JULIA PROGRAMMING

Introduction

dr. ilker arslan

JULIA PROGRAMMING

History of Julia

dr. ilker arslan

Creators of **Julia**

- Jeff Bezanson,
- Stefan Karpinsky,
- Viral B. Shah
- Alan Edelman

“Why we **created Julia?”**

Julia is a programming language developed by Jeff Bezanson, Stefan Karpinsky, Viral B. Shah and Alan Edelman from MIT.

"...We want a language that's open source, with a liberal license.

We want the speed of **C** with the dynamism of **Ruby**.

We want a language that's homoiconic, with true macros like **Lisp**,
but with obvious, familiar mathematical notation like **Matlab**.

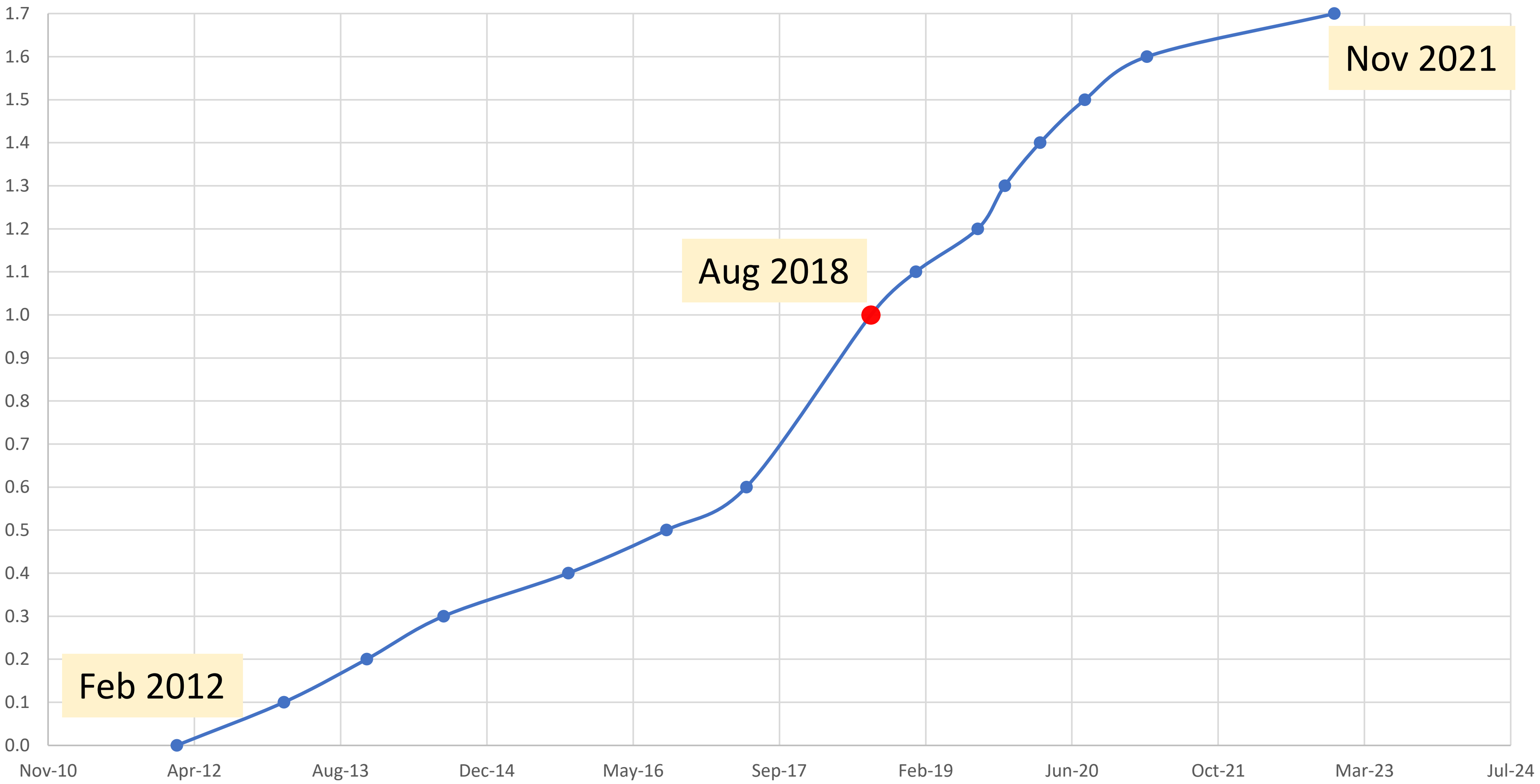
We want something as usable for general programming as **Python**,
as easy for statistics as **R**,
as natural for string processing as **Perl**,
as powerful for linear algebra as **Matlab**,
as good at gluing programs together as the **shell**.

Something that is dirt simple to learn yet keeps the most serious hackers happy.
We want it interactive and we want it compiled.

(Did we mention it should be as fast as **C**?) ..."

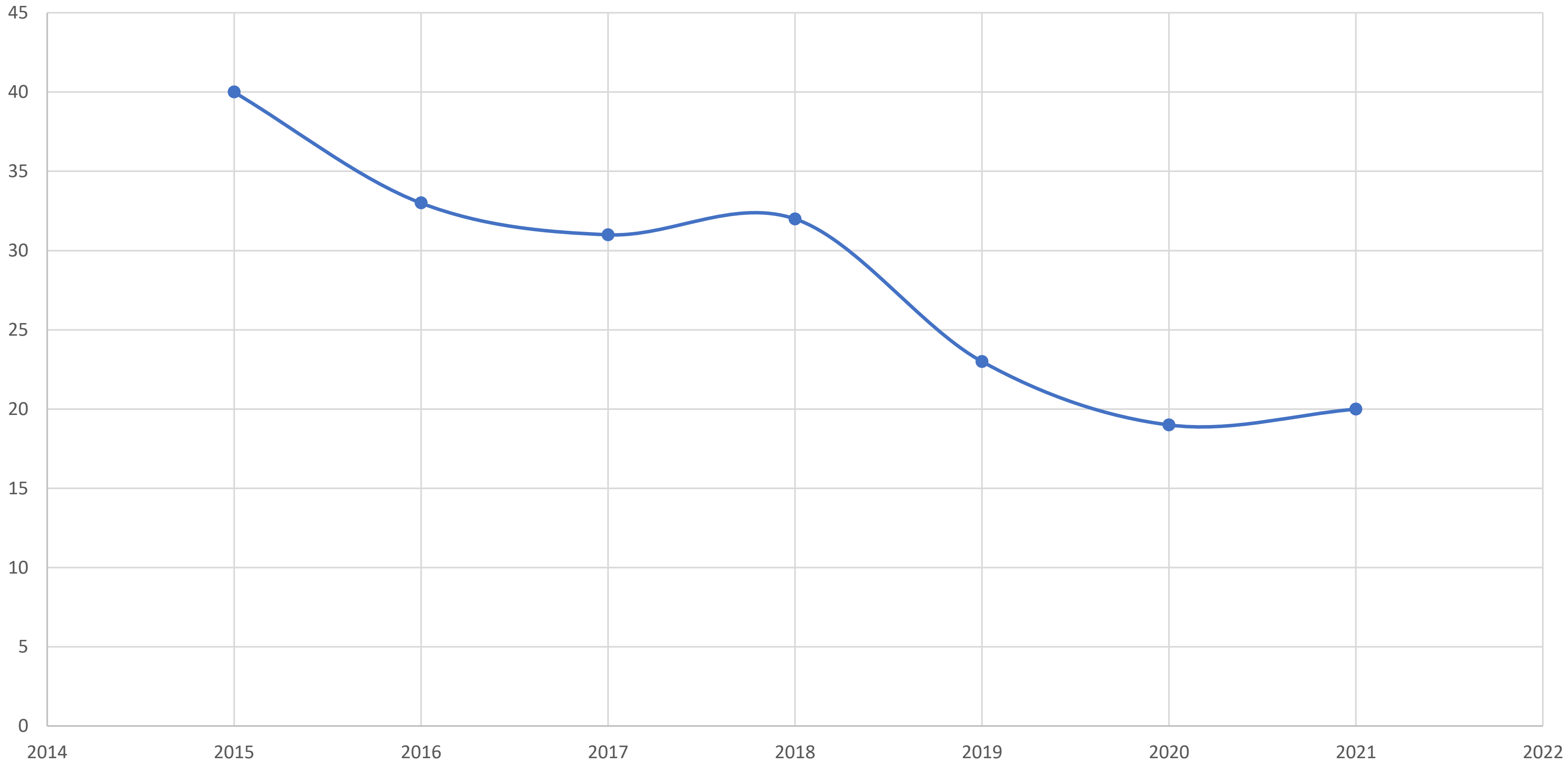
https://julialang.org/blog/2012/02/why-we-created-julia/

Version history



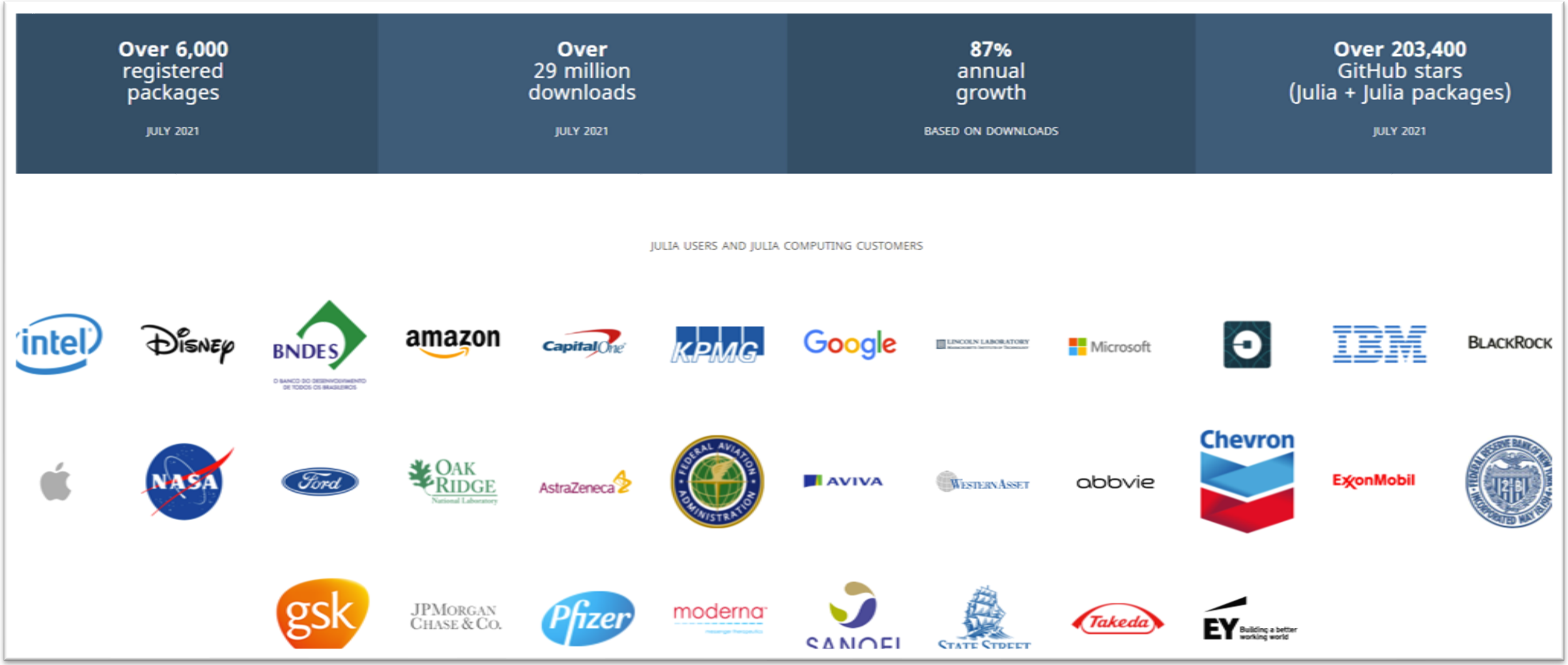
dr. ilker arslan

IEEE Spectrum



dr. ilker arslan

Julia Community



Over 6,000
registered
packages

JULY 2021

Over
29 million
downloads

JULY 2021

87%
annual
growth

BASED ON DOWNLOADS

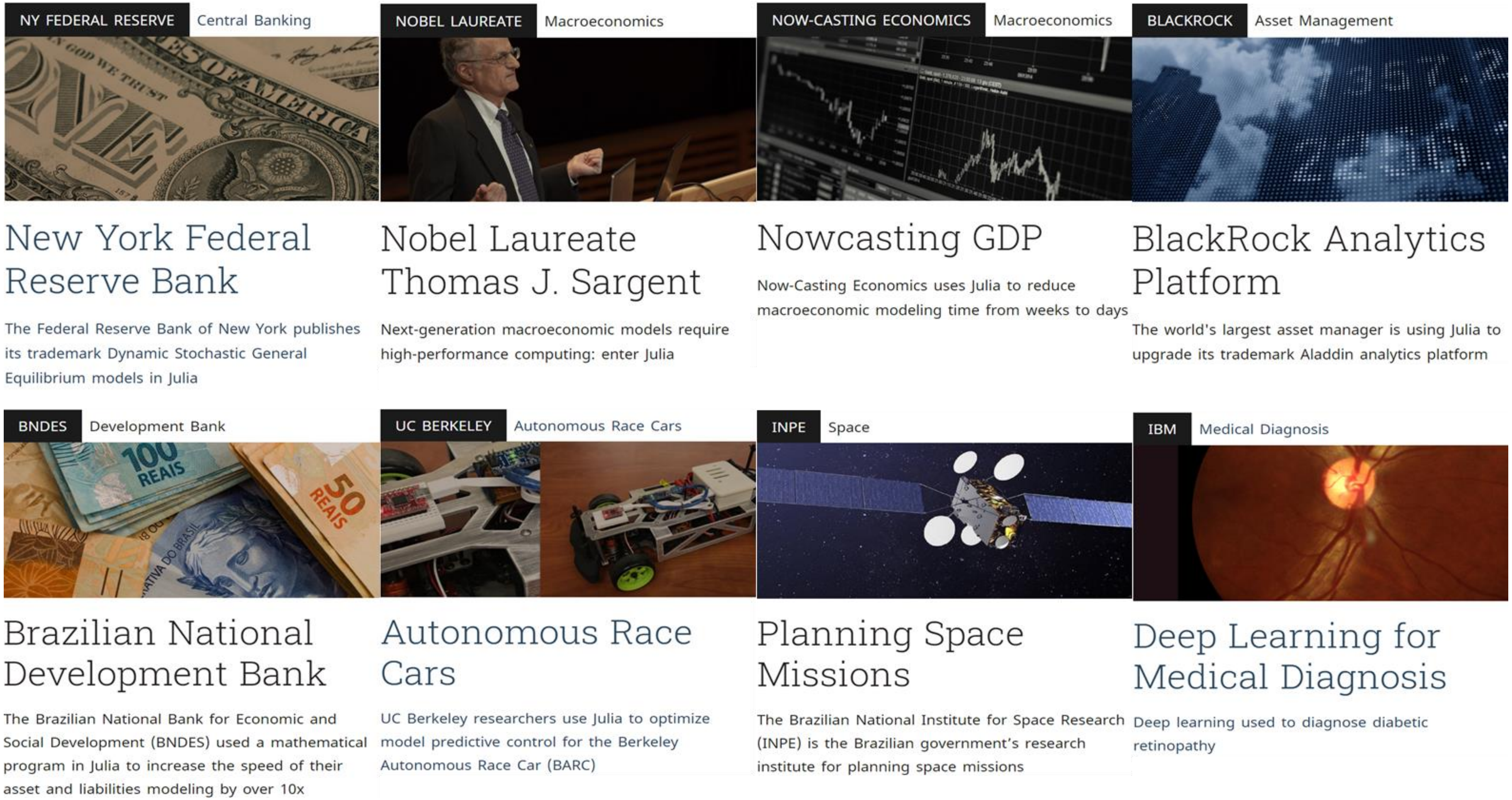
Over 203,400
GitHub stars
(Julia + Julia packages)

JULY 2021

JULIA USERS AND JULIA COMPUTING CUSTOMERS

https://juliacomputing.com/

Case Studies



NY FEDERAL RESERVE | Central Banking

New York Federal Reserve Bank

The Federal Reserve Bank of New York publishes its trademark Dynamic Stochastic General Equilibrium models in Julia

NOBEL LAUREATE | Macroeconomics

Nobel Laureate Thomas J. Sargent

Next-generation macroeconomic models require high-performance computing: enter Julia

NOW-CASTING ECONOMICS | Macroeconomics

Nowcasting GDP

Now-Casting Economics uses Julia to reduce macroeconomic modeling time from weeks to days

BLACKROCK | Asset Management

BlackRock Analytics Platform

The world's largest asset manager is using Julia to upgrade its trademark Aladdin analytics platform

BNDES | Development Bank

Brazilian National Development Bank

The Brazilian National Bank for Economic and Social Development (BNDES) used a mathematical program in Julia to increase the speed of their asset and liabilities modeling by over 10x

UC BERKELEY | Autonomous Race Cars

Autonomous Race Cars

UC Berkeley researchers use Julia to optimize model predictive control for the Berkeley Autonomous Race Car (BARC)

INPE | Space

Planning Space Missions

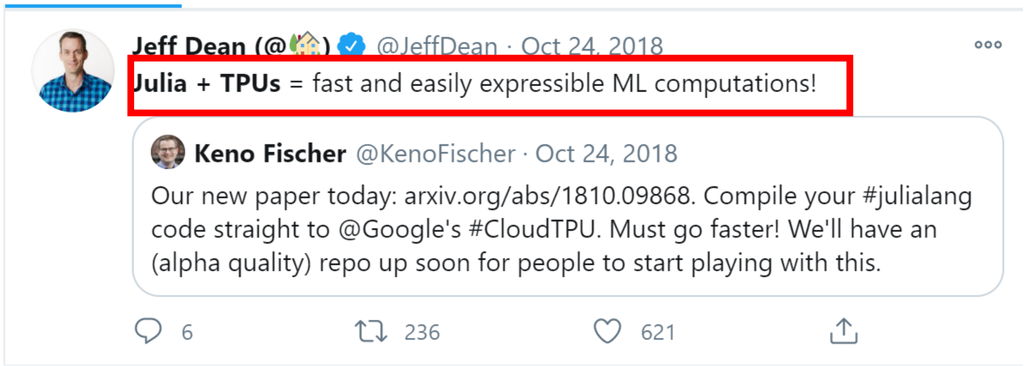
The Brazilian National Institute for Space Research (INPE) is the Brazilian government's research institute for planning space missions

IBM | Medical Diagnosis

Deep Learning for Medical Diagnosis

Deep learning used to diagnose diabetic retinopathy

https://juliacomputing.com/case-studies/

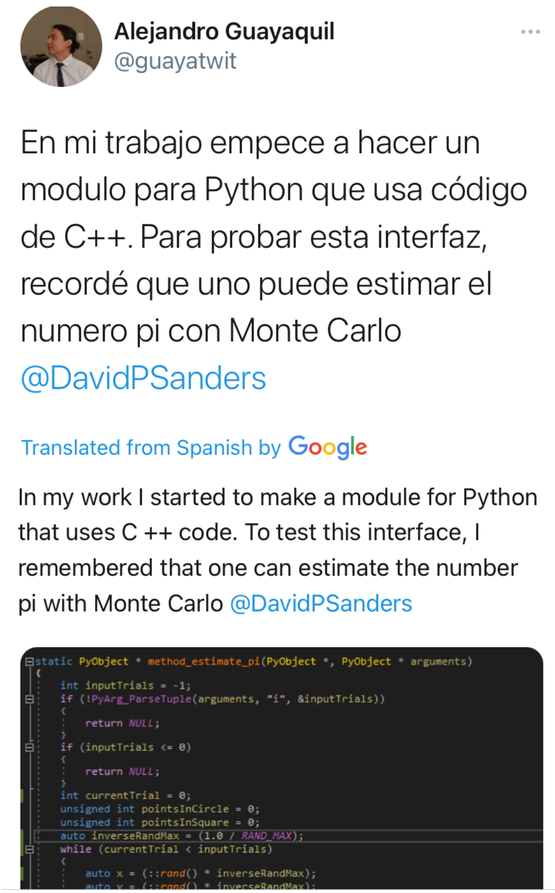


Jeff Dean (@🏡) @JeffDean · Oct 24, 2018

Julia + TPUs = fast and easily expressible ML computations!

Keno Fischer @KenoFischer · Oct 24, 2018

Our new paper today: arxiv.org/abs/1810.09868. Compile your #julialang code straight to @Google's #CloudTPU. Must go faster! We'll have an (alpha quality) repo up soon for people to start playing with this.



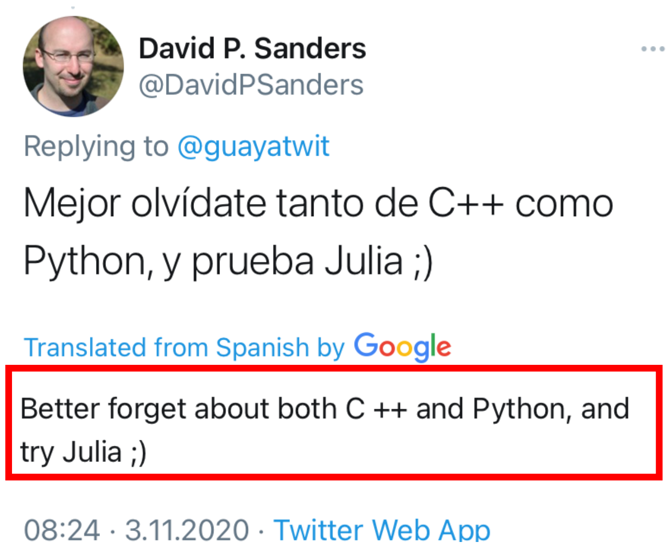
Alejandro Guayaquil
@guayatwit

En mi trabajo empece a hacer un modulo para Python que usa código de C++. Para probar esta interfaz, recordé que uno puede estimar el numero pi con Monte Carlo @DavidPSanders

Translated from Spanish by Google

In my work I started to make a module for Python that uses C ++ code. To test this interface, I remembered that one can estimate the number pi with Monte Carlo @DavidPSanders

```
static PyObject * method_estimate_pi(PyObject *, PyObject * arguments)
{
    int inputTrials = -1;
    if (!PyArg_ParseTuple(arguments, "i", &inputTrials))
    {
        return NULL;
    }
    if (inputTrials <= 0)
    {
        return NULL;
    }
    int currentTrial = 0;
    unsigned int pointsInCircle = 0;
    unsigned int pointsInSquare = 0;
    auto inverseRandMax = (1.0 / RAND_MAX);
    while (currentTrial < inputTrials)
    {
        auto x = (::rand() * inverseRandMax);
        auto y = (::rand() * inverseRandMax);
```



David P. Sanders
@DavidPSanders

Replying to @guayatwit

Mejor olvídate tanto de C++ como Python, y prueba Julia ;)

Translated from Spanish by Google

Better forget about both C ++ and Python, and try Julia ;)

08:24 · 3.11.2020 · Twitter Web App

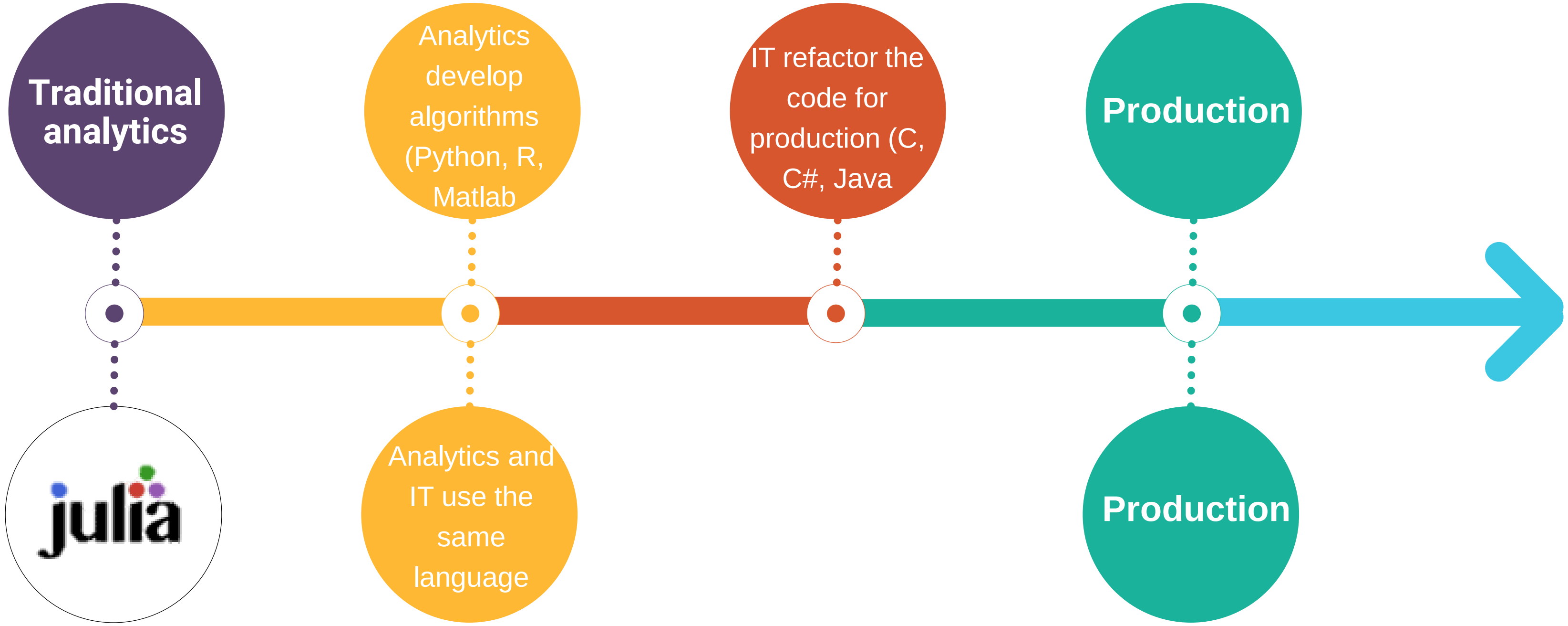
dr. ilker arslan

JULIA PROGRAMMING

Why Julia?

dr. ilker arslan

Julia solves **two-language problem**



Traditional analytics

Analytics develop algorithms (Python, R, Matlab

IT refactor the code for production (C, C#, Java

Production

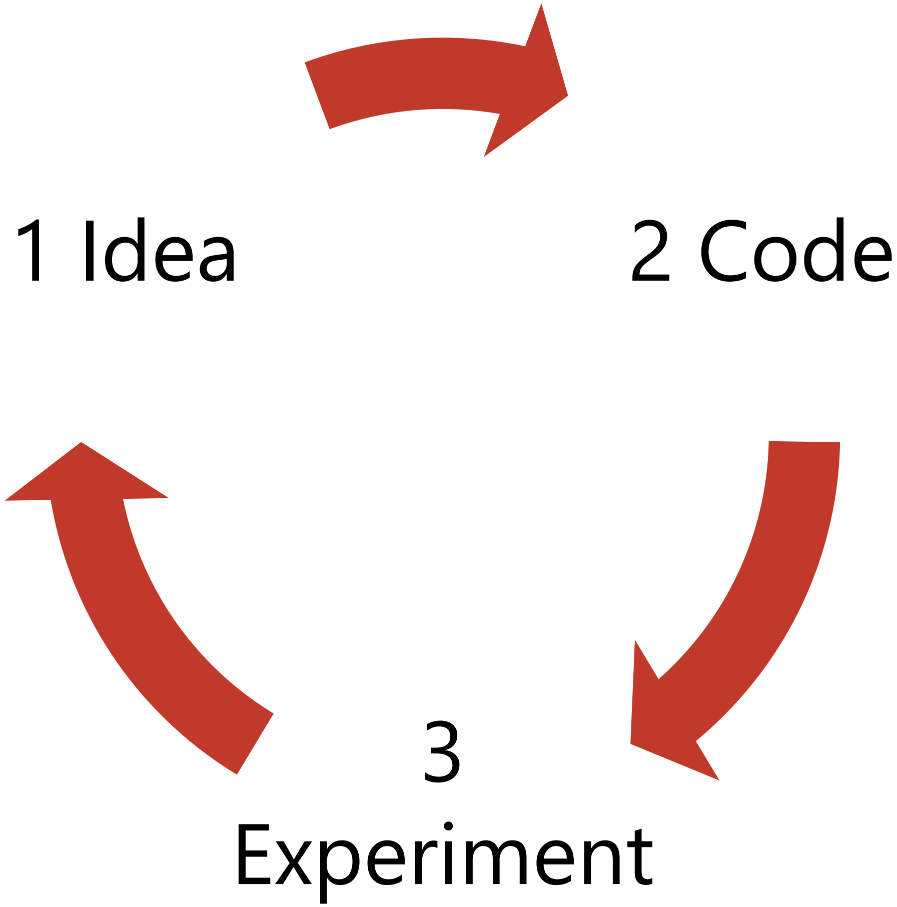
julia

Analytics and IT use the same language

Production

https://www2.slideshare.net/ViralBShah1/julia-a-modern-language-for-software-20/2

Development Phases



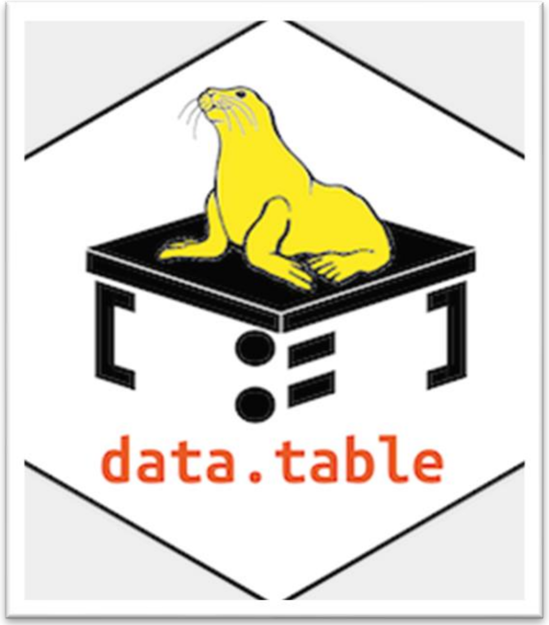
1 Idea

2 Code

3 Experiment

Figure by Andrew Ng, Coursera Deep Learning Specialization

https://numpy.org/



https://rdatatable.gitlab.io/data.table/

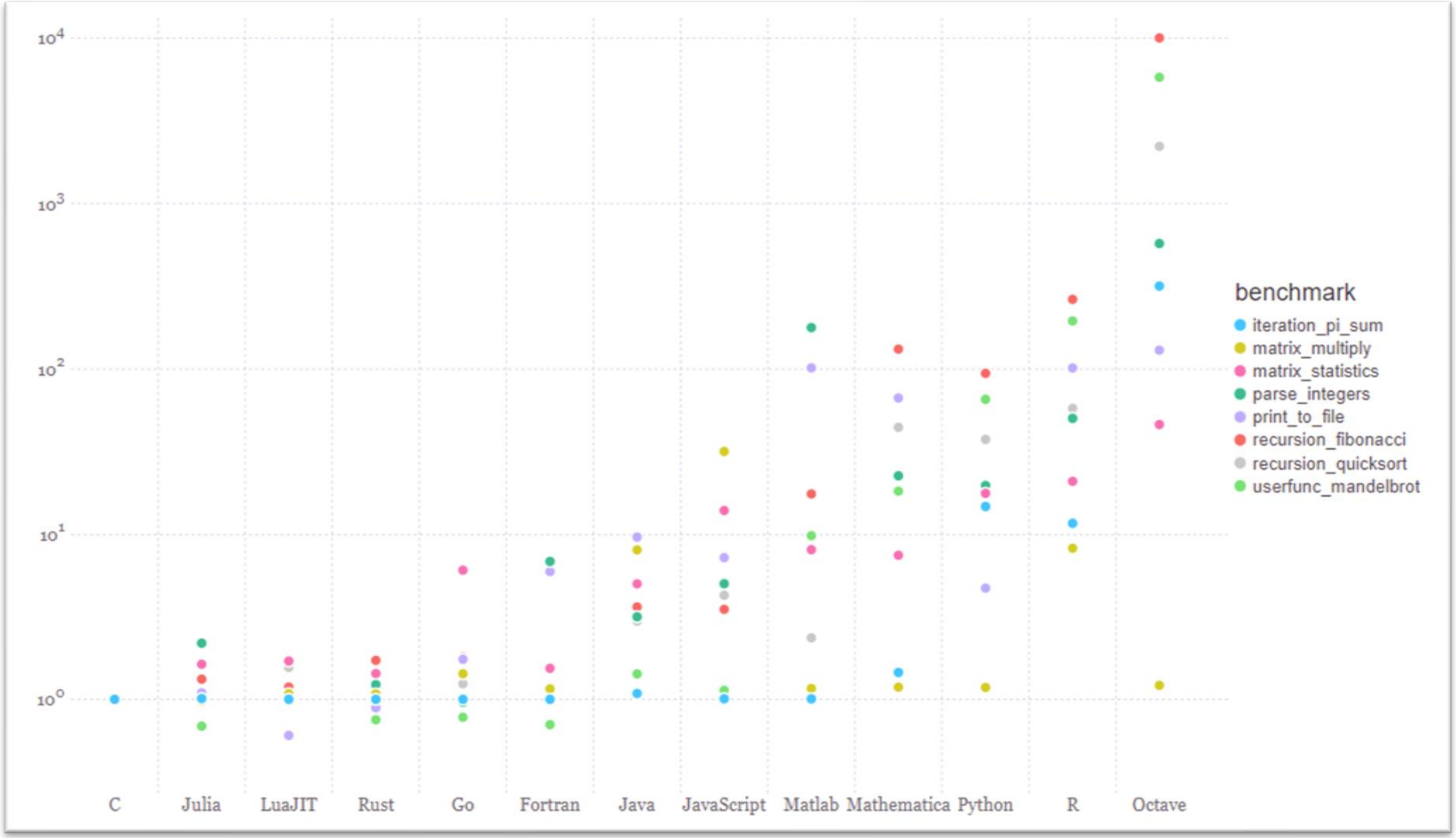
Julia is **Fast**

Adding 10 million numbers

Program	**Mean Duration (ms)**
Julia hand-written simd	3.0
Julia built-in	3.0
C -ffast-math	5.1
Python numpy	8.0
Julia hand-written	8.9
C	9.1
Python built-in	536.9
Python hand-written	760.5

https://github.com/JuliaAcademy/JuliaTutorials/blob/main/introductory-tutorials/intro-to-julia/09.%20Julia%20is%20fast.ipynb

Benchmark Algorithms

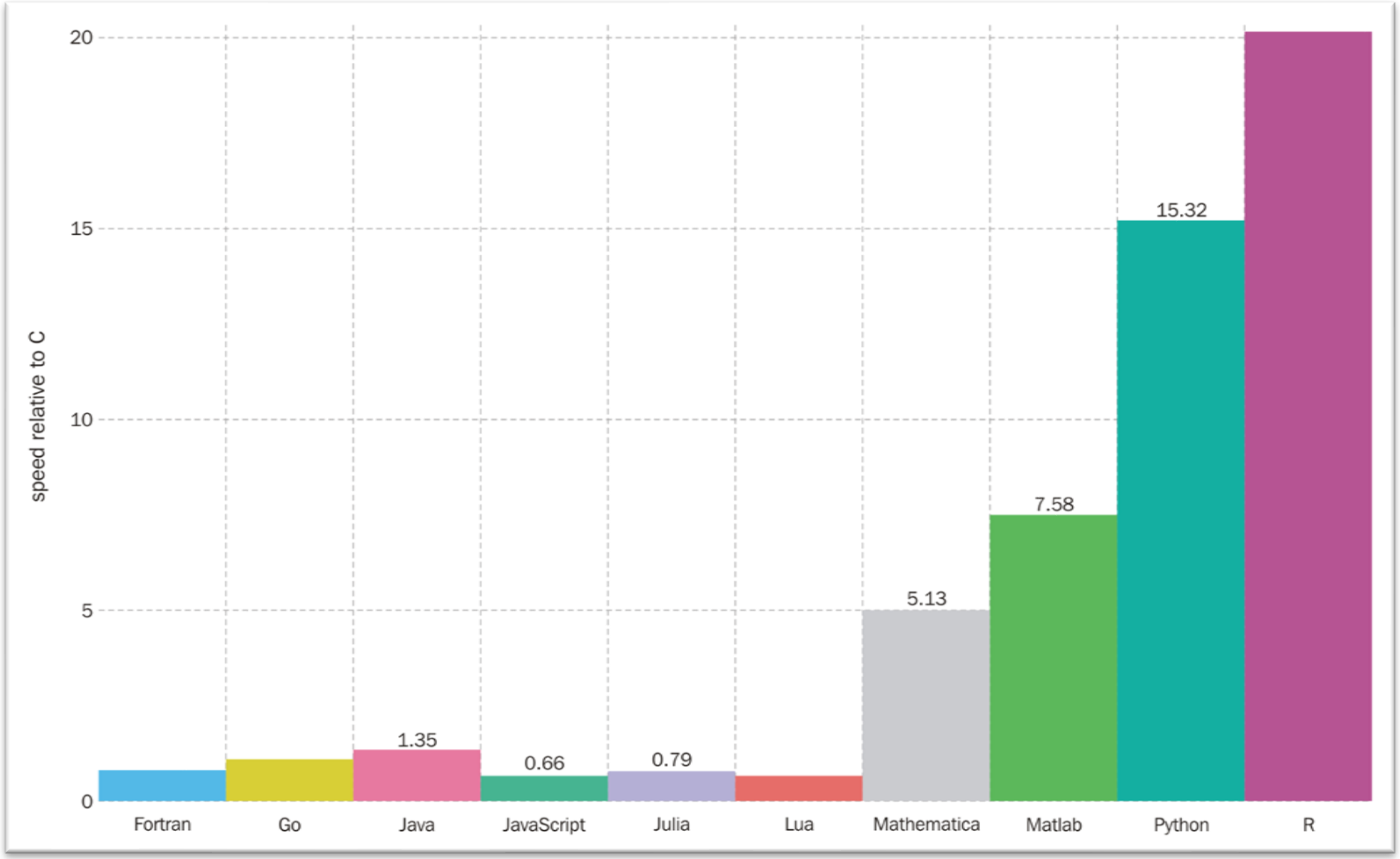


https://julialang.org/benchmarks/

dr. ilker arslan

Computation of Mandelbrot Set

$$f_c(z) = z^2 + c$$



Julia High Performance, Avik Sengupta, Alan Edelman, Computation of Mandelbrot Set

Julia

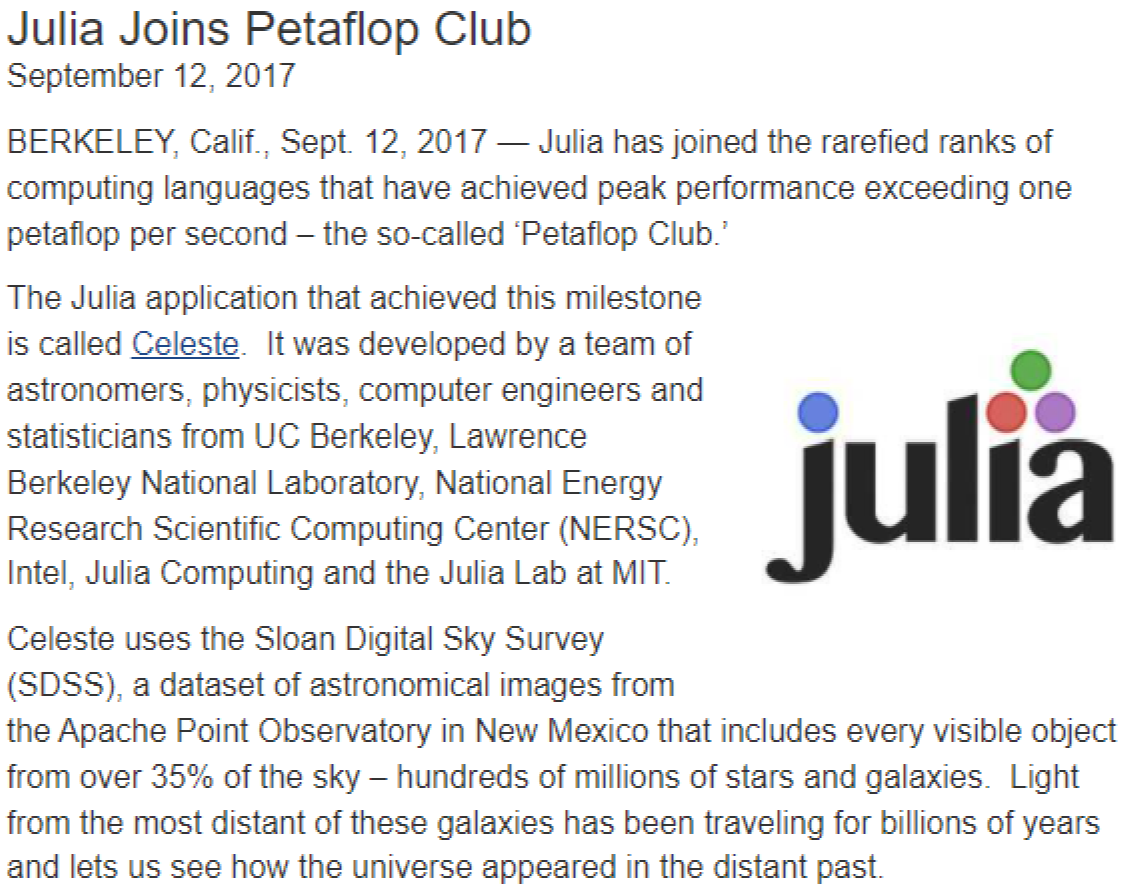
```
function mandel(c)
    z = c
    maxiter = 80
    for n in 1:maxiter
        if abs(z) > 2
        return n - 1
    end
    z = z^2 + c
    end
    return maxiter
end
```

C

```
int mandel(double complex z) {
  int maxiter = 80;
  double complex c = z;
  for (int n = 0; n < maxiter; ++n){
    if (cabs(z) > 2.0) {
    return n;
    }
    z = z*z+c;
  }
  return maxiter;
}
```

dr. ilker arslan

Petaflop Club



Julia Joins Petaflop Club

September 12, 2017

BERKELEY, Calif., Sept. 12, 2017 — Julia has joined the rarefied ranks of computing languages that have achieved peak performance exceeding one petaflop per second – the so-called 'Petaflop Club.'

The Julia application that achieved this milestone is called Celeste. It was developed by a team of astronomers, physicists, computer engineers and statisticians from UC Berkeley, Lawrence Berkeley National Laboratory, National Energy Research Scientific Computing Center (NERSC), Intel, Julia Computing and the Julia Lab at MIT.

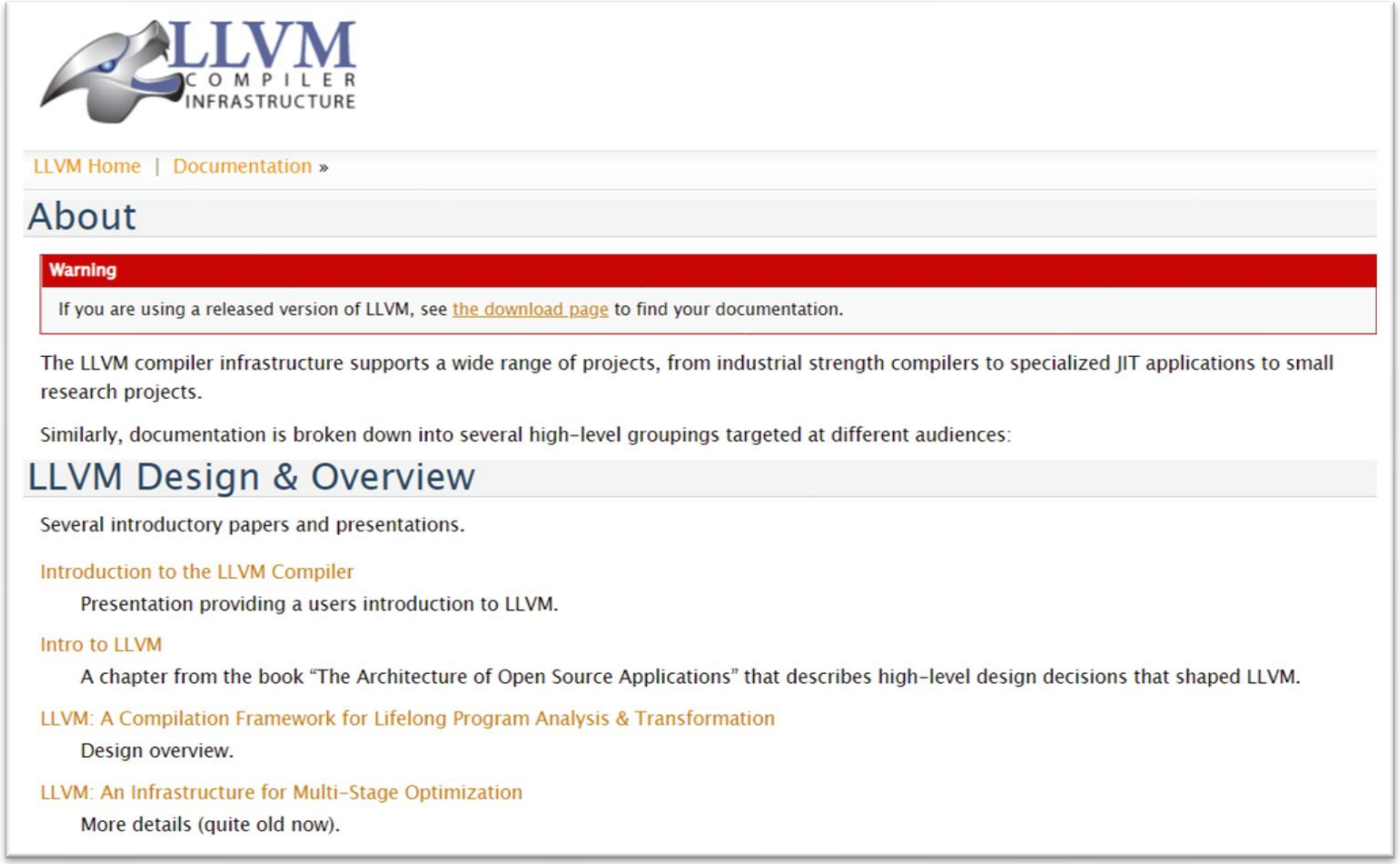
Celeste uses the Sloan Digital Sky Survey (SDSS), a dataset of astronomical images from the Apache Point Observatory in New Mexico that includes every visible object from over 35% of the sky – hundreds of millions of stars and galaxies. Light from the most distant of these galaxies has been traveling for billions of years and lets us see how the universe appeared in the distant past.

HPC wire

https://www.hpcwire.com/off-the-wire/julia-joins-petaflop-club/

Julia is a Just in Time (JIT) compiled language

LLVM



LLVM Home | Documentation »

About

Warning

If you are using a released version of LLVM, see the download page to find your documentation.

The LLVM compiler infrastructure supports a wide range of projects, from industrial strength compilers to specialized JIT applications to small research projects.

Similarly, documentation is broken down into several high-level groupings targeted at different audiences:

LLVM Design & Overview

Several introductory papers and presentations.

Introduction to the LLVM Compiler

Presentation providing a users introduction to LLVM.

Intro to LLVM

A chapter from the book "The Architecture of Open Source Applications" that describes high-level design decisions that shaped LLVM.

LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation

Design overview.

LLVM: An Infrastructure for Multi-Stage Optimization

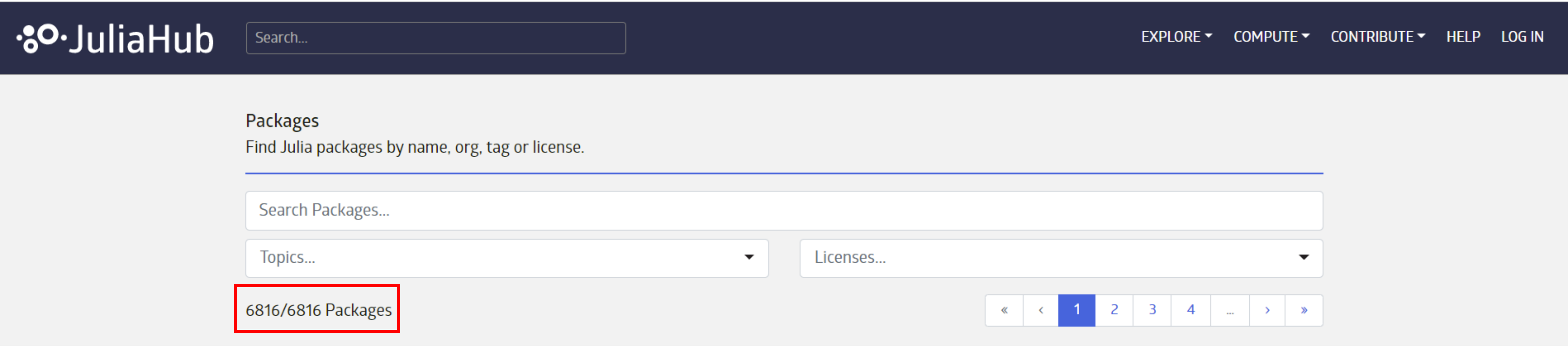
More details (quite old now).

LLVM currently supports compiling of Ada, C, C++, D, Delphi, Fortran, Haskell, Julia, Objective-C, Rust, and Swift using various front ends.

www.llvm.org/docs/

Rich Package Ecosystem

https://juliahub.com/ui/packages



Packages

Find Julia packages by name, org, tag or license.

6816/6816 Packages