

# The (Software) Engineering of Julia's SciML



Chris Rackauckas

VP of Modeling and Simulation, JuliaHub / Research Affiliate MIT / Others

# What is the SciML Open Source Organization?

---

The SciML Open Source organization is a non-profit organization, part of the NumFOCUS affiliate libraries, which builds and supports the development of packages in Julia, Python, and R for scientific simulation and scientific machine learning.

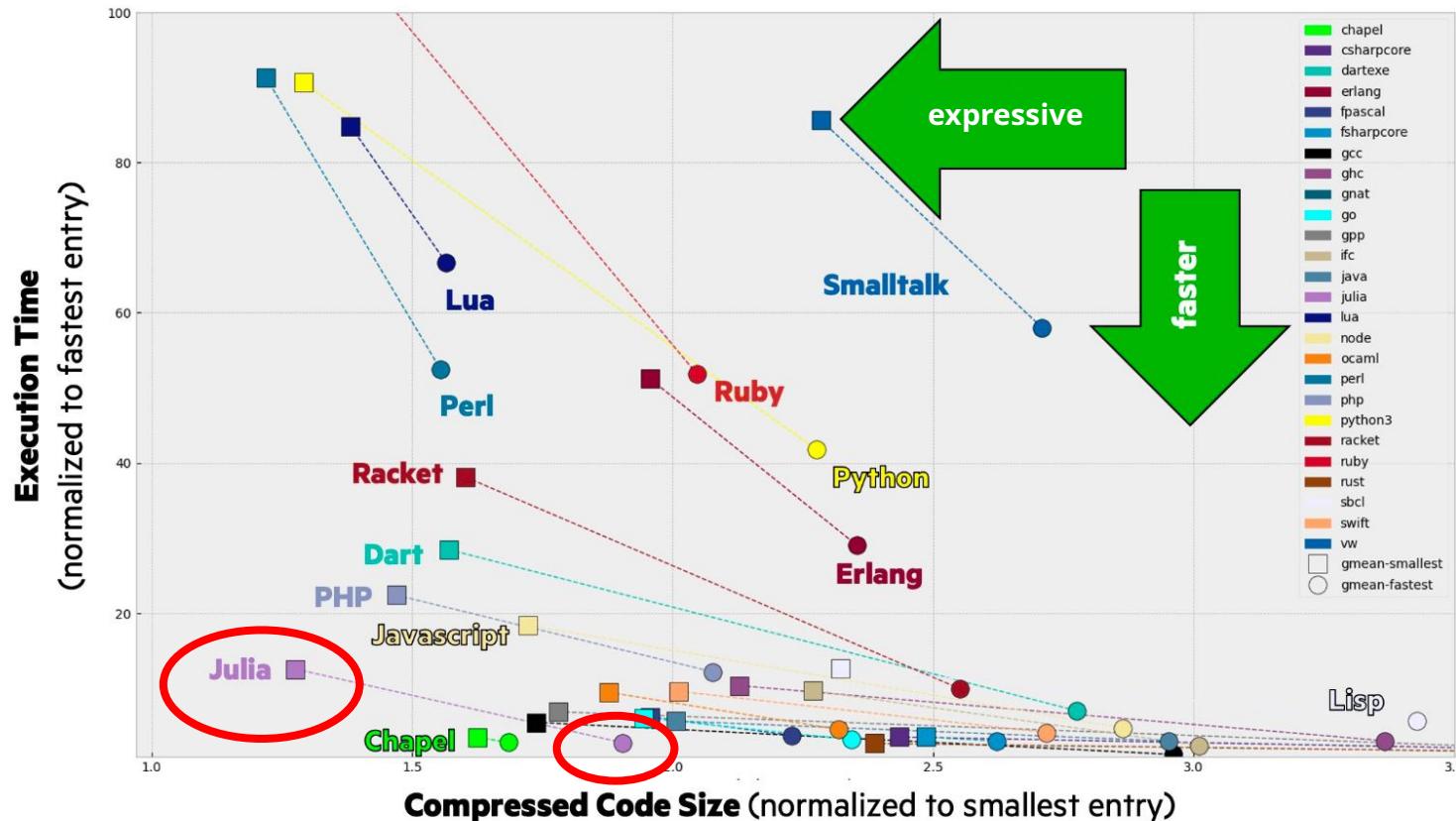
- Over 100 Github repositories, many of which are 10's of packages themselves.
  - Totals ~200 Julia packages. All MIT open source licensed.
- 20,000+ Github stars across the set of packages
- ~100+ unique contributors monthly (variable month to month)
- ~20 core maintainers owning some aspect of the project
- ~5-10 summer students and other trainees per year
  - Many maintainers from the ~20 folks associated with the MIT Julia Lab (and alumni!)
  - Many companies have spun off (PumasAI, JuliaHub, Neuroblox, etc.) which “house” a good number of maintainers with full time jobs related to their contributions.
  - ~10 grant submissions per year related to trying to get more folks into the SciML organization, usually to MIT though sometimes facilitated through one of the contributor’s commercial entities or to the non-profit itself
- I (Chris Rackauckas) am the BDFL, i.e. someone stupid enough to be a maintainer on all of the repositories in order to give it a unifying force

Tl;dr, it will be very hard to summarize all of the activity going on in 30 minutes but I will try my best. If I leave off a topic you like, I’m sorry but many cuts have to have been made. **About ¼ of the organization’s work is represented here.** I hope to start a recurring series on Youtube that better tracks these updates.

**First of all, what is the core topic the open source software is about?**



# Julia is a high-level language that is faster than Matlab, Python, R



# Foundation: Fast Differential Equation Solvers

1. Speed
2. Stability
3. Stochasticity
4. Adjoint and Inference
5. Parallelism



## DifferentialEquations.jl is generally:

- 50x faster than SciPy
- 50x faster than MATLAB
- 100x faster than R's deSolve

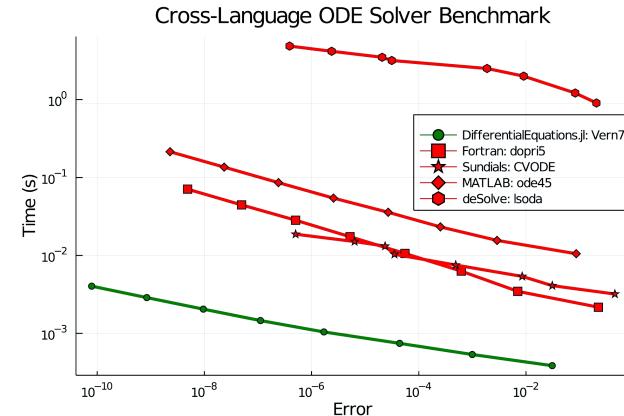


<https://github.com/SciML/SciMLBenchmarks.jl>

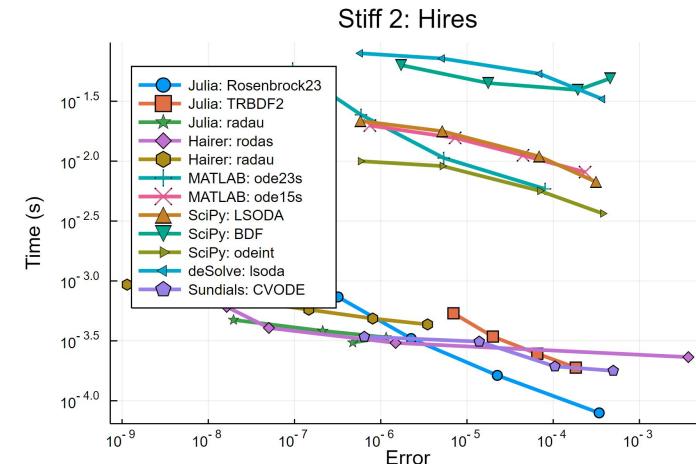
Rackauckas, Christopher, and Qing Nie. "DifferentialEquations.jl—a performant and feature-rich ecosystem for solving differential equations in Julia." *Journal of Open Research Software* 5.1 (2017).

Rackauckas, Christopher, and Qing Nie. "Confederated modular differential equation APIs for accelerated algorithm development and benchmarking." *Advances in Engineering Software* 132 (2019): 1-6.

## Non-Stiff ODE: Rigid Body System

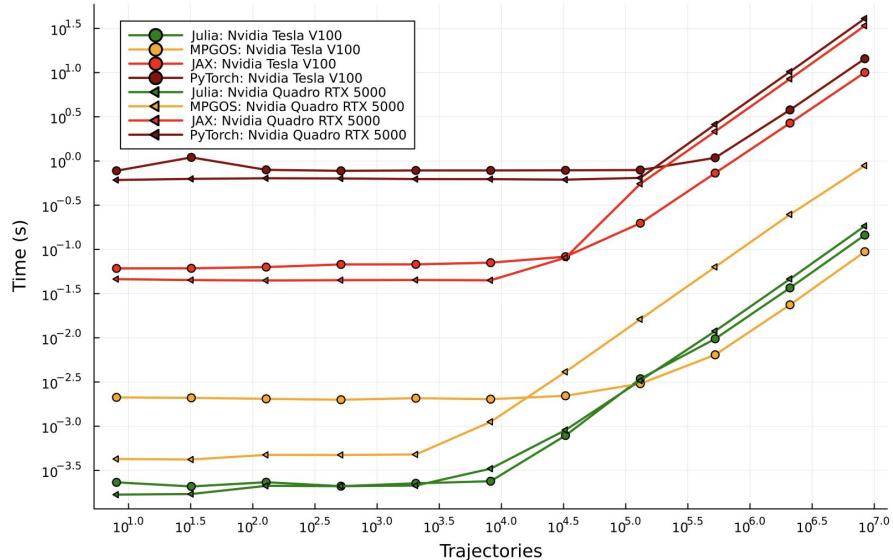


## 8 Stiff ODEs: Hires Chemical Reaction Network



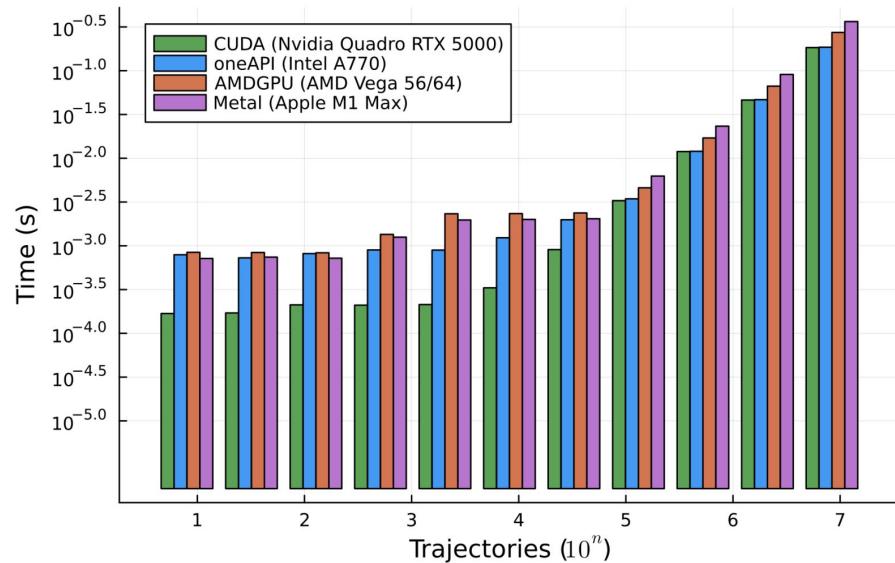
# New Parallelized GPU ODE Parallelism: 20x-100x Faster than Jax and PyTorch

Lorenz Problem: 1000 fixed time-steps



Utkarsh, U., Churavy, V., Ma, Y., Besard, T., Srisuma, P., Gymnich, T., ... & Rackauckas, C. (2024). Automated translation and accelerated solving of differential equations on multiple GPU platforms. *Computer Methods in Applied Mechanics and Engineering*, 419, 116591.

Performance Comparison with different GPU backends



Matches State of the Art on CUDA, but also works with AMD, Intel, and Apple GPUs



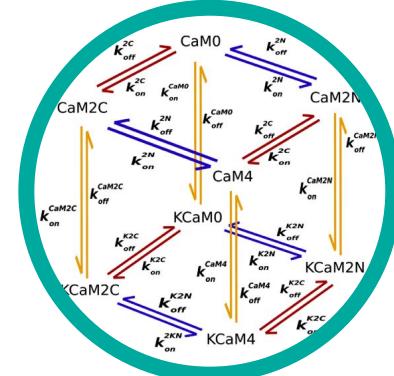
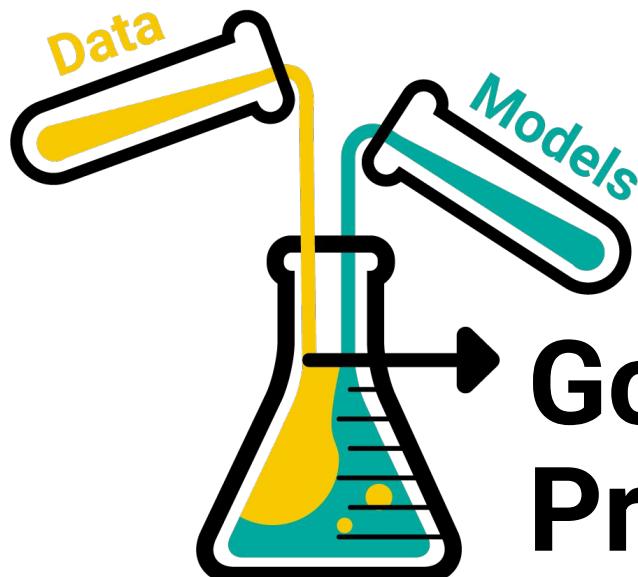
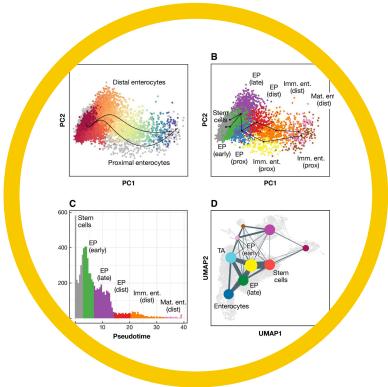
# An Incomplete List of the SciML Common Interface for Julia Equation Solvers

---

- `LinearSolve.jl`  $A(p)x = b$
- `NonlinearSolve.jl`  $f(u, p) = 0$
- `DifferentialEquations.jl`  $u' = f(u, p, t)$
- `Integrals.jl`  $\int_{lb}^{ub} f(t, p) dt$
- `Optimization.jl`  $\text{minimize } f(u, p)$   
 $\text{subject to } g(u, p) \leq 0, h(u, p) = 0$

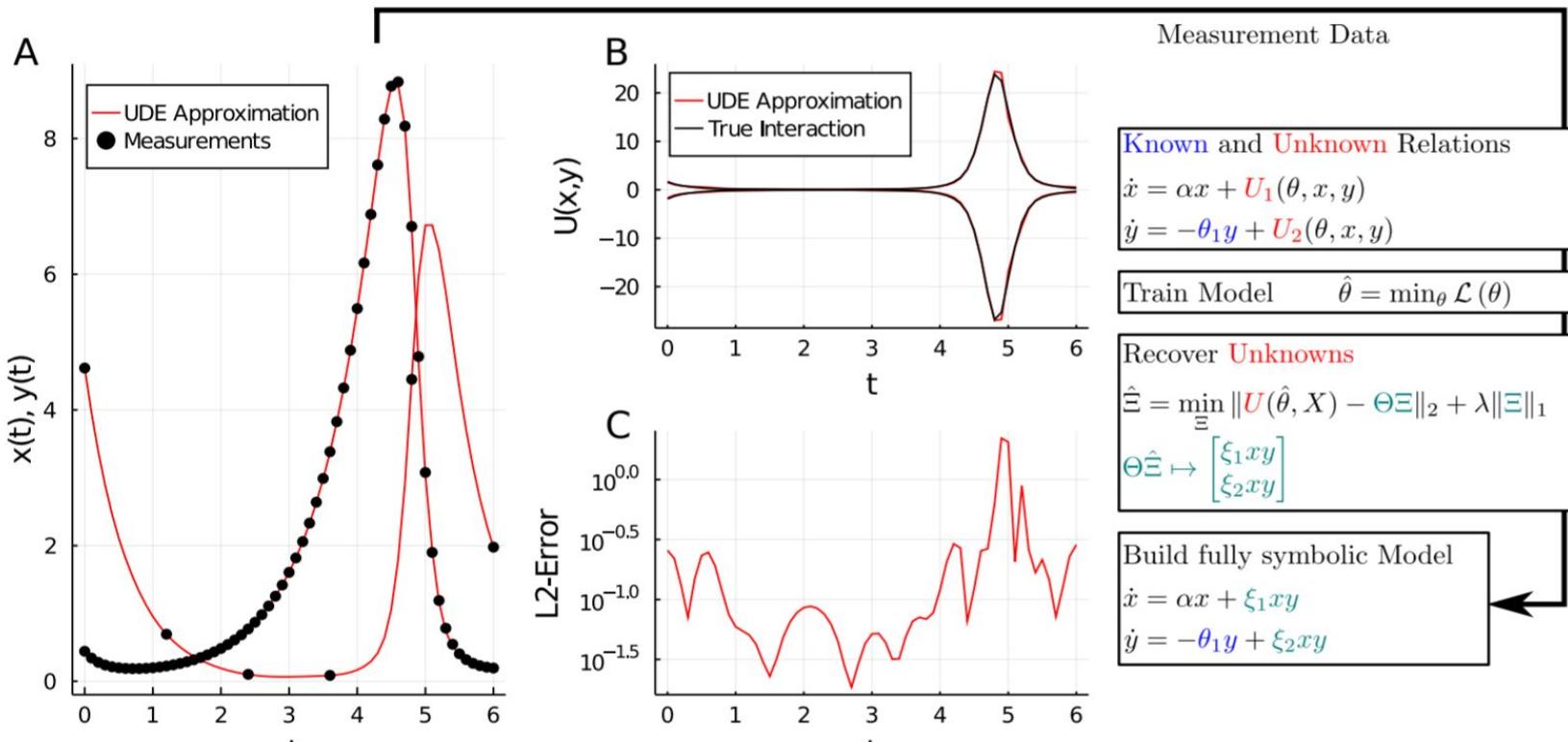
# Scientific Machine Learning is model-based data-efficient machine learning

How do we simultaneously use both sources of knowledge?



Good  
Predictions

# Universal (Approximator) Differential Equations



# Accurate Model Extrapolation Mixing in Physical Knowledge

Upon denoting  $\mathbf{x} = (\phi, \chi, p, e)$ , we propose the following family of UDEs to describe the two-body relativistic dynamics:

$$\dot{\phi} = \frac{(1 + e \cos(\chi))^2}{M p^{3/2}} (1 + \mathcal{F}_1(\cos(\chi), p, e)), \quad (5a)$$

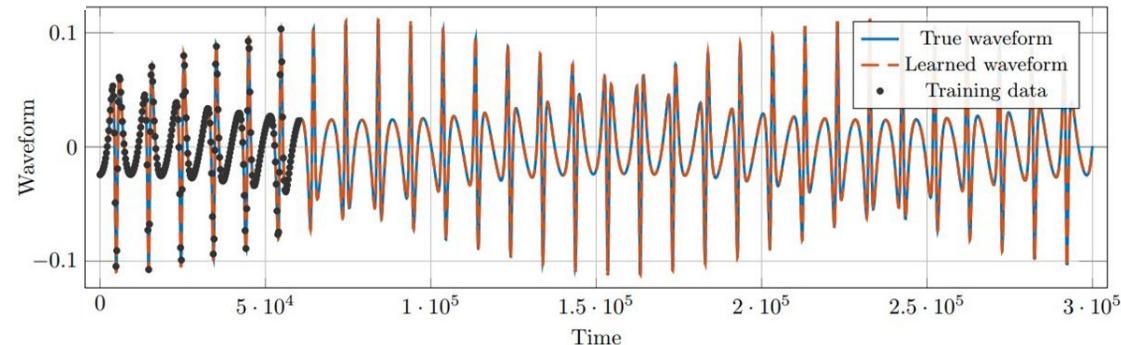
$$\dot{\chi} = \frac{(1 + e \cos(\chi))^2}{M p^{3/2}} (1 + \mathcal{F}_2(\cos(\chi), p, e)), \quad (5b)$$

$$\dot{p} = \mathcal{F}_3(p, e), \quad (5c)$$

$$\dot{e} = \mathcal{F}_4(p, e), \quad (5d)$$

Automated discovery of geodesic equations from LIGO black hole data: run the code yourself!

<https://docs.sciml.ai/Overview/stable/showcase/blackhole/>



Keith, B., Khadse, A., & Field, S. E. (2021). Learning orbital dynamics of binary black hole systems from gravitational wave measurements. *Physical Review Research*, 3(4), 043101.



# **Technical Story: The Composability Model**



# What is the SciML Open Source Organization?

The screenshot shows the SciML website's main navigation bar with links for HOME, MODELING, SOLVERS, ANALYSIS, MACHINE LEARNING, and DEVELOPER TOOLS. Below the navigation is a search bar. The page content is organized into four columns: EQUATION SOLVERS (LinearSolve, NonlinearSolve, DifferentialEquations, Integrals, Optimization, JumpProcesses), INVERSE PROBLEMS / ESTIMATION (SciMLSensitivity, DiffEqParamEstim, DiffEqBayes), PDE SOLVERS (MethodOfLines, NeuralPDE, NeuralOperators, FEniCS, HighDimPDE, DiffEqOperators), and THIRD-PARTY PDE SOLVERS (Trixi, Gridap, ApproxFun, VoronoiFVM).

or the highest performance and parallel implementations one can find.

## Where to Start?

### Getting Started

Getting Started with Julia's SciML

New User Tutorials >

Comparison With Other Tools >

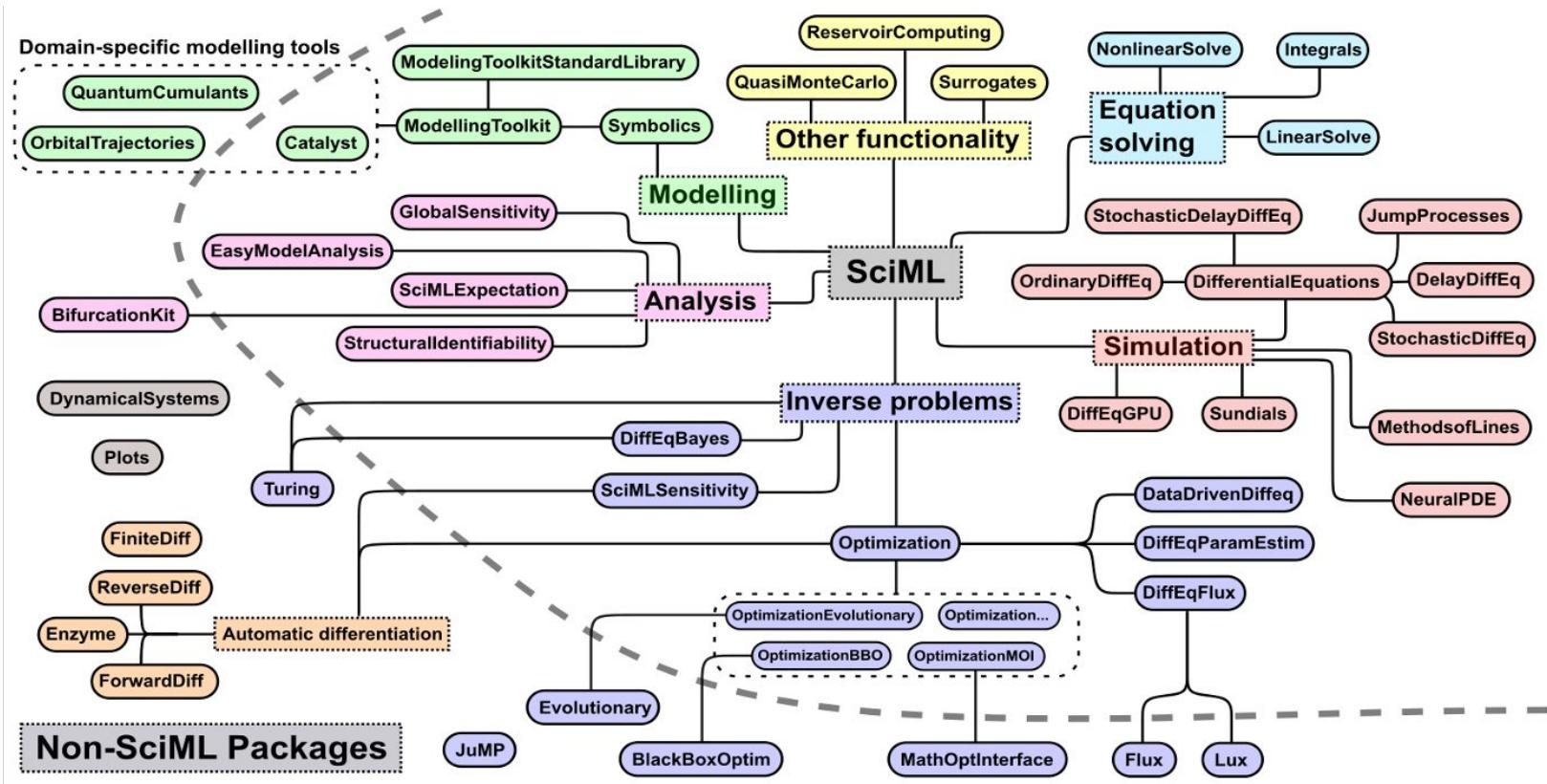
Version v0.2 ▾

## Where to Start?

- Want to get started running some code? Check out the [Getting Started](#) tutorials.
- What is SciML? Check out our [Overview](#).
- Want to see some cool end-to-end examples? Check out the [Extended Tutorials](#).
- Curious about our performance claims? Check out [the SciML Open Benchmarks](#).



# What is the SciML Open Source Organization?



# Composability: key element of language design

```
julia> using Tensors: Vec  
  
julia> struct Planet  
    m::Float64  
    v::Vec{2, Float64}  
end  
  
julia> energy(p::Planet) = 1/2 * p.m * (p.v .* p.v);  
  
julia> total_energy(ps::AbstractVector{Planet}) =  
    sum(energy, planets);  
  
julia> planets = [Planet(3.30*10^23, Vec(35.7, -31.94)),  
    Planet(4.87*10^24, Vec(18.6, -16.67)),  
    Planet(5.97*10^24, Vec(22.2, -19.87))];  
  
julia> sizeof(planets) # 3 * (8 + 2*8) = 72  
72  
  
julia> total_energy(planets)  
9.111808666594879e21
```



Memory

```
julia> using BenchmarkTools  
  
julia> @btime total_energy(planets);  
39.023 ns  
✓ CPU
```

```
julia> using CUDA: CuArray  
  
julia> planets_cuda = CuArray(planets);  
  
julia> total_energy(planets);  
9.111808666594879e21
```



GPU

# Composability, going further

```
julia> using Unitful, Unitful.DefaultSymbols  
  
julia> struct Planet{M<:Unitful.Mass, V<:Unitful.Velocity}  
    m::M  
    v::Vec{2, V}  
end  
  
julia> planets = [Planet(3.30*10^23*kg, Vec(35.7km/s, -31.94km/s)),  
                  Planet(4.87*10^24*kg, Vec(18.6km/s, -16.67km/s)),  
                  Planet(5.97*10^24*kg, Vec(22.2km/s, -19.87km/s))];  
  
julia> sizeof(planets) # 3 * (8 + 2*8) = 72    ✓ Memory  
72  
  
julia> total_energy(planets)  
9.111808666594879e21 kg km^2 s^-2  
  
julia> total_energy(planets) |> GJ  
9.11180866659488e18 GJ  
  
julia> @btime total_energy($planets);    ✓ CPU  
  40.948 ns
```

```
julia> planets_cuda = CuArray(planets);  
  
julia> total_energy(planets_cuda)  
9.111808666594879e21 kg km^2 s^-2
```

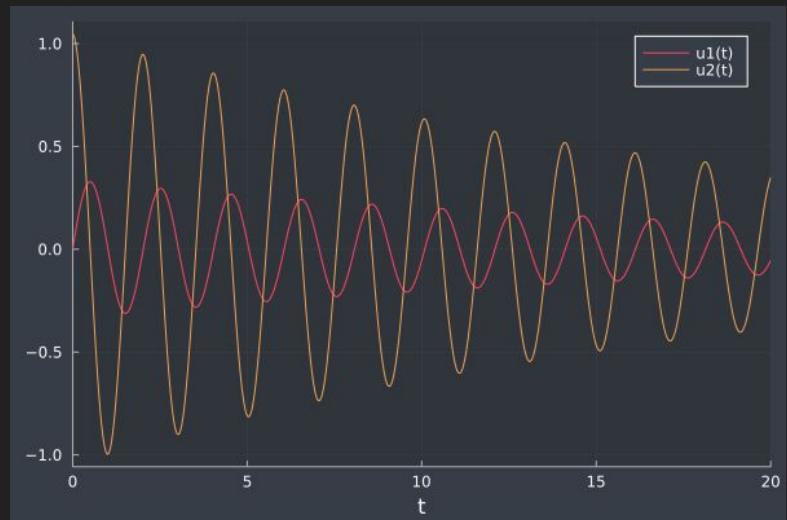
✓ GPU

# Code is truly composable

```
using DifferentialEquations, Plots

function simplependulum(du,u,p,t)
    g, L = p
    θ, dθ = u
    du[1] = dθ
    du[2] = -g/L * sin(θ) - 0.1dθ
end

g      = 9.79          # Gravitational constant
L      = 1.00          # Length of the pendulum
u₀    = [0, π/3]        # Initial angle and velocity
prob  = ODEProblem(simplependulum, u₀, (0, 20), (g, L))
sol   = solve(prob)
plot(sol)
```

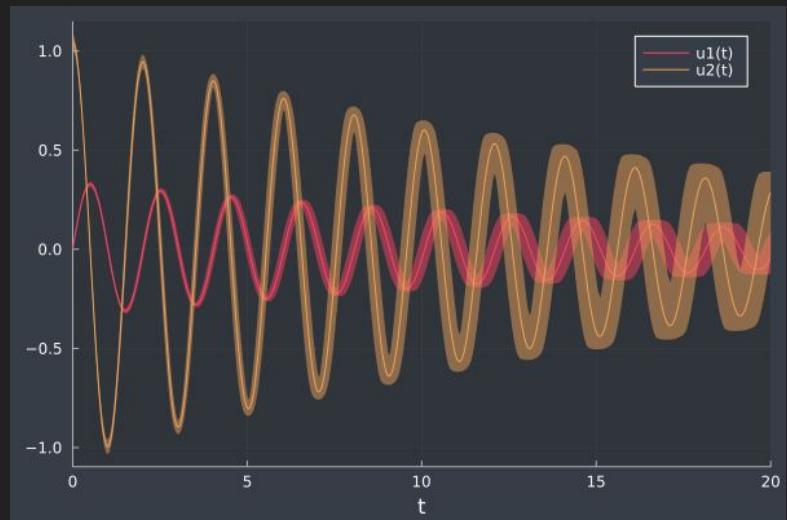


# Code is truly composable – Uncertainty propagation

```
using DifferentialEquations, Plots, MonteCarloMeasurements

function simplependulum(du,u,p,t)
    g, L = p
    θ, dθ = u
    du[1] = dθ
    du[2] = -g/L * sin(θ) - 0.1dθ
end

g      = 9.79 ± 0.02 # Gravitational constant
L      = 1.00 ± 0.02 # Length of the pendulum
u₀    = [0, π/3 ± 0.02] # Initial angle and velocity
prob  = ODEProblem(simplependulum, u₀, (0, 20), (g, L))
sol   = solve(prob)
plot(sol, N=0)
```



# Composing packages

Bayesian inference – Turing.jl

Differential equations – DifferentialEquations.jl

Bayesian inference in differential equations?

Turing.jl + DifferentialEquations.jl + Unitful.jl + ...

# Code is truly generic

```
-g/L * sin(θ) - 0.1dθ
```

`sin(θ)` works not matter what type  $\theta$  has

- `Float64` (double)
- Uncertain value
- Symbolic value
- Dual number (for automatic diff.)

`np.sin(θ)` does not accept anything other than floating point

# Code is truly generic – Neural ODEs

```
using DifferentialEquations, DiffEqFlux

function neural_pendulum(du,u,p,t)
    θ, dθ = u
    du[1] = dθ
    du[2] = neural_network(θ, dθ, p)
end
```

- Scientific Machine Learning in Julia requires no special solver
  - [diffeqflux.sciml.ai/](https://diffeqflux.sciml.ai/) (100s of solvers for SciML using ODE, DAE, PDE, DDE, SDE)
  - All solvers work with AD, GPU, units, uncertainty etc.
- Pure Julia solvers routinely benchmark faster than previous Fortran gold standard

# Scientific Modeling Cheatsheet

## MATLAB – Python – Julia Quick Reference

### Automatic Differentiation

#### ⚠ Python Compatibility Warning:

- SciPy functions (scipy.integrate, scipy.optimize, etc.) - must use PyTorch equivalents or pure Python
- NumPy operations - must convert arrays to tensors and use torch functions
- CasADi (used in Component-Based Modeling section) - cannot mix symbolic and AD frameworks

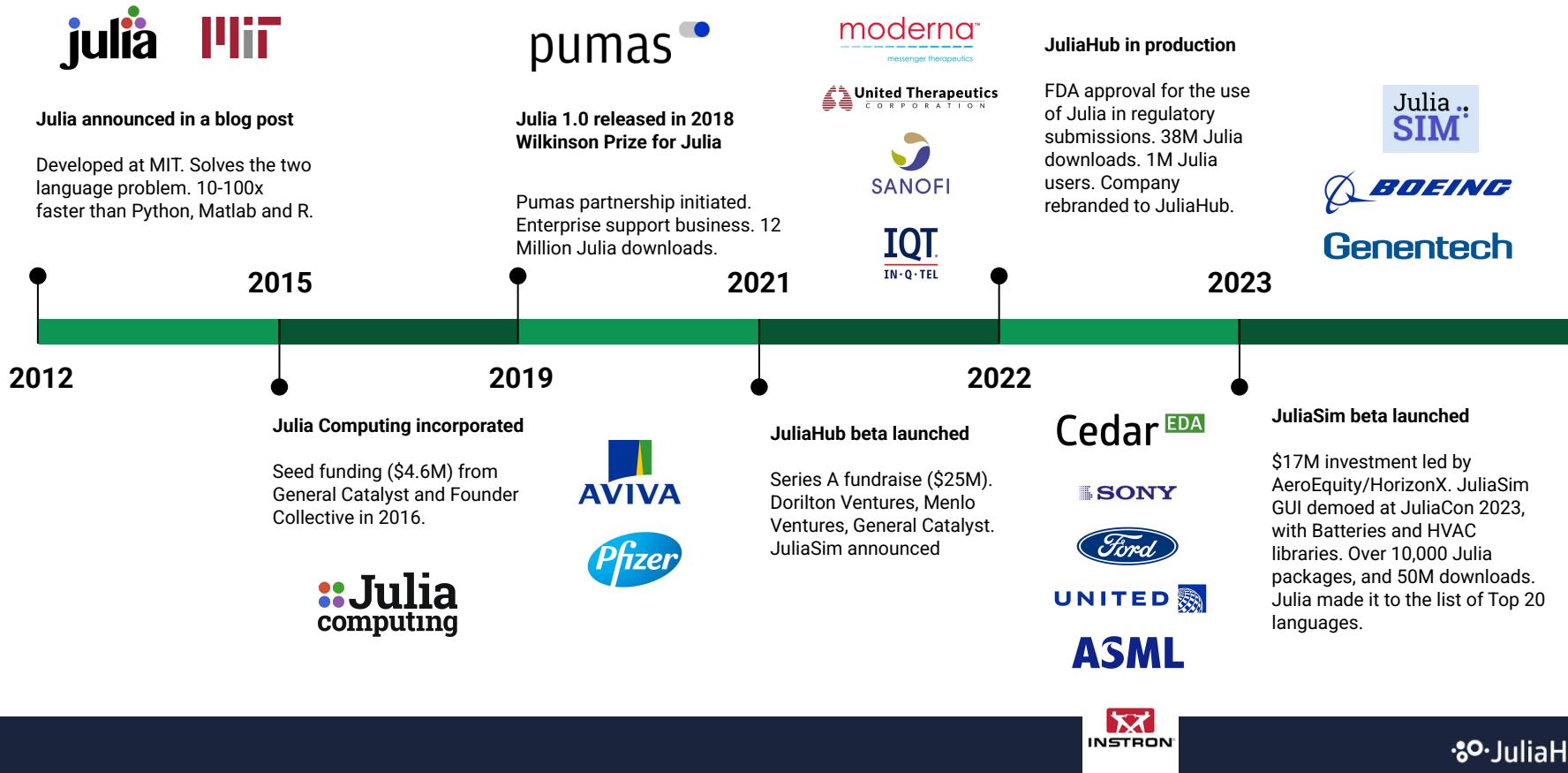
### Forward Mode AD

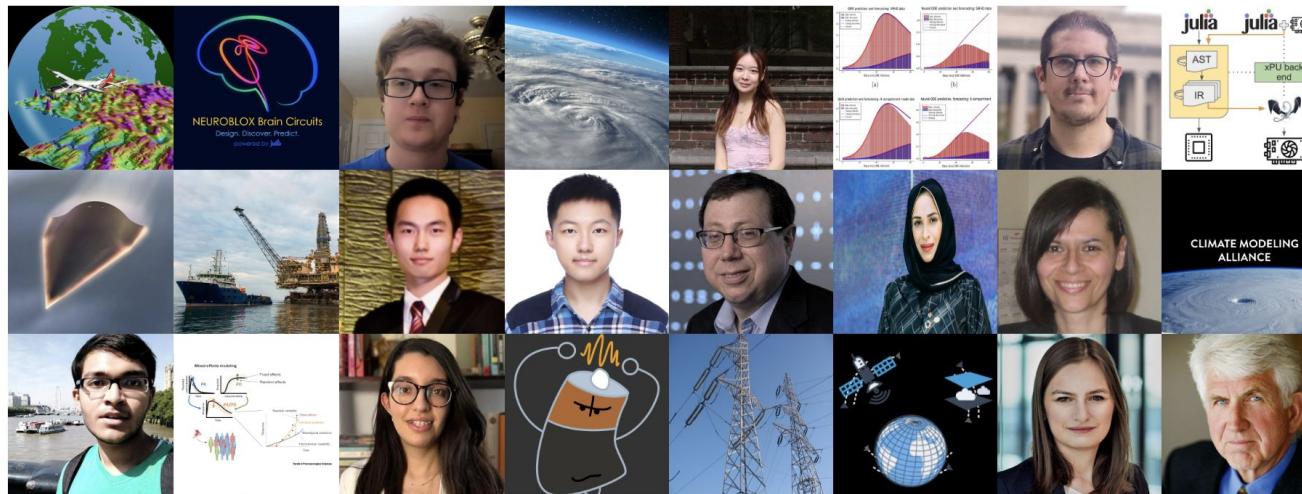
Description	MATLAB	Python	Julia
Setup	% Not available natively	import torch	using ForwardDiff
Gradient (scalar → scalar)	% Not available natively % Use symbolic or finite differences	from torch.autograd.functional import jvp  def f(x): return x**2 + torch.sin(x)  x = torch.tensor(2.0) v = torch.tensor(1.0) # tangent vector y, df = jvp(f, x, v) # df is derivative at x=2.0	f(x) = x^2 + sin(x) df = ForwardDiff.derivative(f, 2.0)

# **Non-Technical Story: The Growth Model**



# Who are we? JuliaHub Timeline





# The MIT Julia Lab's Effect on Open Source



Alan Edelman  
Principal Investigator

- Runs major courses every year that train new students in Julia (Linear Algebra)
- Writes many of online training materials



Chris Rackauckas  
Expert Collaborator

- Me!
- Creator and maintainer of ~200 packages in SciML and beyond



Avik Pal  
PhD Student

- Created and maintains the Lux.jl deep learning system
- Maintains many of the SciML and deep learning packages



Bowen Zhu  
PhD Student

- Becoming a maintainer of Symbolics.jl



Nicholas E Klugman  
PhD Student

- Creator of NeuralLyapunov.jl



Utkarsh  
PhD Student

- Major contributions to SciML packages



Raye Kimmerer  
Master's Student

- Chair for JuliaCon
- Maintainer for Julia sparse linear algebra libraries and LinearSolve.jl



Songchen Tan  
PhD Student

- Creator of TaylorDiff.jl
- Getting involved in SciML packages



Vaibhav Dixit  
Master's Student

- Creator and maintainer of Optimization.jl and related SciML packages



Julian Samoodoo  
Research Software Engineer

- Creator of Julia's AMD GPU support stack
- Maintainer of Dagger.jl



Torkel Loman  
Postdoctoral Associate

## Last Year's Alumni

- Creator of Catalyst.jl for systems biology
- Runs many community events and teaching workshops



Frank Schaefer  
Postdoctoral Associate

- Maintainer of many SciML packages



Shashi Gowda  
PhD Student

- Creator of Symbolics.jl



Valentin Churavy  
PhD Student

- Creator of Julia's HPC and GPU stack
- Creator of Enzyme automatic differentiation library



Gaurav Arya  
Undergraduate

- Creator of StochasticAD.jl

# Difficulties of Centering an Open Source Community on Academic Labs

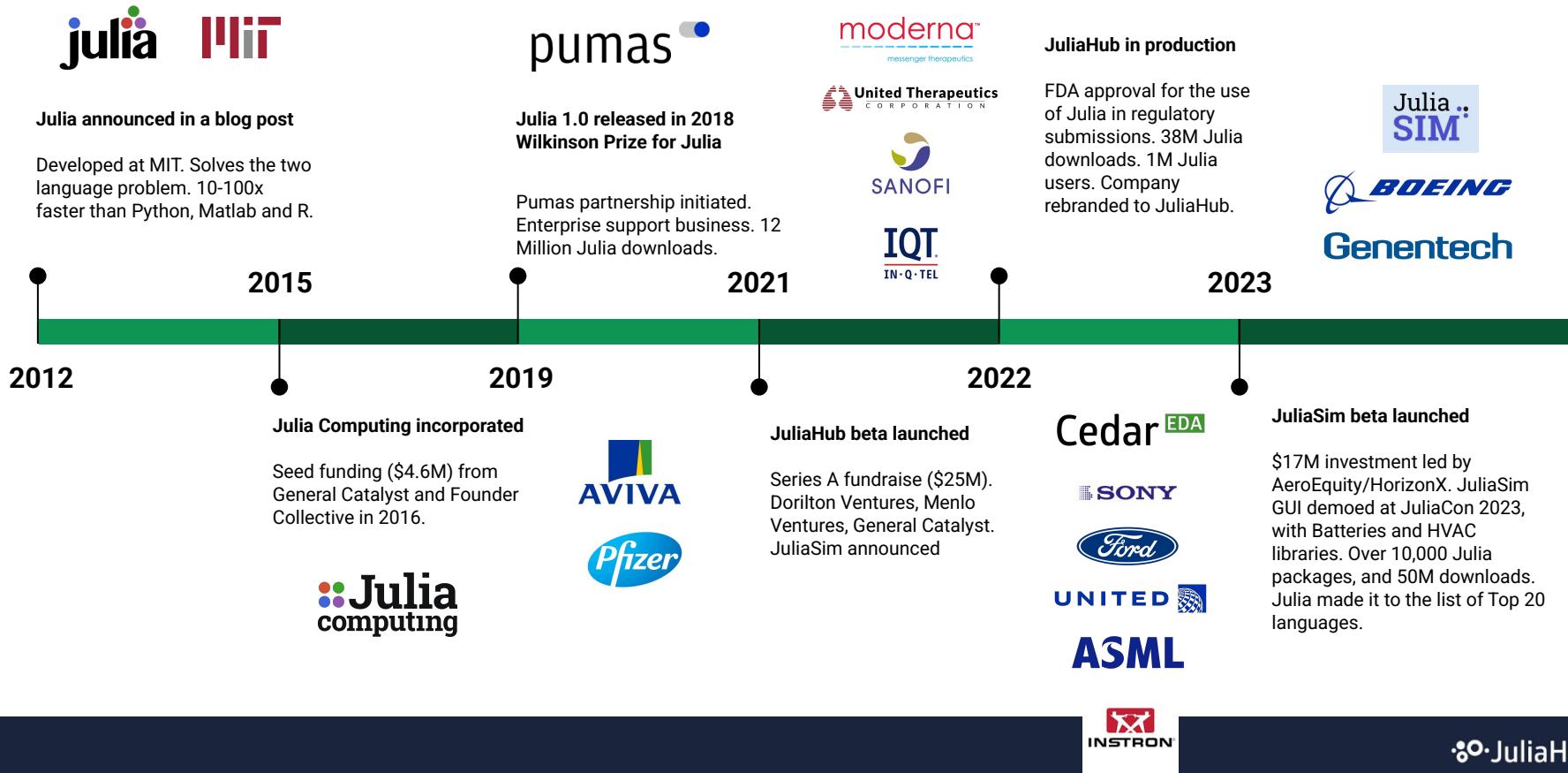
**Academic funding requires a lot of work  
and is naturally transient**

- Most positions have time limits (PhD student: 5 years, Postdoc: 3 years) or it can ruin their career
- All funding comes from writing 30 page grants and having a 17% chance of acceptance
  - ~\$0.5-3 million at a time
  - University overhead is >50%
  - Must fund tuition for students
- No funding is guaranteed to reoccur after a few years (boom and bust cycles)
- Funding agencies are looking for scientific merit, not software maintenance
- Students need to spend a significant amount of time in early years on course work, and later years time is spent on research papers and thesis writing
- Students are younger and earlier in their career (they are trainees!)

**What is PumasAI and JuliaHub and how has it helped build the Julia and SciML open source communities?**



# Who are we? JuliaHub Timeline



# Fundamental of Clinical Pharmacology: Nonlinear Mixed Effects (NLME) Modeling

Goal: Learn to predict patient behavior (**dynamics**) from simple data (**covariates**)

$$Z_i = \begin{bmatrix} wt_i, \\ sex_i, \end{bmatrix}$$

Covariates

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} = \begin{bmatrix} \theta_1 e^{\eta_{i,1} \kappa_{i,k,1}}, \\ \theta_2 \left(\frac{wt_i}{70}\right)^{0.75} \theta_4^{sex_i} e^{\eta_{i,2}}, \\ \theta_3 e^{\eta_{i,3}}, \end{bmatrix}$$

Structural Model (pre)

Intuition:  $\eta$  (the random effects) are a fudge factor

Find  $\theta$  (the fixed effect, or average effect) such that you can predict new patient dynamics as good as possible

Math: Find  $(\theta, \eta)$  such that  $E[\eta] = 0$

Requires special fitting procedures (Pumas)

$$\frac{d[\text{Depot}]}{dt} = -Ka[\text{Depot}],$$

$$\frac{d[\text{Central}]}{dt} = Ka[\text{Depot}] - \frac{CL}{V}[\text{Central}].$$

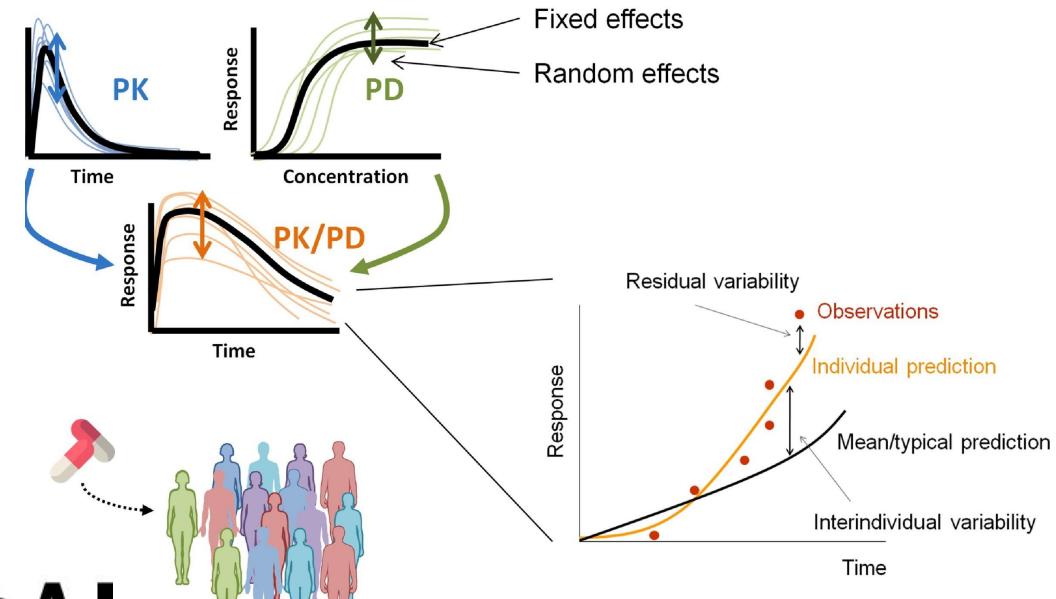
Dynamics

# Why Pumas?

## A Front-End to SciML and the Julia Ecosystem for Pharmacometrics

### SciML-enhanced modeling for clinical trials

- Readers for the common dataset types
- Specialized tooling for NLME models
- Report generation processes for FDA-compliant results
- Clinical consulting team to help
- Specialized domain-specific packages for further analysis
- ... many many features



 **PumasAI**

Trends in Pharmacological Sciences

# The Impact of Pumas (PharmacUtical Modeling And Simulation)

“

**We have been using Pumas software for our pharmacometric needs to support our development decisions and regulatory submissions.**

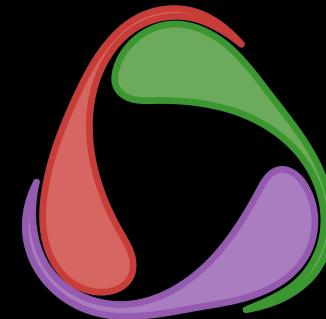
Pumas software has surpassed our expectations on its accuracy and ease of use. We are encouraged by its capability of supporting different types of pharmacometric analyses within one software. **Pumas has emerged as our "go-to" tool for most of our analyses in recent months.** We also work with Pumas-AI on drug development consulting. We are impressed by the quality and breadth of the experience of Pumas-AI scientists in collaborating with us on modeling and simulation projects across our pipeline spanning investigational therapeutics and vaccines at various stages of clinical development

**Husain A. PhD (2020)**

Director, Head of Clinical Pharmacology and Pharmacometrics,  
*Moderna Therapeutics, Inc*

”

**Built on SciML**



# Why was it better for open source for Pumas to be closed source?

## Highly specialized packages do not get industrial contributors

- Industrial users are heavy users but not heavy contributors
- The industrial system has solid methods for paying for resources, but does not have solid ways to make consistent and reliable donations
- Regulatory processes need an agency to ensure it can be used in regulated practices
- It gives a funding mechanism to consistently fund some of the major contributors to the open source ecosystem with full time jobs

Benefits to



PumasAI

- Hires are already well-versed in the tools and have demonstrated prior ability to work in large teams to build usable software
- Hires have knowledge of the full software stack and all of its dependencies
- “Free work” is usable: by being in the open source system, they can guide and review the PRs to ensure that new features that are being developed by others are also useful to Pumas.

# What is the concrete effect of PumasAI on Open Source?



David Widmann, PhD  
Product Engineer

- SciML steering council member
- DiffEq.jl #4 all time?
- Bayesian statistics (Turing.jl) top maintainer



Niklas Korsbo, PhD  
Senior Product Engineer

- Latexify.jl
- Contributions to Catalyst.jl
- Many many solid bug reports and MWEs
- Many training materials



Mohamed Tarek, PhD  
Senior Product Engineer

- TopOpt.jl
- AbstractDifferentiation.jl
- Nonconvex.jl
- Many many posts in the Julia discourse helping people in the open source community



Julius Krumbiegel  
Product Engineer

- Makie.jl top 3 contributor
- Chain.jl
- DataFrameMacros.jl



Dilum Aluthge  
Clinician Scientist

- One of the maintainers of the Julia package ecosystem repository
- One of the big maintainers in the full Julia CI/CD system



Lucas Pereira  
Product Engineer

- TopOpt.jl



Chris Rackauckas, PhD  
Director, Scientific Research

- Me!



Andreas Noack, PhD  
Head of Product Engineering



Patrick Kofod Mogensen  
pkofod

- Developed and maintains Julia's Linear Algebra system
- Big contributor to statistics libraries
- Maintains Optim.jl, currently the most used nonlinear optimization package in Julia
- Maintains JuliaNLSolvers Organization, ~5 other packages



Michael Hatherly  
MichaelHatherly : he/him

- Creator of the Documenter.jl documentation system for Julia
- Created many of the Julia packages for markdown support

[← Back to News Feed](#)

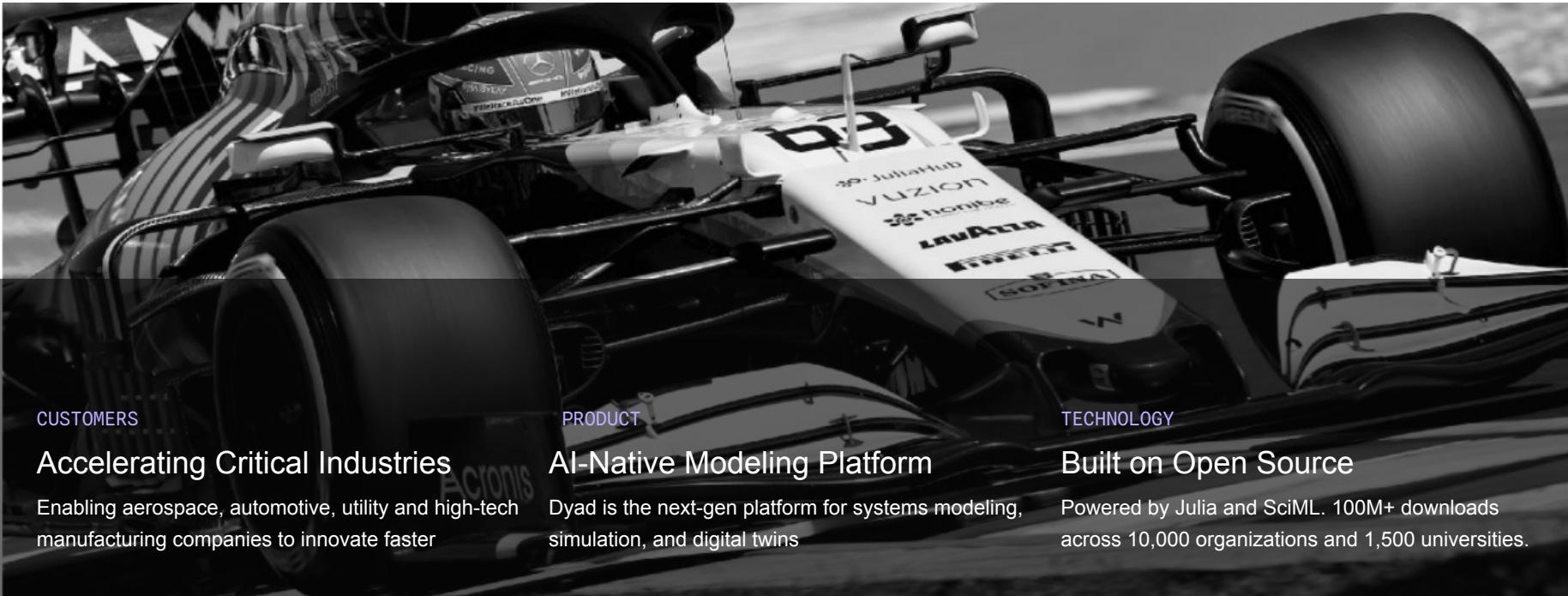
JUNE 27, 2023

HORIZONX

# JuliaHub Receives \$13 Million Strategic Investment from AE Industrial Partners HorizonX

CAMBRIDGE, Mass. and BOCA RATON, Fla., June 27, 2023 — JuliaHub has announced \$13 million in a strategic new investment led by AE Industrial Partners HorizonX (“AEI HorizonX”). AEI HorizonX is AE Industrial Partners’ venture capital investment platform formed in partnership with The Boeing Company.

# Dyad accelerates breakthrough engineering with Software-Defined Machines



## CUSTOMERS

### Accelerating Critical Industries

Enabling aerospace, automotive, utility and high-tech manufacturing companies to innovate faster

## PRODUCT

### AI-Native Modeling Platform

Dyad is the next-gen platform for systems modeling, simulation, and digital twins

## TECHNOLOGY

### Built on Open Source

Powered by Julia and SciML. 100M+ downloads across 10,000 organizations and 1,500 universities.

# Scientific Machine Learning Digital Twins: More Realistic Results than Pure ML

WILLIAMS  
RACING



**Result:**

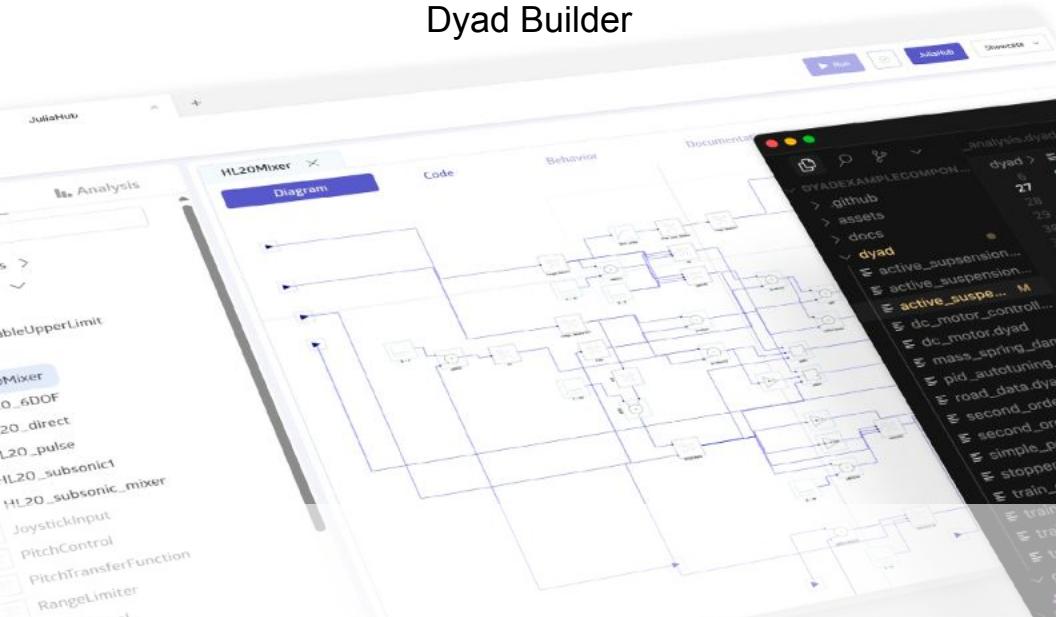
**The Julia community and JuliaHub making major inroads into industrial use cases**



# Dyad is the first AI-native physics modeling platform designed to accelerate hardware design

PHYSICAL ENGINEERS

Dyad Builder



Text to model

SOFTWARE DEVELOPERS

Dyad Studio



Agile development of hardware with Git and CI/CD



Dr. Viral B. Shah  
Co-founder and CEO

- Co-Founder of Julia
- Continues to help with linear algebra and binaries



Alex Jones  
xtsotx

- Maintains MethodOfLines.jl and other SciML PDE tools



Dr. Jeff Bezanson  
Co-founder and CTO

- Co-Founder of Julia
- Lead developer of the Julia compiler



Aayush Sabharwal  
AayushSabharwal - Rebirth

- Maintainer of ModelingToolkit.jl and other base SciML libraries



Dr. Matt Bauman  
Director of Customer Success

- Many contributions to Julia's Base library (arrays)
- Moderator for Julia community forums



Cody Tapscott  
Nimbley

- Major contributions to the Julia compiler



Avik Sengupta  
VP of Engineering

- Moderator for Julia community forums



Anant Thazhemandam  
Thazhemandam

- Keeps our CI/CD and devops alive throughout the Julia community



Keno Fischer  
Co-founder and CTO  
(R&D)

- Major contributor to the Julia compiler



Dhairya Gandhi  
Dhary Gandhi

- Major contributor to Flux.jl deep learning library



Stefan Karpinski  
Co-founder and CPO

- Co-Founder of Julia
- Continues to develop the Julia package manager



Gabriel Baraldi  
stbiralid

- Major contributions to the Julia compiler



Dr. Chris Rackauckas  
VP of Modeling and Simulation

- Me!



Fredrik Ekre  
FredrikEkre

- Creator and maintainer of the Ferrite FEM library



Sebastian Pfützner  
seppf

- Creator and maintainer Julia VS Code plugin



Jameson Nash  
jgnash

- Major contributions to the Julia compiler



Josh Day

- Created and maintains OnlineStats.jl
- Major contributions to JuliaStats libraries



Kristoffer Carlsson

- Maintains the Julia release process
- Creator and maintainer of the Ferrite FEM library
- Major contributions to the package manager



Oscar Smith  
oscarosmith

- Major contributions to the Julia core math functions
- Maintainer of many SciML packages



Morten Pilbøe  
mortenpli

- Maintainer of Documenter.jl and Julia's Documentation



Παναγιώτης Τσιριπακούδης  
panagiotis\_tsiripakoudis

- Maintainer of the Pluto.jl notebook system



Sebastian Mielut-  
Câmpeneu

- Major contributions to SciML and JuliaDynamics



Tim Besard

- Creator and maintainer of the Julia GPU stack and CUDA.jl

# Conclusion

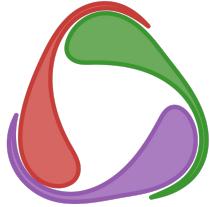
---

Julia might not have the most users right now, but as a development community it is going strong

- Being high level = not requiring C/Fortran which means there is a higher percentage of users that turn into package developers (I'd estimate this is at least an order of magnitude higher!)
- The modular design greatly increases the developer base
  - Python has what seems to be 10 package managers, 10 JIT compilers, 10 GPU interfaces, 10 copies of scientific computing and array libraries, and each combination seems incompatible...
- Designing research laboratories and companies into the open source community has greatly increased the developer base

Tl;dr, when looking at package development statistics, you see far more contributors to the Julia SciML ecosystem than even many of the major Python or R ecosystems, and this is by design.

Let's see how this developer focus turns out a decade from now.



## Connect with SciML



**Join our community chatrooms**  
[julialang.org/slack](https://julialang.org/slack)



**Follow on Twitter**  
[@ChrisRackauckas](https://twitter.com/ChrisRackauckas)



**Follow on LinkedIn**  
<https://www.linkedin.com/in/chrisrackauckas/>



**Star the GitHub repositories**  
<https://github.com/SciML>



**Dr. Chris Rackauckas**

**VP of Modeling and Simulation**  
JuliaHub

**Research Affiliate**  
MIT CSAIL

**Director of Scientific Research**  
Pumas-AI