Why Julia?

Julia is a fast and expressive programming language, delivering the speed of C++ and Fortran together with the productivity of Python, MATLAB, and R. This fundamentally empowers diverse teams to work together to develop and deploy performant programs at scale by solving the two language problem.

With that strong combination, switching to Julia might seem like a no-brainer, but there may be some users for whom the speed and productivity gains do not justify making the switch.

What Is the Purpose of this White Paper and Who Is It For?

The purpose of this white paper is to explain some of the benefits of Julia and describe who might — and who might not — realize significant gains from switching to Julia.

The audience for this white paper includes:

- Engineers, researchers, scientists, programmers and managers
- Julia users and hobbyists who are looking to expand their use of Julia in production
- Julia champions who are looking to expand the use of Julia within their organization
- Decision-makers who are responsible for improving operations, efficiency, innovation and product delivery
- Users of Python, R, MATLAB and other high-level languages who would benefit from increased performance
- Users of C++, Java, Fortran and other low-level languages who would benefit from using a more expressive high-level language for fewer lines of code without sacrificing speed

- Anyone who would benefit from faster time to market with improved communication across teams and a shared expressive codebase

Benefits

Julia delivers:

High performance with efficient native compilation, multi-threading, and distributed and GPU compute

- High productivity with the semantics and expressiveness of a dynamic language like Python, R, Stata and MATLAB while avoiding the traditional overheads (up to 100x faster)
- Ease of use with straightforward semantics that are easy to learn, easy to write, and easy to analyze and understand
- Composability with packages that are built to work together and with the code that you write by default
- Interoperability with Python, R, C, C++, Java, and other languages through tooling that enables 2-way integration of codebases
- A single language that can span prototyping and deployment,reducing errors and solving the two-language problem
- A robust open-source package ecosystem, including more than 10,000 registered open-source packages
- A rich community of users and contributors - Julia has been downloaded more than 50 million times, has 440,000 cumulative GitHub stars (including Julia + Julia packages), and has more than 10,000 contributors to registered packages

As a result, Julia is the leading language for scientific machine learning, simulation, modeling, and much more.

Users include:

Pharmaceuticals	AstraZeneca, Merck, Pfizer, Sanofi, United Therapeutics
Technology	Amazon, Apple, Cisco, Facebook, Google, Intel, NVIDIA, Uber
Finance & Economics	Aviva, BlackRock, Conning, Federal Reserve Bank of New York, State Street
Space	NASA, Brazil National Institute for Space Research (INPE), US National Energy Research Scientific Computing Center (NERSC)
Sport	Williams F1 Racing
Energy	AOT Energy, EDF, LAMPS PUC-RIO, Los Alamos, Mitsubishi Electric Research Laboratories
Industrial Simulation	Instron Auto Crash Simulation
Aviation	Boeing, FAA, Lincoln Labs, Zipline
Robotics	MIT Robot Locomotion Group, UC Berkeley Autonomous Race Car

How do these organizations use Julia?

- Instron uses Julia to improve their automobile crash simulators, including a 500x speedup that reduced runtime from months to hours.
- Sanofi uses Julia to treat more cancerous tumor cells faster, optimize dosage and timing, and simulate combination therapies.
- NASA uses Julia to model spacecraft operations.
- The Federal Reserve Bank of New York uses Julia to model the macroeconomy 10-11x faster, provide better estimates faster, and with 50% fewer lines of code.
- State Street uses Julia to improve the efficiency of foreign exchange trading.

What Is the Two Language Problem, and How Does Julia Help Solve It?

Historically, programming languages have fallen into two categories:

	Examples	Advantages	Disadvantages
Fast, low-level languages	C, C++, Java, Fortran	Fast in production	Complex functions need to be written out in long form
High-level, slow languages	Python, R, MATLAB	Easy to write, easy to read	Slow in production, often requiring re-writes to another language

Researchers often use high-level languages like Python, R, or MATLAB for research and prototyping because it is relatively quick and easy to try different model specifications using a relatively small dataset.

But when the results need to be implemented at massive scale, with large quantities of data, and speed is of the essence, the resulting algorithms need to be rewritten in a fast low-level language such as C, C++, Java, or Fortran.

The costs of this two-language approach include:

- Cost of translating the algorithm from high-level language to low-level language, including the cost of testing, bug detection, etc.
- Additional bugs introduced during translation
- Communication challenges between researchers using one language and programmers using a different language
- Need to use two languages and often two teams for each update or modification

In the past, engineers, computer scientists and developers believed that a language had to be one type or the other: high productivity (and slow in production), or fast in production (and slow for research and prototyping).

The creators of Julia challenged this belief when they released Julia to the world on February 14, 2012.

They introduced Julia, a new language that delivered all of the benefits of a fast production language + all of the benefits of a high-productivity research language:

- Eliminates the two-language problem
- Allows researchers, scientists, developers, programmers, and engineers to use a single language
- Improves communication and reduces misunderstandings and miscommunications
- Reduces bugs and eliminates an important source of errors
- Results that work in a laboratory setting also work in production
- No need for translation
- Saves time - no code translation
- Eliminates several steps
- Speeds time to market
- Reduces need to maintain infrastructure for multiple languages

Switching to Julia

For users with a lot of legacy code, switching to Julia can require a significant investment of time and energy to realize all the benefits of Julia's speed.

Fortunately, there is a robust ecosystem of packages that allow 2-way interoperability between Julia and C, C++, Python, R, Java, and other languages.

This means that the whole organization doesn't have to be trained on and converted to Julia all at once - significant benefits can be obtained by starting a single pilot project or a small handful of projects in Julia, or converting one or more programs or functions into Julia.

As Julia proves itself within the organization, users will identify opportunities to expand Julia usage, prioritizing areas that would benefit most from the increased speed and improved functionality that Julia offers.

Who Benefits Most from Switching to Julia?

Users who would benefit the most from switching to Julia have one or more of the following characteristics:

- Currently experiencing bottlenecks using Python, R, or other low-speed languages.
- Massive or rapidly expanding data volumes.
- Interest in new languages and processes.
- Currently using both a slow language (such as Python, R, or MATLAB) for prototyping and a fast language (such as C, C++, Java, or Fortran) for production - and would benefit from consolidating to just one language.
- Facing communication and implementation challenges because researchers (modeling and prototyping) and engineers (production) are using different programming languages.
- Would benefit from shorter development time and faster time to market.

Users who would **NOT** benefit from switching to Julia have the following characteristics:

- Their legacy code is working and performing fine. Many high-level languages have standardized workflows using libraries that are themselves written in an efficient language; use-cases that are well-covered by a performant library that someone else has written are unlikely to have dramatic speed improvements.
- They are satisfied with their current speed and time to market. No need to improve.
- They do not have massive or rapidly increasing data volumes.

- They have a lot of experienced programmers in Python, C++, or other languages who do not want to learn a new language.
- They are not experiencing the ‘two language problem’ or communication challenges between researchers (prototyping, modeling, simulation) and engineers (production).

Contact JuliaHub

If your organization would benefit from faster speed, simpler code, composability, and solving the two-language problem, please contact sales@juliahub.com. We will work with you to identify how and why Julia and JuliaHub might be right for you.