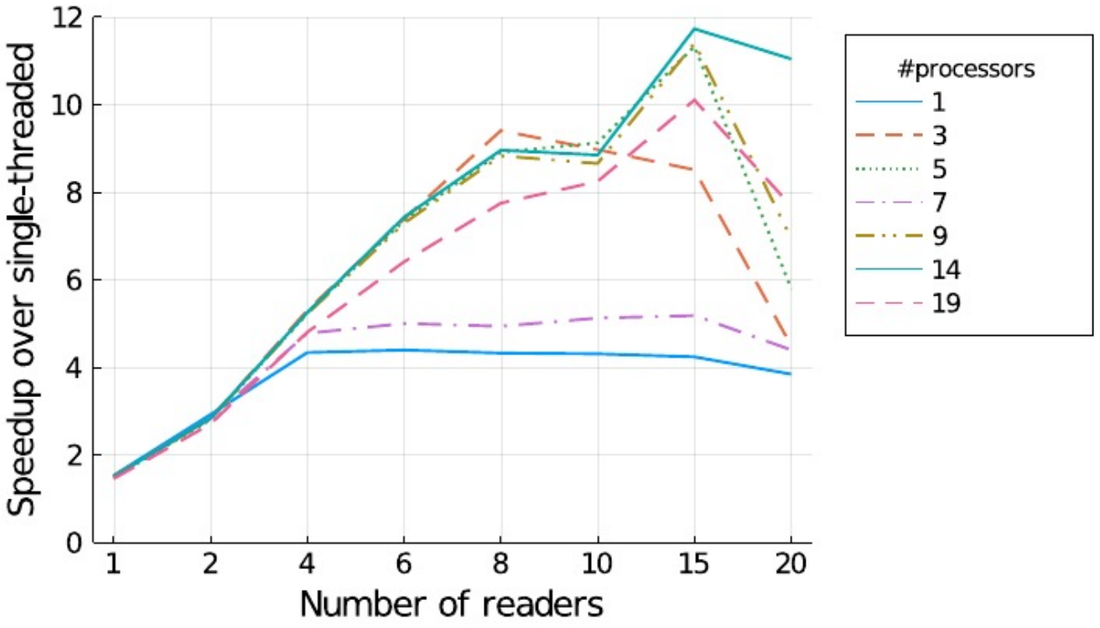
Many event shape processors, can use shared-memory parallelism

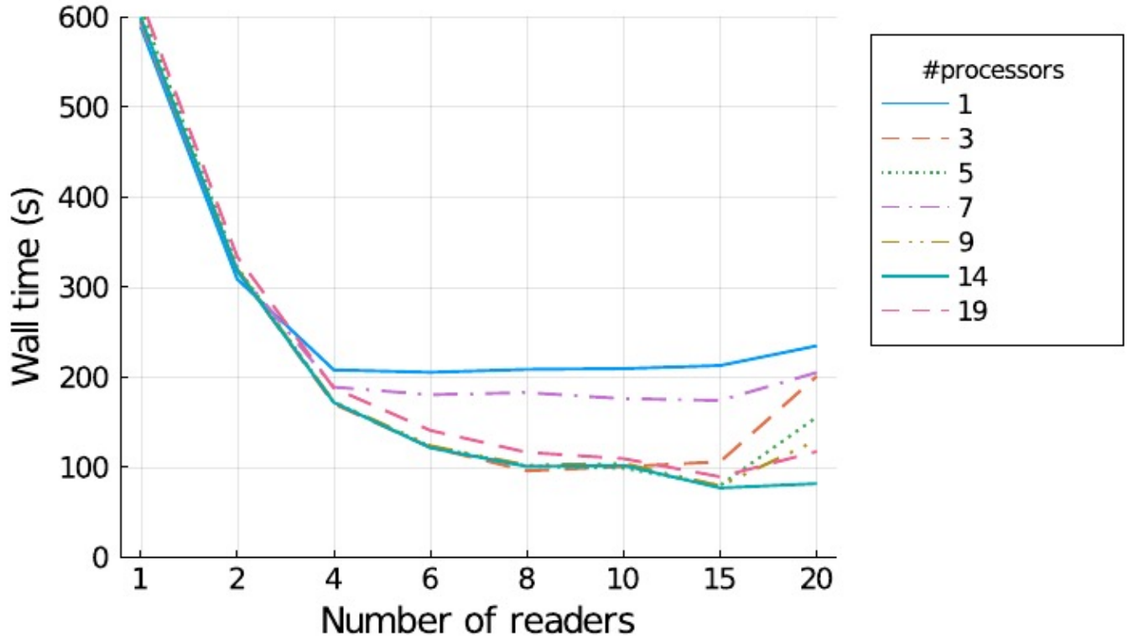
One queue

One aggregator

Results of speed-up test

Intel® Xeon® CPU E5-2640 v4 @ 2.40GHz CPU, 128 GB RAM.





Significant speedup can be achieved on a multi-core system by combining multiple file readers with multiple event processors. The details depend on the workload and the architecture.

Pacific Northwest NATIONAL LABORATORY

Summary

- Support for interactive programming, package manager, integrated test suite help getting newcomers up to speed quickly
- Support for distributed computing and multi-threading paradigms enable efficient use of modern compute architectures
- Interoperability with C++ and Python allows mixing / matching of existing libraries with new code

=> Julia is a good option for productive HEP analysis work

Pacific Northwest
NATIONAL LABORATORY

Thank you