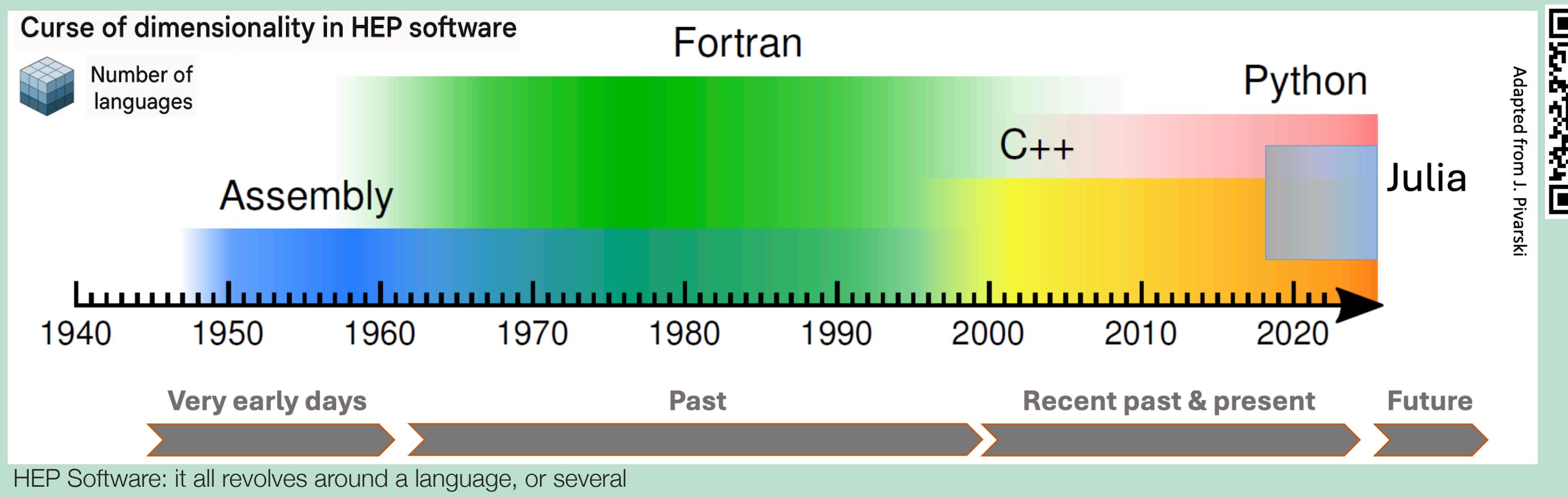




# Advancing High Energy Physics Data Analysis with Julia: A Case for JuliaHEP

Ianna Osborne, Princeton University

For the past 25 years, the High-Energy Physics (HEP) community has steadily adopted Python as its primary language for data analysis, supported by compiled-backend libraries like NumPy ([Harris et al., 2020](#)) and [Awkward Array](#), along with other Python tools like [Uproot](#) for I/O. However, the Julia programming language offers a compelling alternative, addressing the two-language problem with C++-comparable performance and Python-like ease of use. [JuliaHEP](#) is an informal organization that aims to unify effort in developing Julia projects related to high energy physics, outlining its advantages, ongoing developments, and integration with existing Python-based tools.

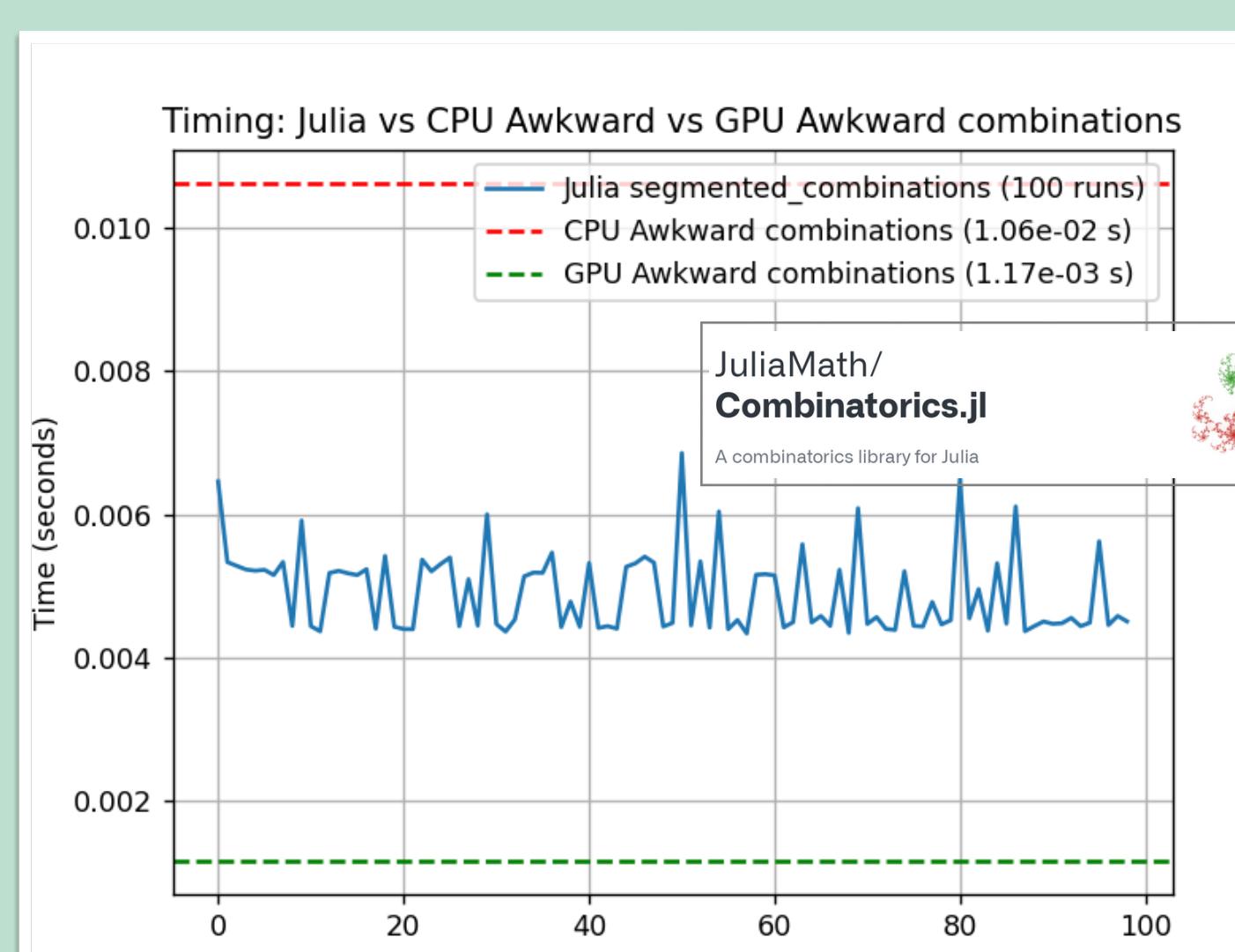


**Python** provides ease of use and a rich scientific ecosystem, but it struggles with performance for large-scale analyses.

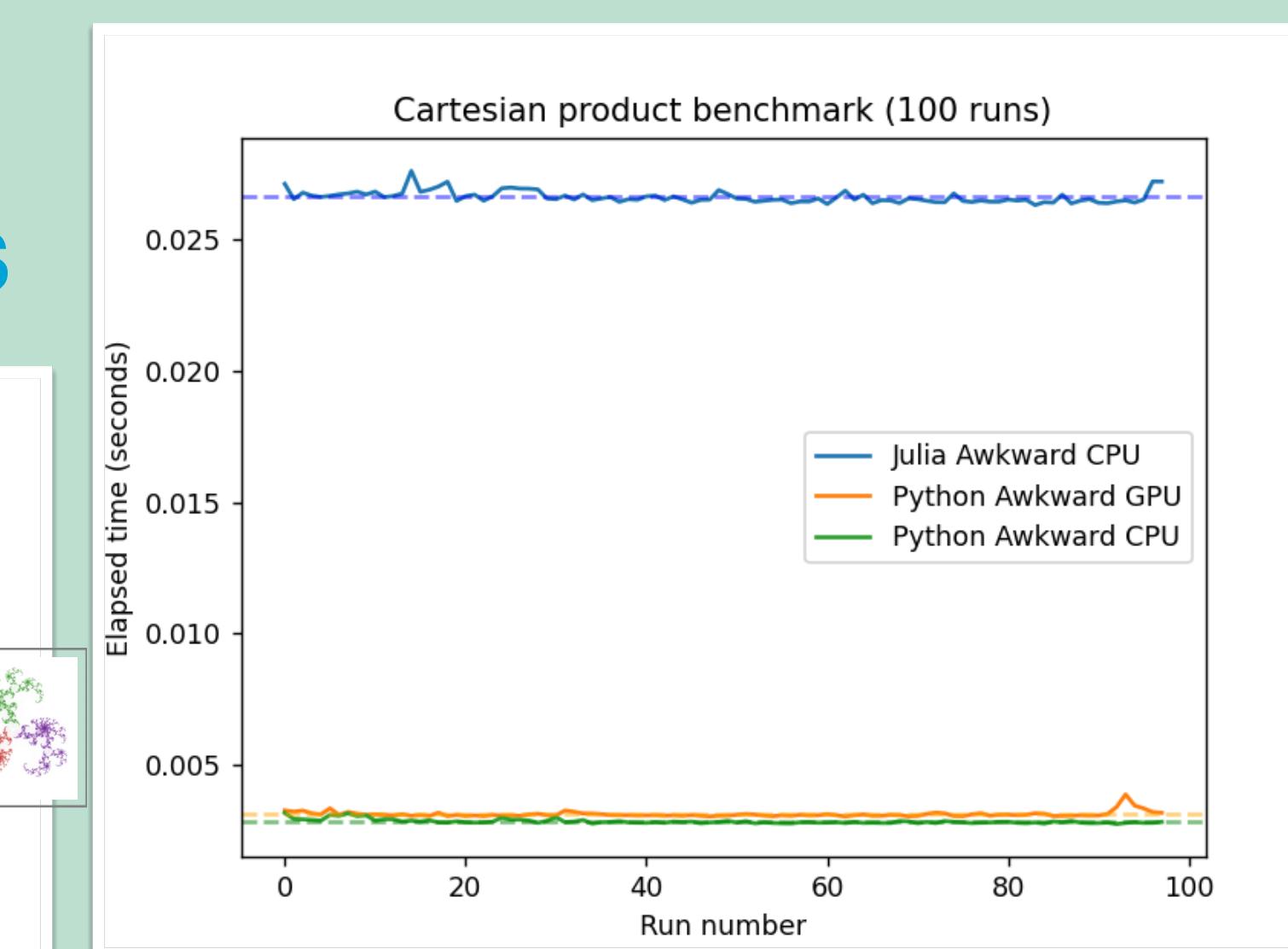
**C++**, on the other hand, offers speed but comes with increased complexity and slower development cycles.

**Julia**, a modern language designed for scientific computing, promises the best of both worlds: high-level expressiveness with near-native execution speed.

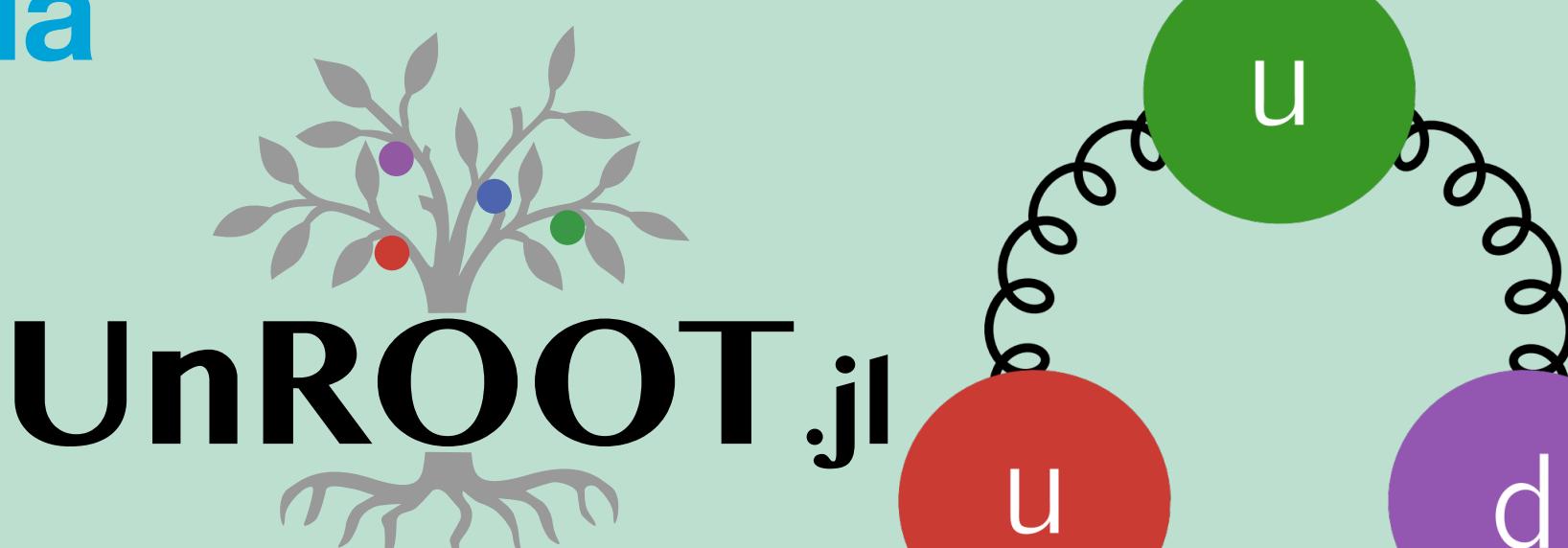
## Performance benefits for HEP computations



Comparing Awkward Array combinatorics in Julia and Python with C++ and GPU backends, highlight cases where Julia can offer significant speedup with less boilerplate code.



## JuliaHEP: an emerging set of tools and libraries designed to facilitate HEP data analysis in Julia



**Julia in HEP**  
Graeme Andrew Stewart, Alexander Moreno Brivio, Philippe Gras, Benedict Heijer, Uwe Hernandez Acosta, Tamas Gal, Jerry Ling, Pere Matz, Mikhail Mikhasev, Oliver Schulz, Sam Skjærseth  
Julia is a mature general-purpose programming language, with a large ecosystem of libraries and more than 12000 third-party packages. It is highly performant, dynamic, interactive, and accessible as Python with NumPy, but achieves run-time performance on par with C/C++. In this paper, we describe the state of adoption of Julia in HEP, where momentum has been gathering over a number of years. We show how, as the support for HEP has matured, developments have benefited from Julia's core design choices, including its JIT compilation, multiple dispatch, and GPU acceleration. We also show how libraries developed outside HEP for plotting, statistics, fitting, and scientific machine learning are extremely useful. We believe that the powerful combinators of flexibility and speed, the wide selection of scientific programming tools, and support for all modern programming paradigms and tools, make Julia the ideal choice for a future language in HEP.

<https://arxiv.org/abs/2503.08184>

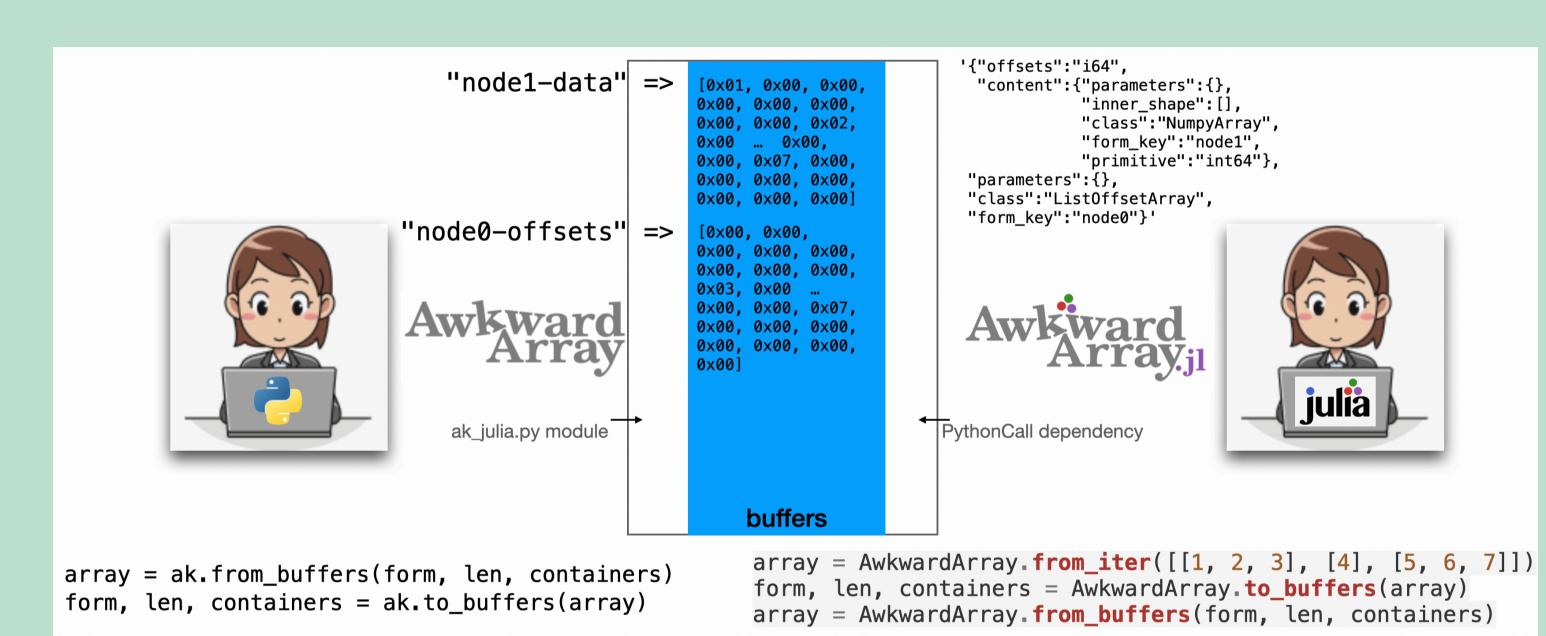
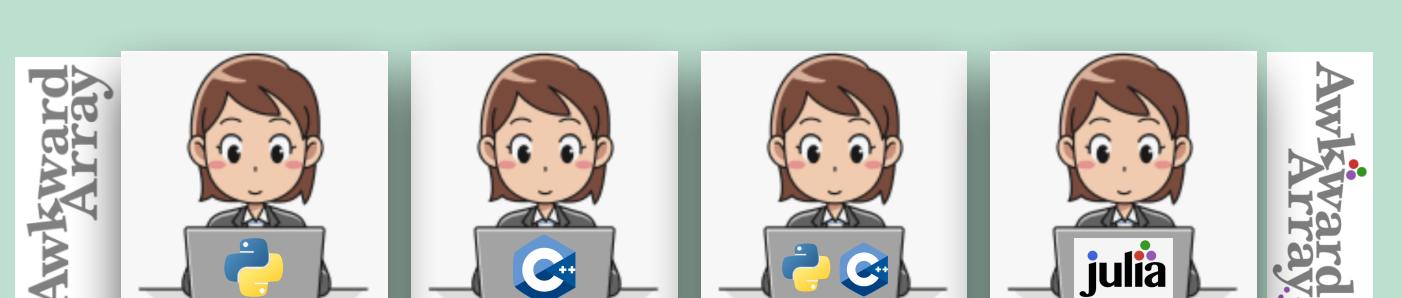
Julia's strengths: just-in-time (JIT) compilation, multiple dispatch, and GPU acceleration.

- Speed of code execution
- Designed for interactive use
- Composability, leading to highly reusable code that is easy to maintain
- Package management: built in state-of-the-art package manager
- Ease of integration with other languages

## Julia Introduction to Python HEP Community

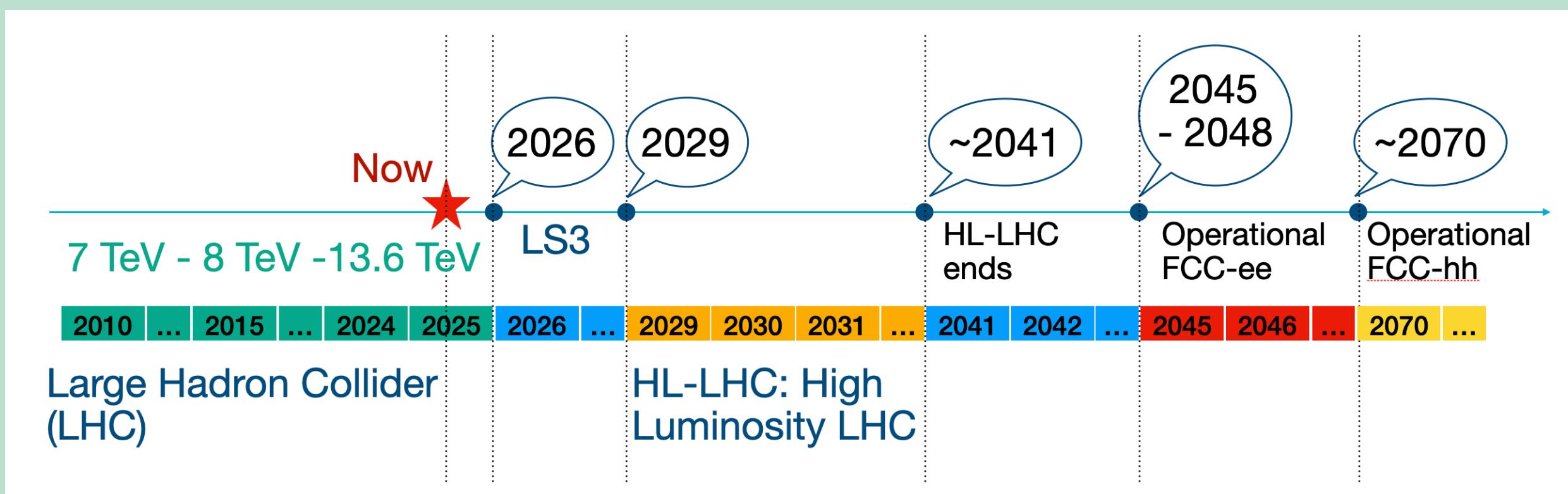
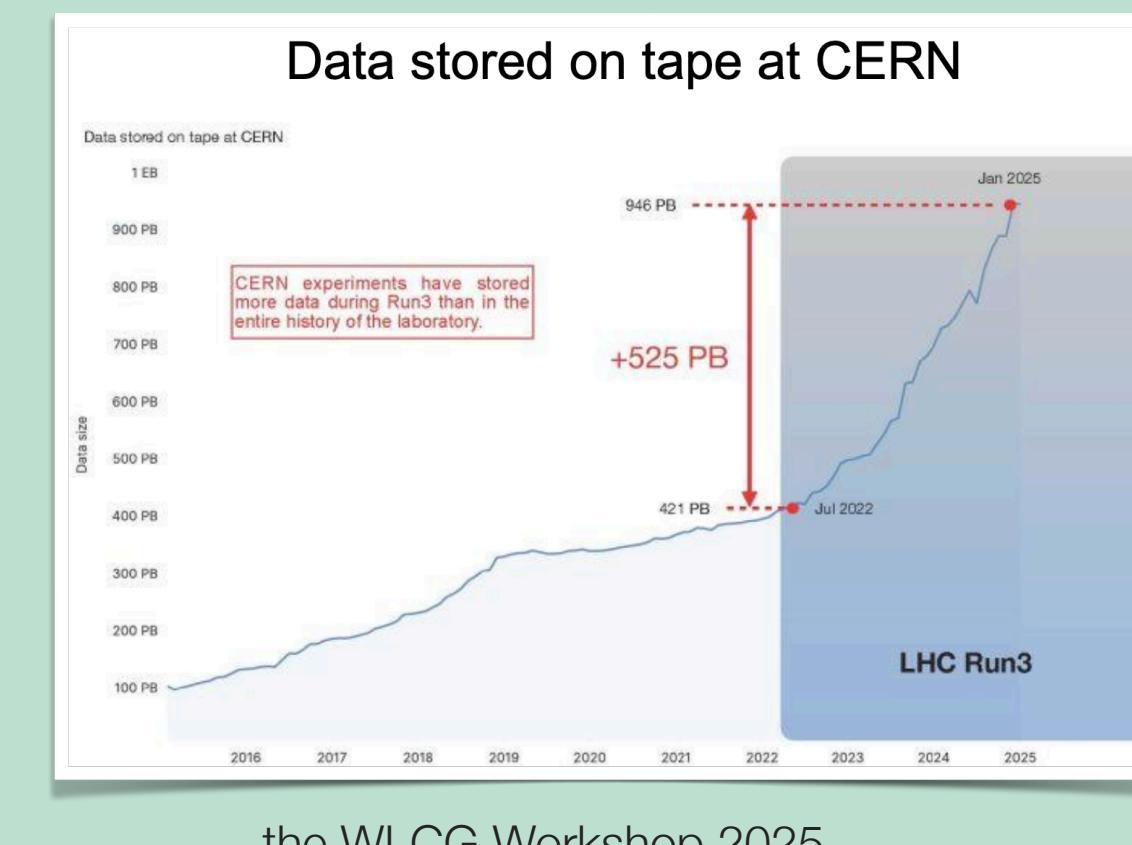
# Awkward Array.jl

Physicists can gradually adopt Julia without abandoning Python-based tools, thanks to interlanguage operability.



Physicists are using Awkward Array in Python and data format conversion is the hardest part of language boundary-hopping.

Sharing Awkward Array data structures between Python and Julia to encourage the Python users to run their analysis both in an eco-system of their choice and in Julia



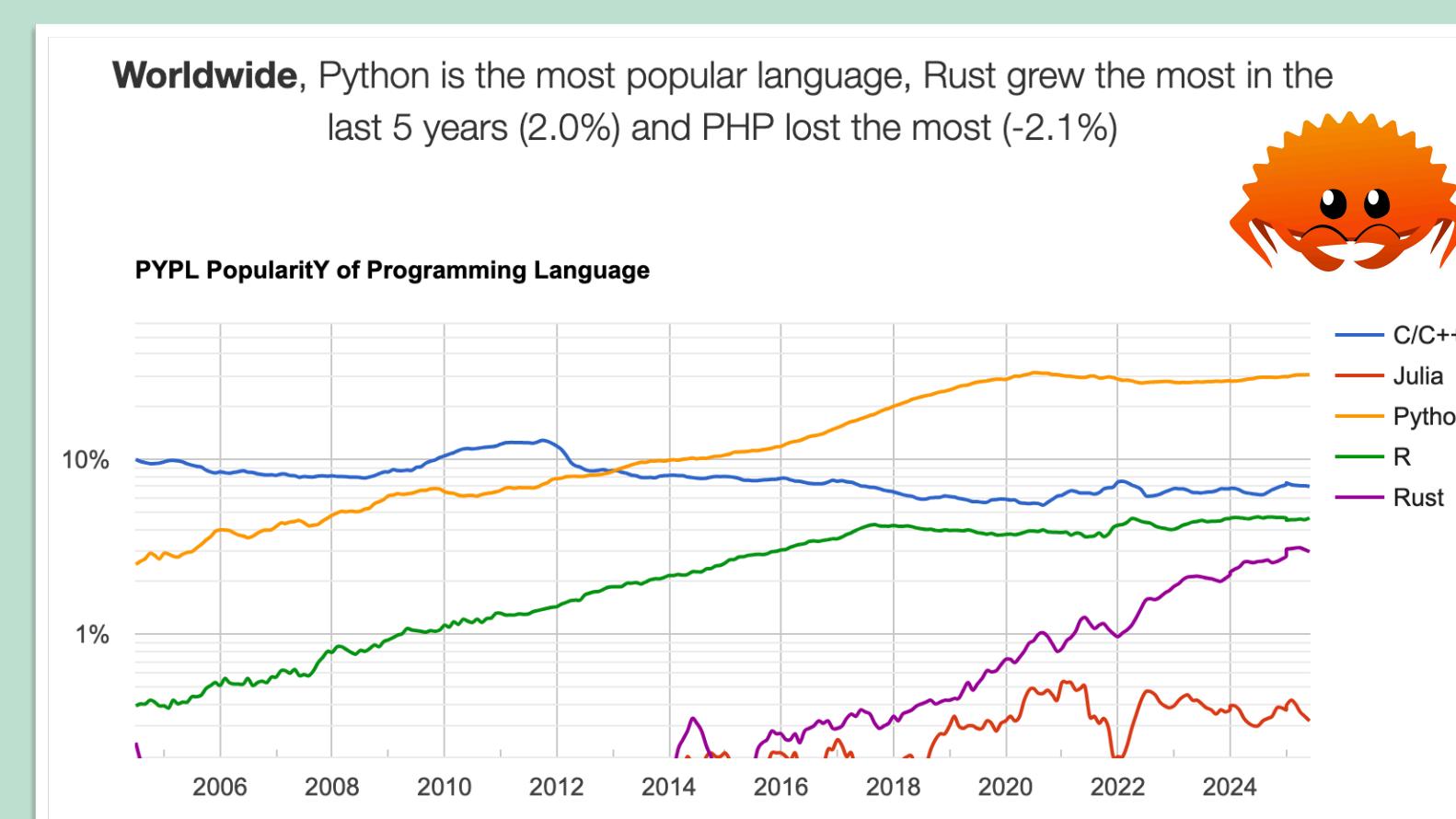
Case studies from CMS and other experimental collaborations illustrate Julia's real-world applicability in tasks like event processing, data analysis, and simulation (see upcoming JuliaHEP 2025 workshop).



## Scikit-HEP and PyHEP: Scientific Python HEP Community



Julia's potential for parallel computing and its built-in support for GPUs can provide an efficient pathway for scaling HEP computations.



SciPy 2025