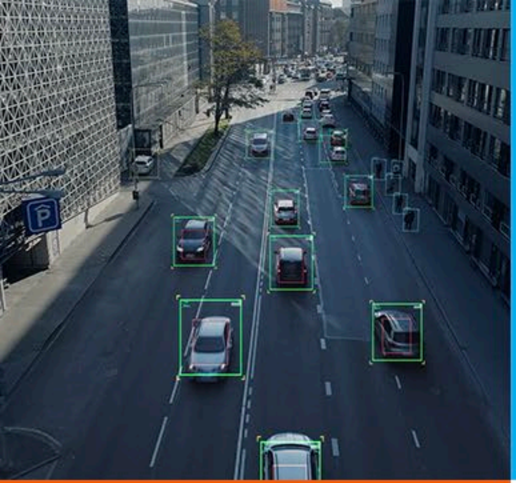


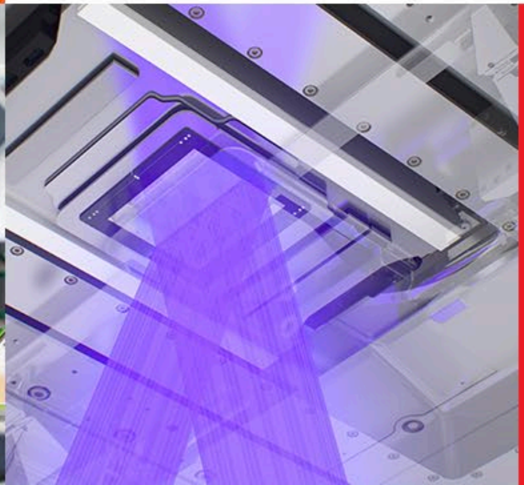
MECHATRONICS

MATHWARE

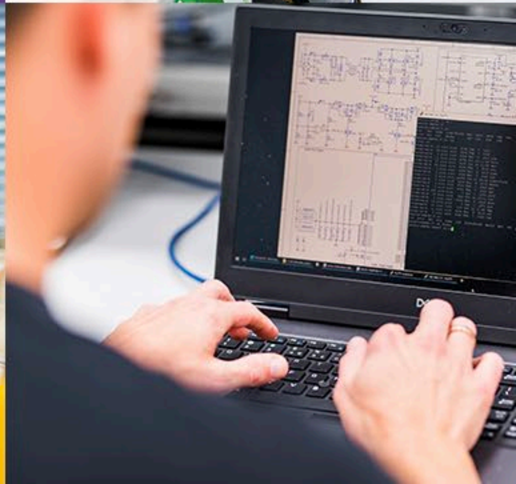
ASSEMBLY

SIOUX TECHNOLOGIES

ELECTRONICS



SOFTWARE



julia

Why Julia?

14 February 2012 | **Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman**

In short, because we are greedy.

We are power Matlab users. Some of us are Lisp hackers. Some are Pythonistas, others Rubyists, still others Perl hackers. There are those of us who used Mathematica before we could grow facial hair. There are those who still can't grow facial hair. We've generated more R plots than any sane person should. C is our desert island programming language.

We love all of these languages; they are wonderful and powerful. For the work we do — scientific computing, machine learning, data mining, large-scale linear algebra, distributed and parallel computing — each one is perfect for some aspects of the work and terrible for others. Each one is a trade-off.

We are greedy: we want more.

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?)

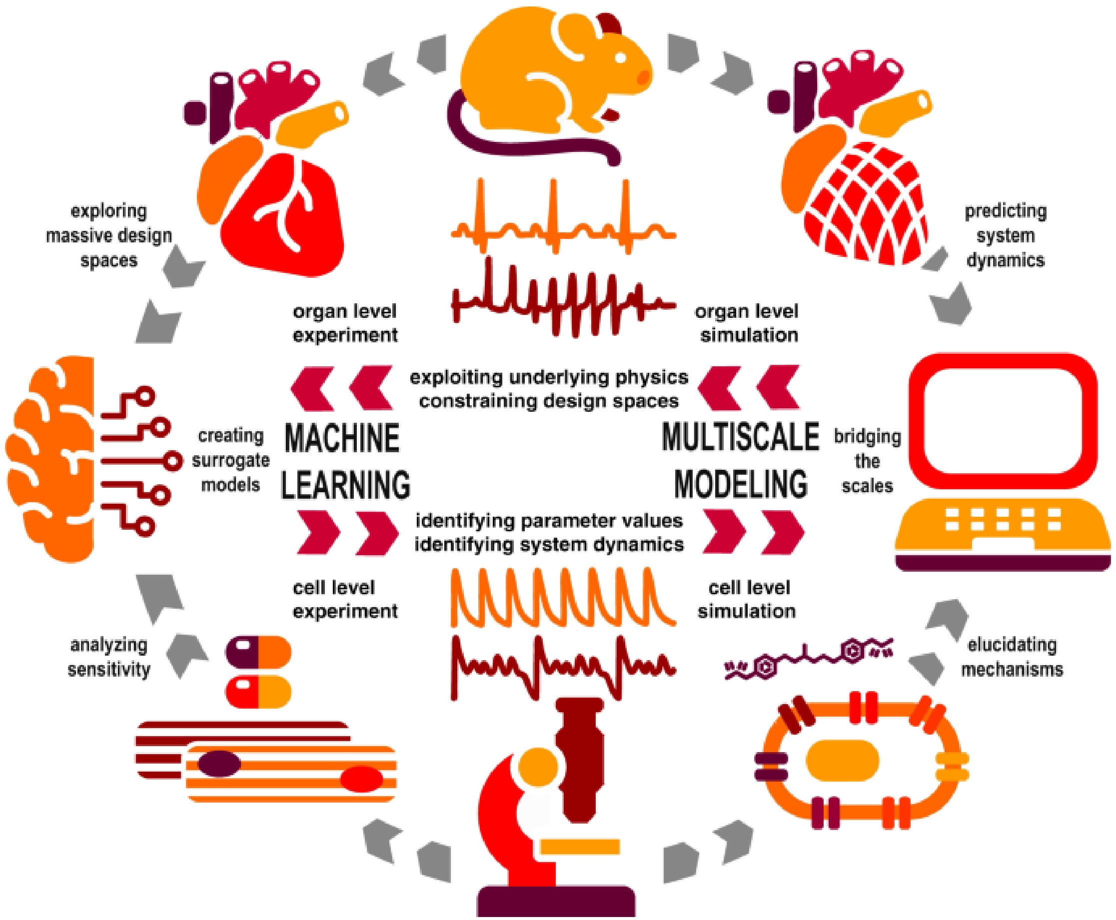
While we're being demanding, we want something that provides the distributed power of Hadoop — without the kilobytes of boilerplate Java and XML; without being forced to sift through gigabytes of log files on hundreds of machines to find our bugs. We want the power without the layers of impenetrable complexity. We want to write simple scalar loops that compile down to tight machine code using just the registers on a single CPU. We want to write A*B and launch a thousand computations on a thousand machines, calculating a vast matrix product together.

We never want to mention types when we don't feel like it. But when we need polymorphic functions, we want to use generic programming to write an algorithm just once and apply it to an infinite lattice of types; we want to use multiple dispatch to efficiently pick the best method for all of a function's arguments, from dozens of method definitions, providing common functionality across drastically different types. Despite all this power, we want the language to be simple and clean.

All this doesn't seem like too much to ask for, does it?

Even though we recognize that we are inexcusably greedy, we still want to have it all. About two and a half years ago, we set out to create the language of our greed. It's not complete, but it's time for an initial[1] release — the language we've created is called Julia. It already delivers on 90% of our ungracious demands, and now it needs the ungracious demands of others to shape it further.

So, if you are also a greedy, unreasonable, demanding programmer, we want you to give it a try.



The Unreasonable Effectiveness of Mathematics

Algorithms everywhere

My grudge with modern C++

Simplify C++!

It’s verbose and does not look like math!

```
template<class... Fs>
struct covariant : overload<Fs...>{
    covariant(Fs... fs) : overload<Fs...>(fs...){}
    template<class... Ts, typename = decltype(overload<Fs.
    decltype(auto) call(Ts&&... ts) const{
        if constexpr(std::is_same<decltype(overload<Fs...>
            return overload<Fs...>::operator()(std::forwar
        else
            return overload<Fs...>::operator()(std::forwar
    }
    template<
        class... Variants,
        class Ret = detail::variant_of_set_t<
            detail::results_of_setn_t<
                overload<Fs...> const&,
                detail::variant_types_list_t<Variants>...
            >
        >
    >
    Ret operator()(Variants const&... vs){
        return pivot([&](auto&&... es)->Ret{return call(es
    }
};
```

Multiple dispatch in C++

My grudge with Python

75x higher energy consumption!

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	[illegible]	(i) Lua	82.91	(i) Jruby	19.84

My grudge with Python

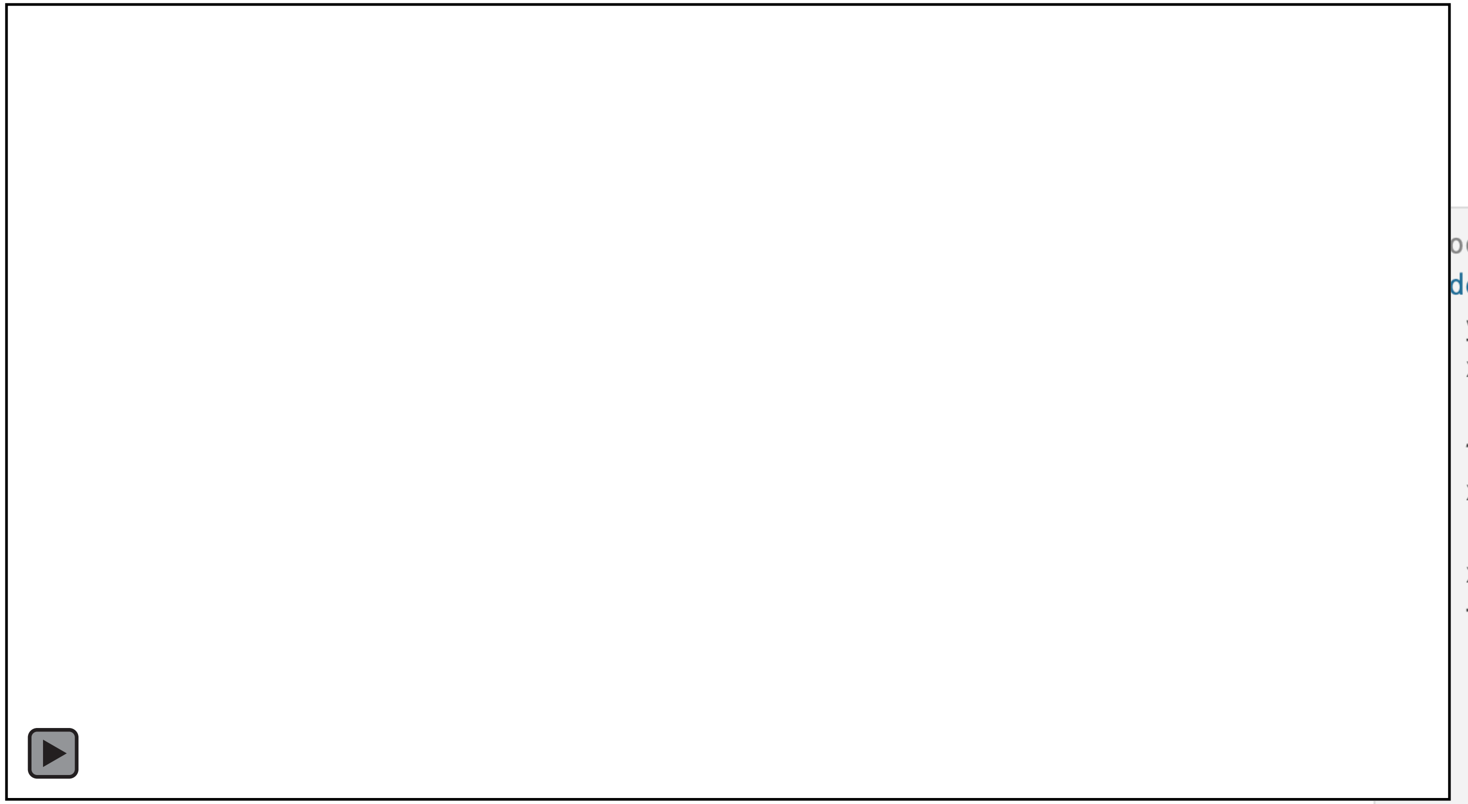
75x higher energy consumption!

Oh…**and**
It’s verbose and does not look like math!

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	[illegible]	(i) Lua	82.91	(i) Jruby	19.84

Julia = performance software looking like math



https://biaslab.github.io/RxInfer.jl/

```
odel of the complete system dynamics
del function noisy_pendulum(n)
 y = datavar(Float64, n)
 x = randomvar(n)

 τ ~ Gamma(α = 1.0, β = 1.0)
 x_0 ~ MvNormal(μ = zeros(2), Σ = diageye(2))

 x_prev = x_0
 for i in 1:n

     x[i] ~ g(x_prev)
     y[i] ~ Normal(μ = dot([1.0, 0.0], x[i]), γ=τ)
     x_prev = x[i]
 end

end
```

Program

1 This intro

2 Deepak Vincchi on Juliahub

3 Chris Rackauckas on SciML

4 matthijs Cox and Keith Myerscough on Julia in the Eindhoven practice

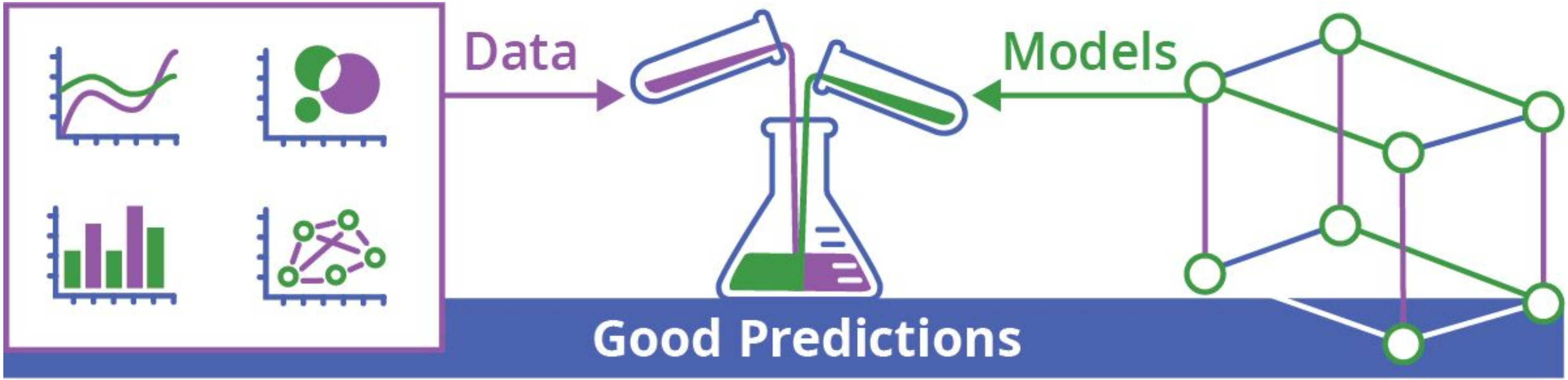
5 Question round

Why Julia in High-Tech Industry?
Scientific Machine Learning

April, 2023

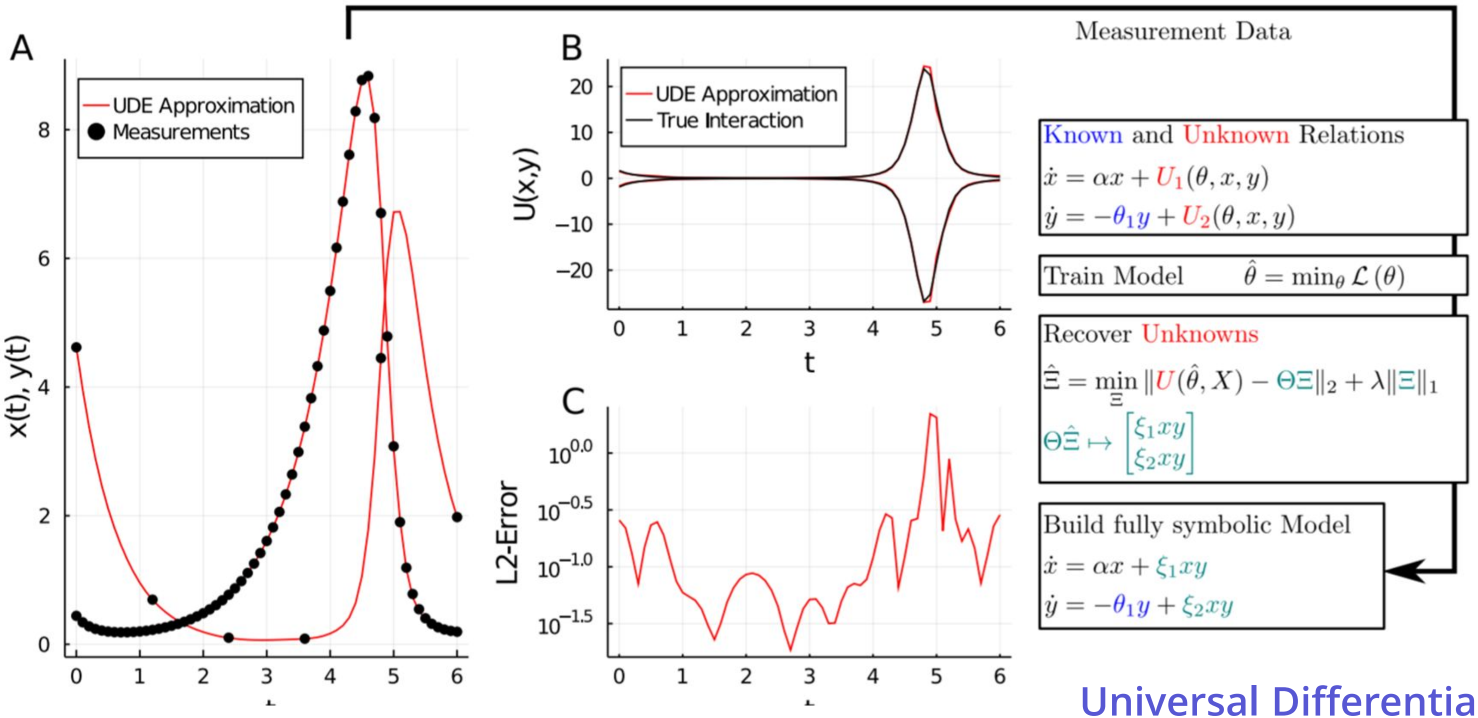
Scientific Machine Learning

Mixing Data and Models



Julia SIM

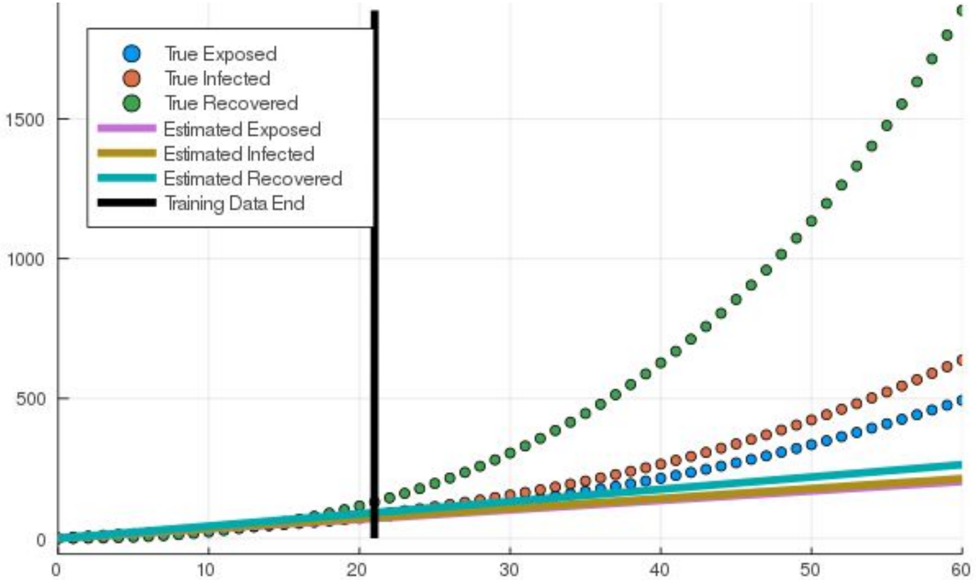
Auto-Completing Models with Machine Learning



Universal Differential Equations

Let’s dive in a bit!

Standard Machine Learning: Learn the whole model



u’=NN(u) trained on 21 days of data

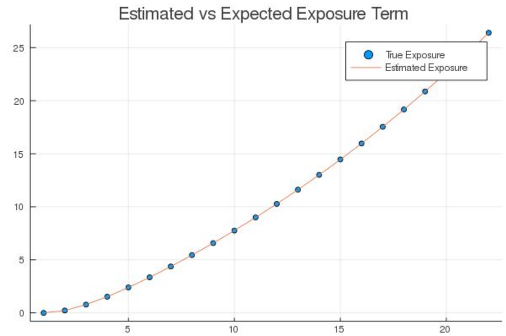
Can fit, but not enough information to accurately extrapolate

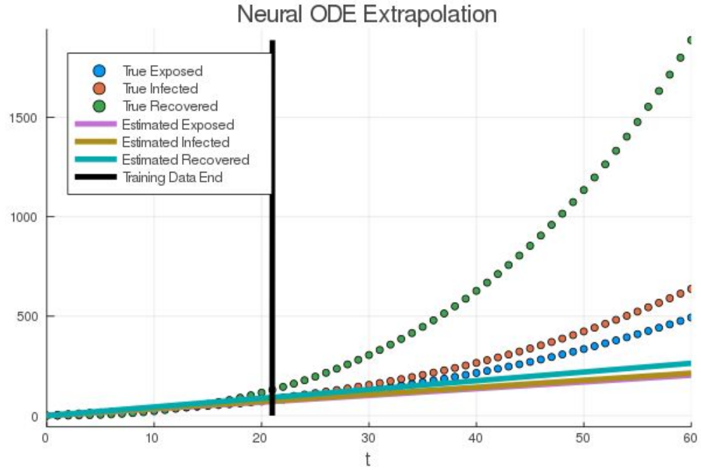
Does not have the correct asymptotic behavior

More examples of this issue:

Ridderbusch et al. "Learning ODE Models with Qualitative Structure Using Gaussian Processes."

Universal ODE





$$
\begin{aligned}
S' &= -\frac{\beta_0 S F}{N} - \quad - \mu S, \\
E' &= \frac{\beta_0 S F}{N} + \quad - \Big(\sigma + \mu\Big) E, \\
I' &= \sigma E - (\gamma + \mu) I, \\
R' &= \gamma I - \mu R, \\
N' &= -\mu N, \\
D' &= d\, \gamma I - \lambda D, \quad \text{and} \\
C' &= \sigma E,
\end{aligned}
$$

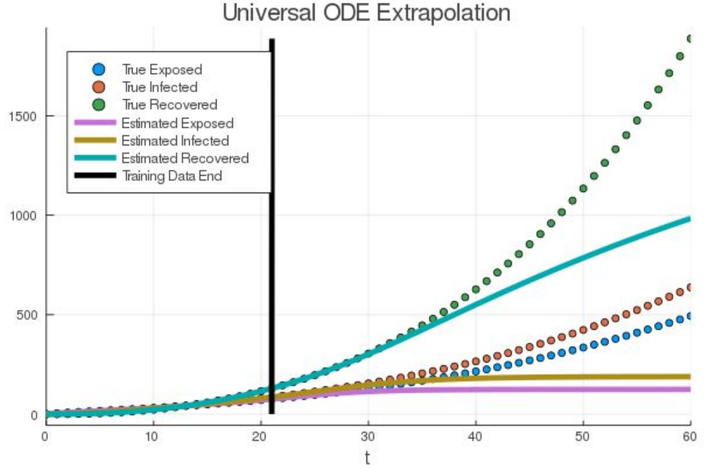
Replace Unknown Portion

Replace Unknown Portion

Exposure:
Unknown

Infection rates: known
From disease quantities

Percentage of cases known to be severe, can be estimated

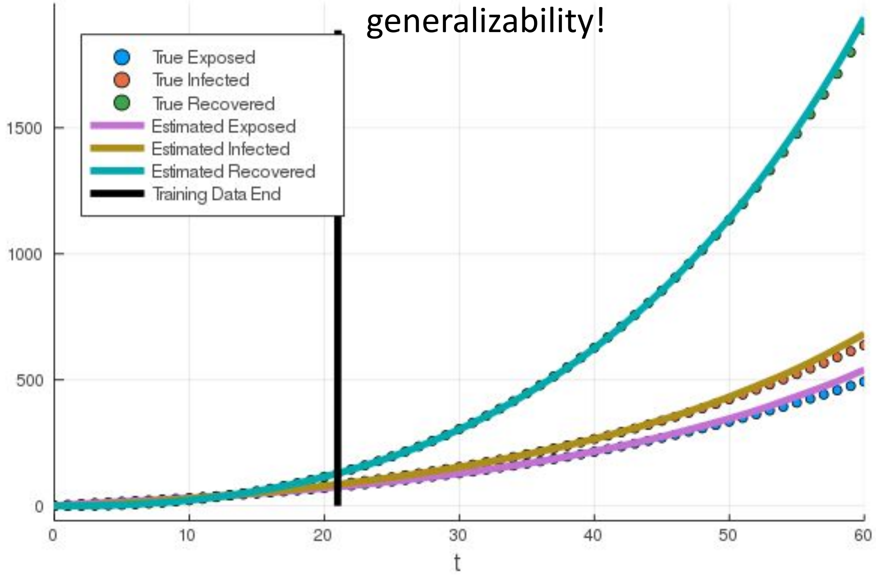


Universal ODE -> Internal Sparse Regression

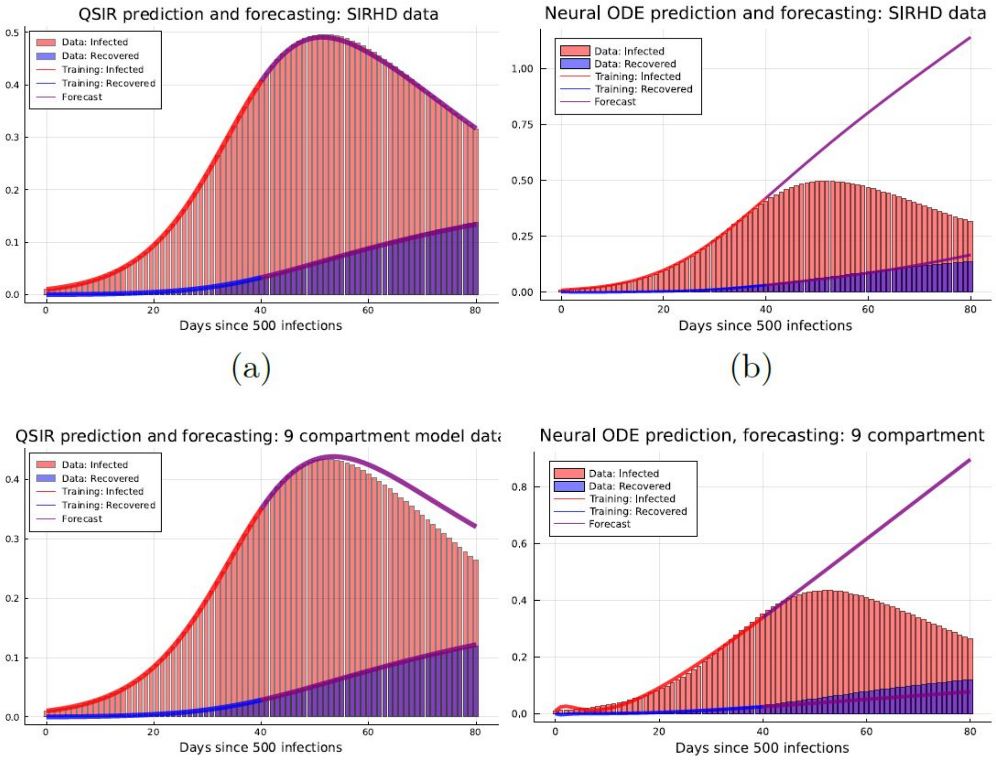
Sparse Identification on only the missing term:
I * 0.10234428543435758 + S/N * I * 0.11371750552005416 + (S/N) ^ 2 * I * 0.12635459799855597

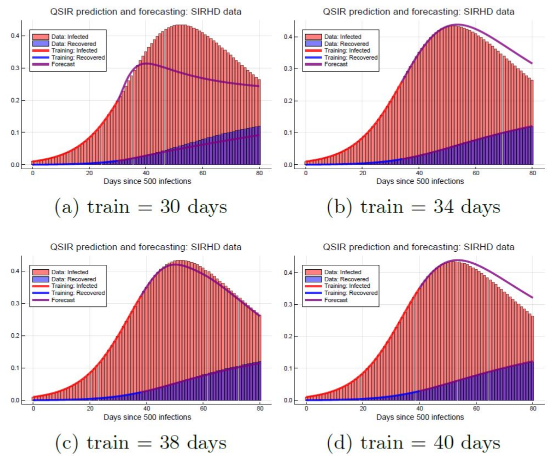
$$
\begin{aligned}
S' &= -\frac{\beta_0 S F}{N} - \text{Replace Unknown Portion} - \mu S,\\
E' &= \frac{\beta_0 S F}{N} + \text{Replace Unknown Portion} - (\sigma + \mu) E,\\
I' &= \sigma E - (\gamma + \mu) I,\\
R' &= \gamma I - \mu R,\\
N' &= -\mu N,\\
D' &= d\,\gamma I - \lambda D, \quad \text{and}\\
C' &= \sigma E,
\end{aligned}
$$

Sparsity improves generalizability!



Scientific Machine Learning: Improving Predictions with Less Data





Dandekar, R., Rackauckas, C., & Barbastathis, G. (2020). A machine learning-aided global diagnostic and comparative tool to assess effect of quarantine control in COVID-19 spread. Patterns, 1(9), 100145.

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:

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

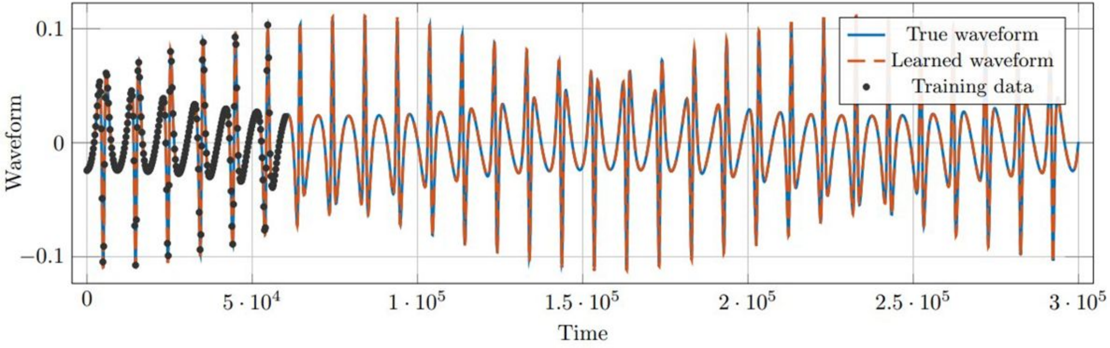
$$\dot{\chi} = \frac{(1 + e\cos(\chi))^2}{M p^{3/2}}(1 + \mathcal{F}_2(\cos(\chi), p, e)), \tag{5b}$$

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

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

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

https://github.com/Astroinformatics/ScientificMachineLearning/blob/main/neuralode_gw.ipynb



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.

Universal Differential Equations Build Earthquake-Safe Buildings

"Scientific Machine Learning for Earthquake-Safe Buildings"

Structural identification with physics-informed neural ordinary differential equations.

Lai, Zhili, Mylonas, Charilaos, Nagarajaiah, Staish, Chatzi, Eleni

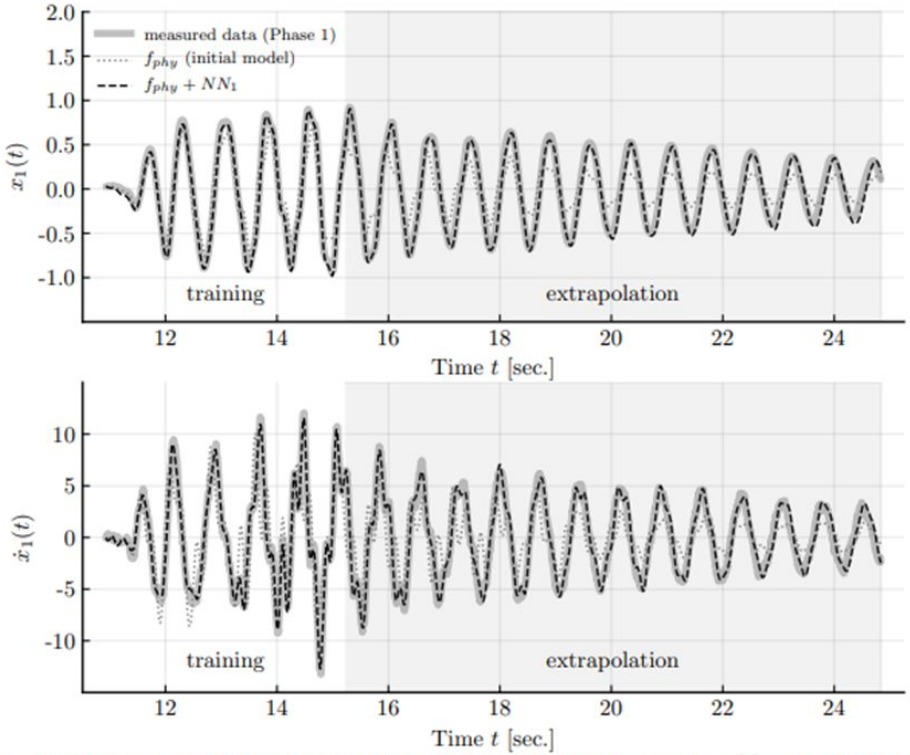


Figure 12: Comparison of time history of the response for displacement $x_1(t)$ and velocity $\dot{x}_1(t)$ for the NSD experiment (Phase 1).

Universal Differential Equations Predict Chemical Processes

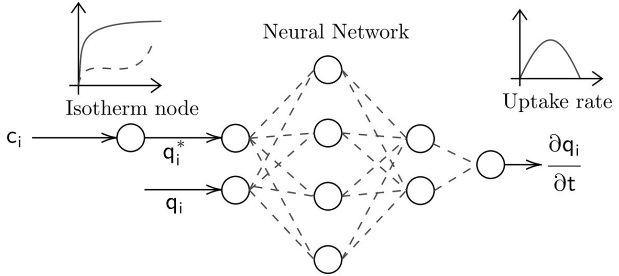
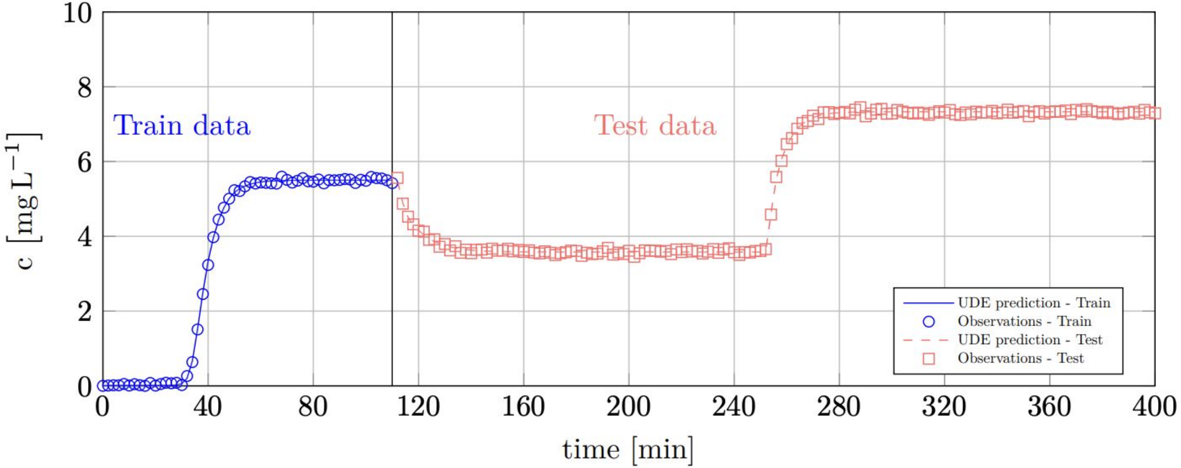


Figure 2: Schematic representation of the proposed hybrid model.



(a)

Langmuir isotherm - LDF

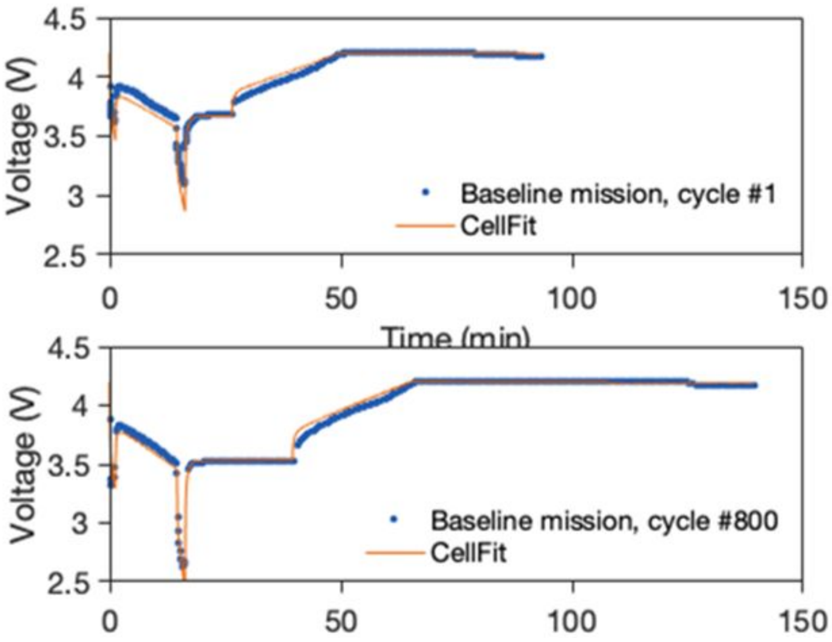
Santana, V. V., Costa, E., Rebello, C. M., Ribeiro, A. M., Rackauckas, C., & Nogueira, I. B. (2023). Efficient hybrid modeling and sorption model discovery for non-linear advection-diffusion-sorption systems: A systematic scientific machine learning approach. *arXiv preprint arXiv:2303.13555*.

Universal Differential Equations Generate More Accurate Models of Battery Degradation

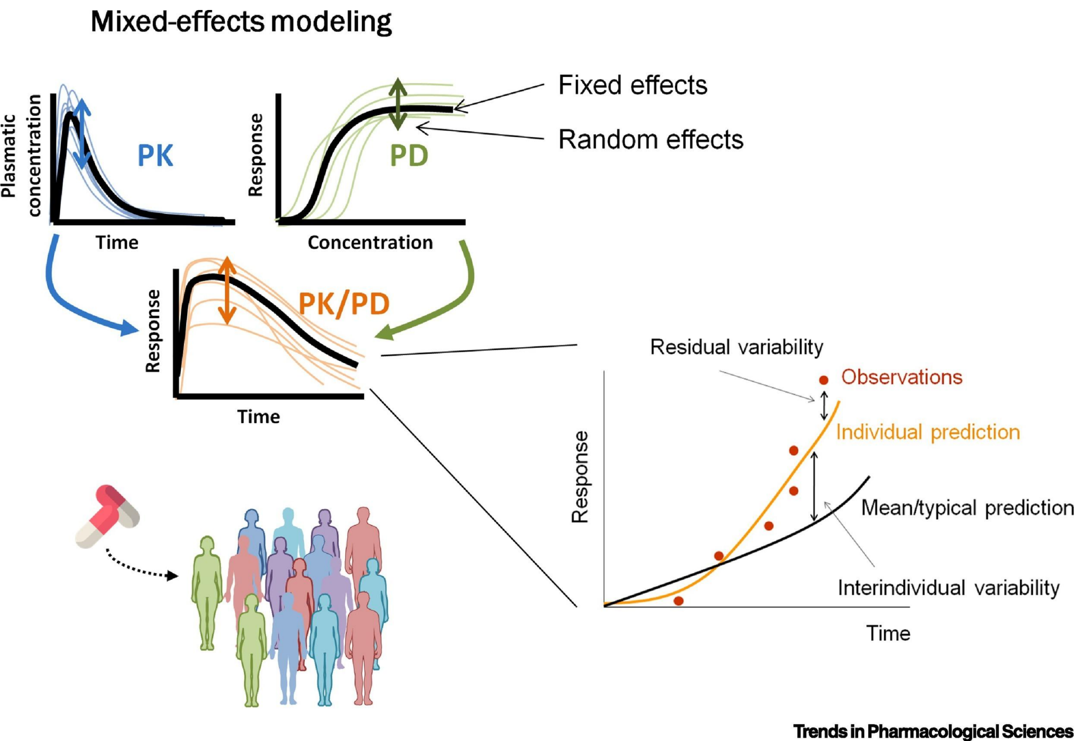
Researchers at CMU Used Universal Differential Equations to Improve Models of Battery Degradation to Suggest Better Batter Materials

“Universal Battery Performance and Degradation Model for Electric Aircraft”

Nills, Sripad, Fredericks, Gutenberg, Charles, Frank, Viswanathan



DeepNLME: Integrate neural networks into traditional NLME modeling
DeepNLME is SciML-enhanced modeling for clinical trials



pumas

- Automate the discovery of predictive covariates and their relationship to dynamics
- Automatically discover dynamical models and assess the fit
- Incorporate big data sources, such as genomics and images, as predictive covariates

From Dynamics to Nonlinear Mixed Effects (NLME) Modeling

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

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

**Covariate
s**

Requires special fitting procedures (Pumas)

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

**Structural Model
(pre)**

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

**Dynamic
s**

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

moderna™

messenger therapeutics

Built on SciML

From Dynamics to 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

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

Structural Model (pre)

How can we find these models?

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

Dynamics

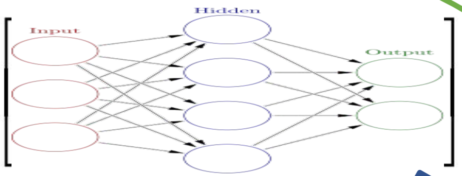
From Dynamics to Nonlinear Mixed Effects (NLME) Modeling

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

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

Covariates

$$g_i = \begin{bmatrix} Ka \\ CL \\ V \end{bmatrix} =$$

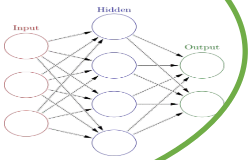


Structural Model (pre)

How can we find these models?

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

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



Dynamics

Idea: Parameterize the model such that the models can be neural networks, where the weights of the neural networks are fixed effects!

Indirect learning of unknown functions!

DeepNLME in Practice: Data Mining for Predictive Covariates

```
model = @model begin
  @param begin
    θ ∈ VectorDomain(lower=[0.1,0.0008,0.0040.1],upper=[5.0,0.5,0.9,5.0])
    Ω ∈ PSDDomain(3)
    σ²_add ∈ RealDomain(lower=0.001, init=sqrt(0.388))
    p1 ∈ NeuralDomain(FastChain(FastDense(2,50,tanh),FastDense(50,1),(x,p)->x.^2))
    p2 ∈ NeuralDomain(FastChain(FastDense(2,50,tanh),FastDense(50,1),(x,p)->x.^2))
  end

  @random begin η ~ MvNormal(Ω) end

  @pre begin
    Ka = SEX == 0 ? θ[1] + η[1] : θ[4] + η[1]
    K = nn1([θ[2],η[2]],p1)[1]
    CL = nn2([θ[3]*WT,η[3]],p2)[1]
    Vc = CL/K
    SC = CL/K/WT
  end

  @covariates SEX WT
  @vars begin conc = Central / SC end
  @dynamics Depots1Central1
  @derived begin dv ~ @. Normal(conc, sqrt(σ²_add)) end
end
```

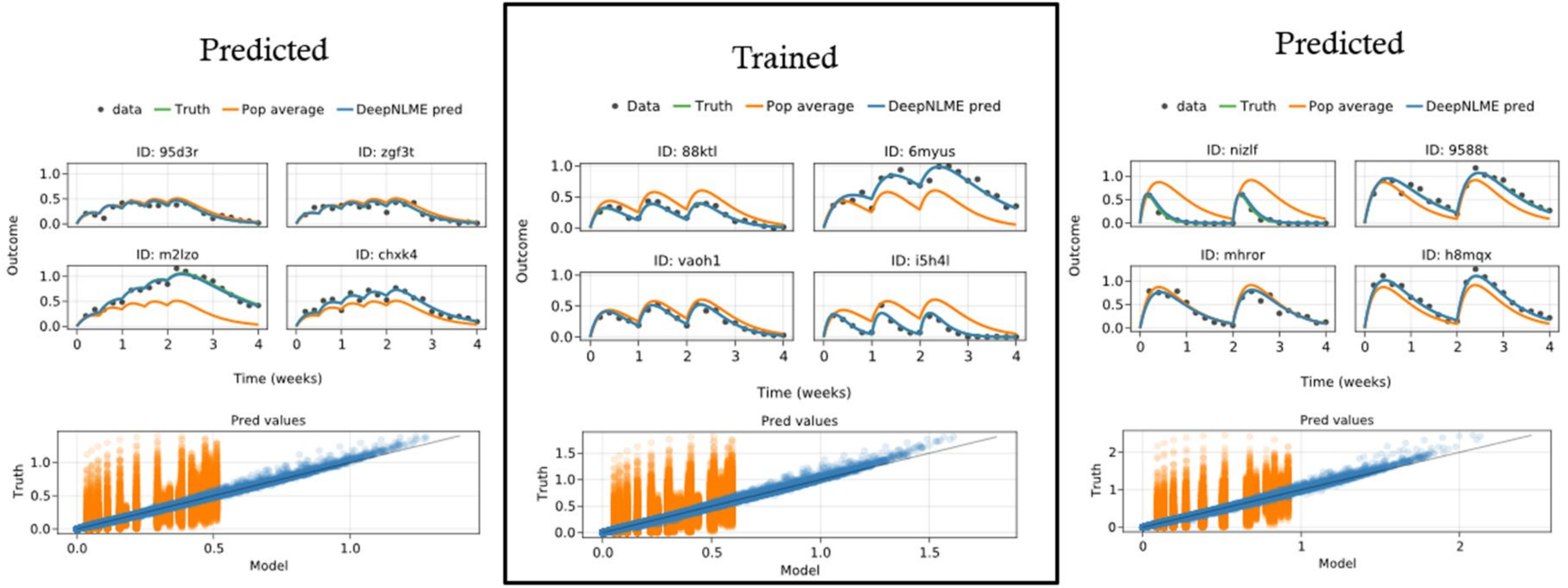
Automate the discovery of covariate models

- Train convolutional neural networks to incorporate images as covariates
- Train transformer models to utilize natural language processing on electronic health records
- Utilize automated model discovery to prune genomics data to find the predictive subset

Utilize GPU acceleration for neural networks

Currently being tested on clinical trial data

DeepNLME: Automated Construction of Patient-Specific Pharmacological Models for Individualized Dosing

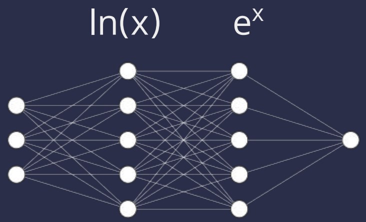


Award by International Society of Pharmacometrics
Currently being tested in clinical trials!

Scientific Machine Learning Gives More Realistic Results than Pure ML

Physically-Informed Machine Learning

$$\dot{\beta} \approx \frac{a_y}{u} - \frac{a_x}{u} - r$$



Using knowledge of the physical forms as part of the design of the neural networks.

New Architecture: DigitalEcho

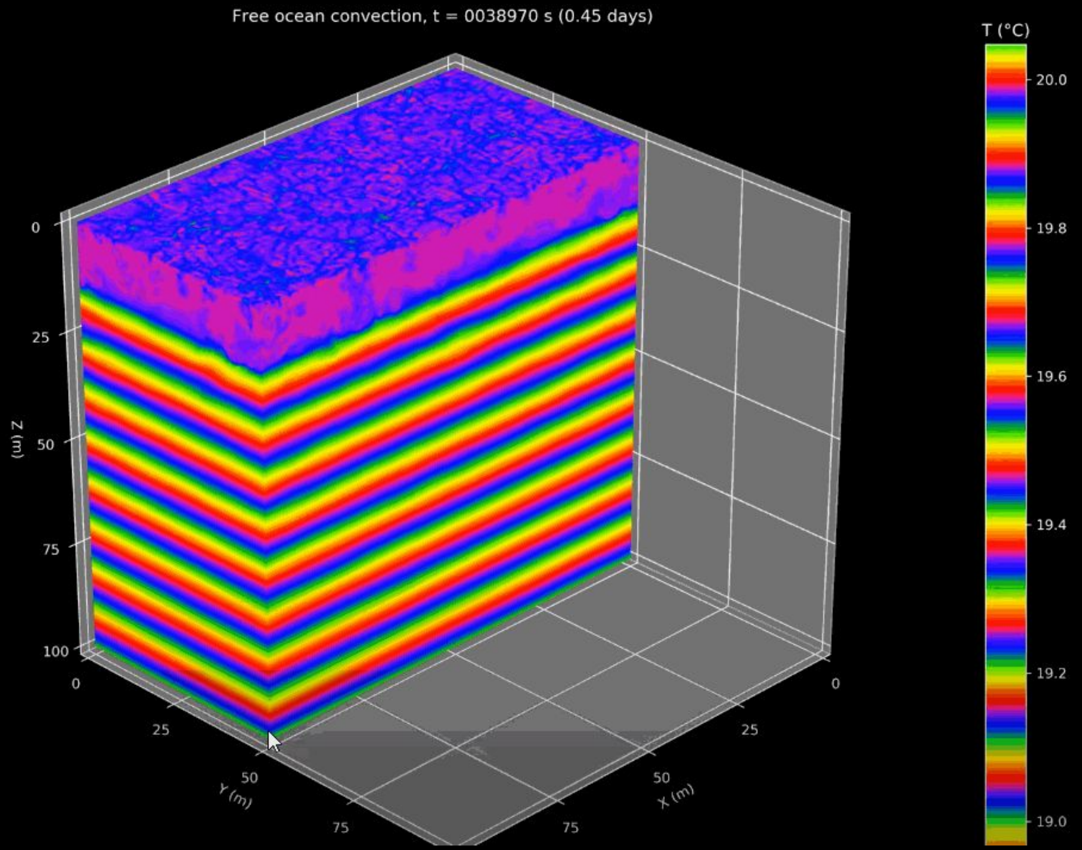
Smoother, more accurate results

High fidelity surrogates of ocean columns for climate models

3D simulations are high resolution but too expensive.

Can we learn faster models?

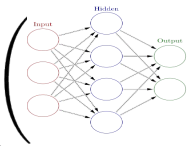
Ramadhan, Ali, John Marshall, Andre Souza, Gregory LeClaire Wagner, Manvitha Ponnapati, and Christopher Rackauckas. "Capturing missing physics in climate model parameterizations using neural differential equations." *arXiv preprint arXiv:2010.12559* (2022).



Neural Networks Infused into Known Partial Differential Equations

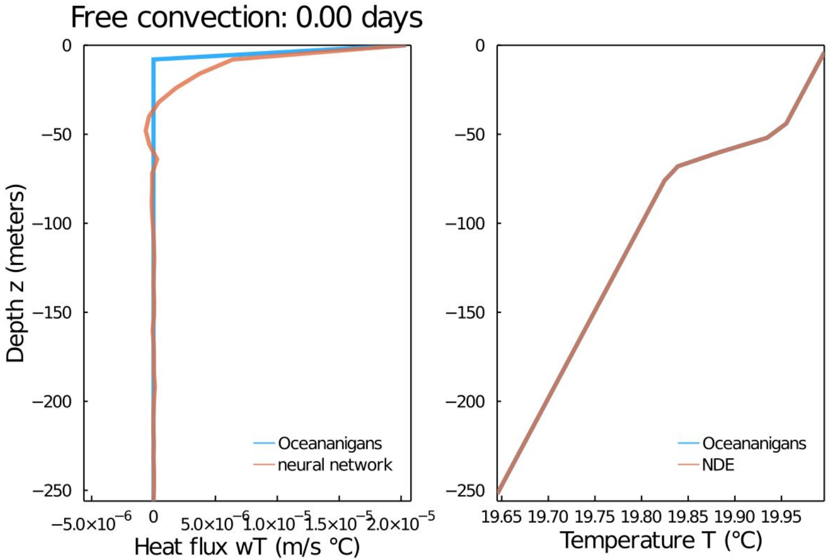
Derive a 1D approximation to the 3D model

$$\frac{\partial T}{\partial t} = -\frac{\partial}{\partial z}\underbrace{\left(\quad - K\frac{\partial T}{\partial z}\right)}_{\overline{w'T'}}$$



Incorporate the “convective adjustment”

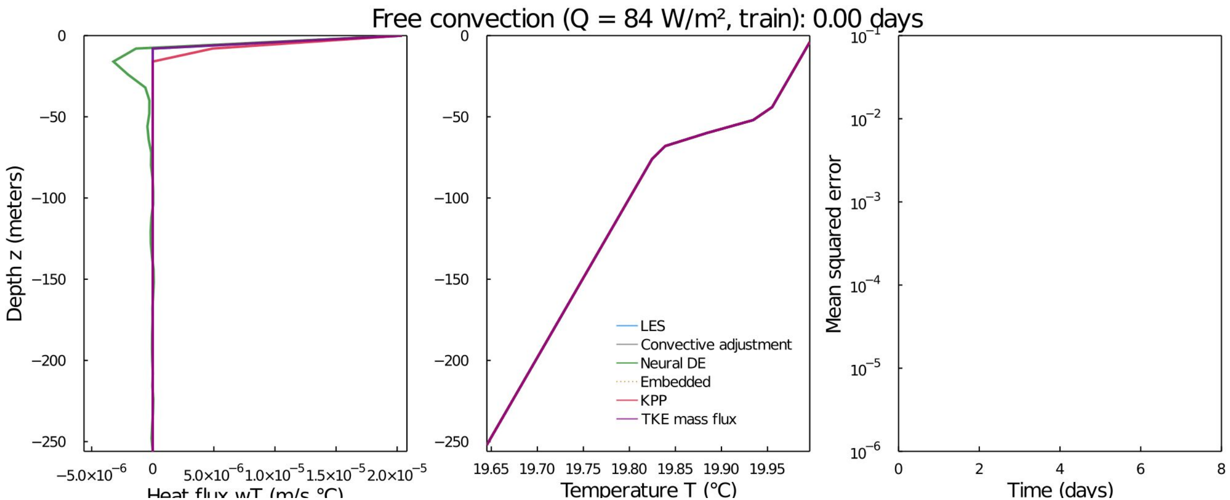
$$K = \begin{cases} 0 & \text{if } \partial_z T > 0 \\ 100\ \mathrm{m}^2/\mathrm{s} & \text{if } \partial_z T < 0 \end{cases}$$

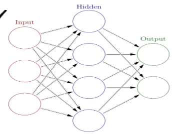


$$loss(T, wT) = |NN(T) - wT|^2$$

Training against datasets: only okay

Simulation + Machine Learning = Success





$$\frac{\partial T}{\partial t} = -\frac{\partial}{\partial z}\underbrace{\left(\qquad - K\frac{\partial T}{\partial z}\right)}_{\overline{w'T'}}$$

$$loss(T_{NN}, T) = |T_{NN}(z,t) - T(z,t)|^2$$

Integrating the simulator into training!

SciML: A Pervasive Ecosystem of Well-Documented Differentiable Packages

LinearSolve.jl: Unified Linear Solver Interface

$$A(p)x = b$$

NonlinearSolve.jl: Unified Nonlinear Solver Interface

$$f(u,p) = 0$$

DifferentialEquations.jl: Unified Interface for all Differential Equations

$$u' = f(u,p,t)$$

$$du = f(u,p,t)dt + g(u,p,t)dW_t$$

$$\vdots$$

Optimization.jl: Unified Optimization Interface

$$\text{minimize } f(u,p)$$

$$\text{subject to } g(u,p) \leq 0, h(u,p) = 0$$

Integrals.jl: Unified Quadrature Interface

$$\int_{lb}^{ub} f(t,p)dt$$

Unified Partial Differential Equation Interface

$$u_t = u_{xx} + f(u)$$

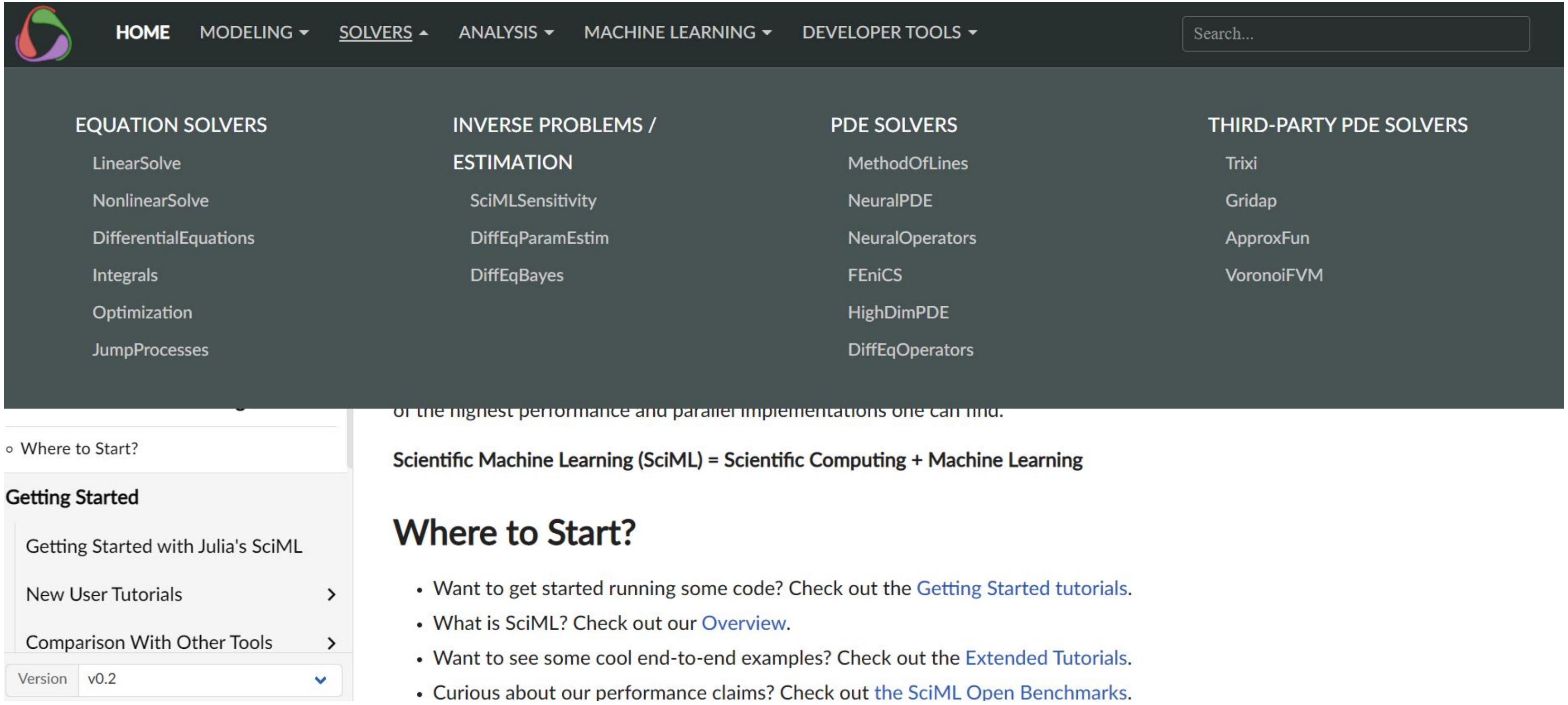
$$u_{tt} = u_{xx} + f(u)$$

$$\vdots$$

The SciML Common Interface for Julia Equation Solvers

All are compatible with Neural Networks and Scientific Machine Learning

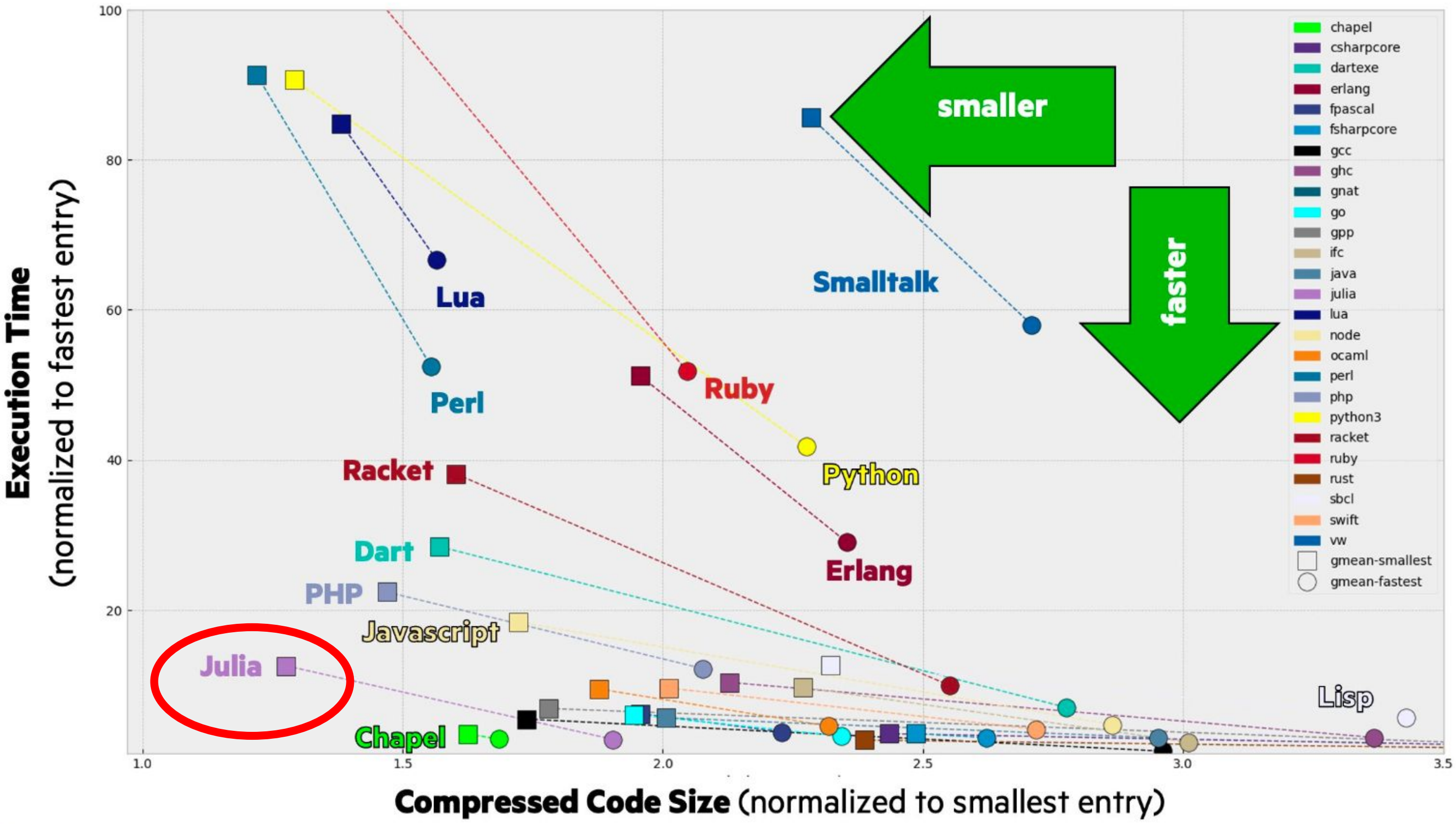
SciML Docs: Comprehensive Documentation of Differentiable Simulation



Why is Julia leading Scientific Machine Learning?

April, 2023

Productivity vs. Performance



Foundation: Fast Differential Equation Solvers

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

DifferentialEquations.jl is generally:

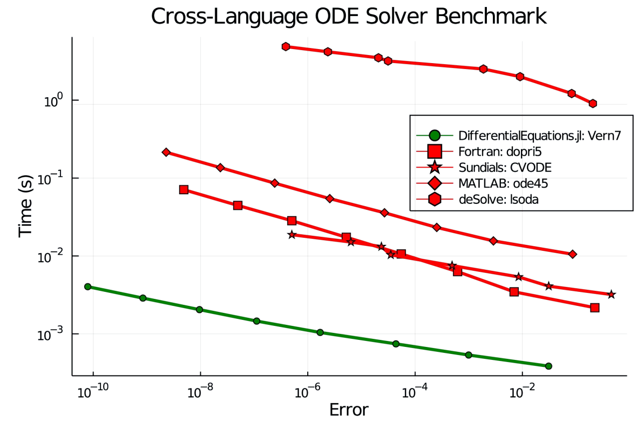
- 50x faster than SciPy
- 50x faster than MATLAB
- 100x faster than R's deSolve

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

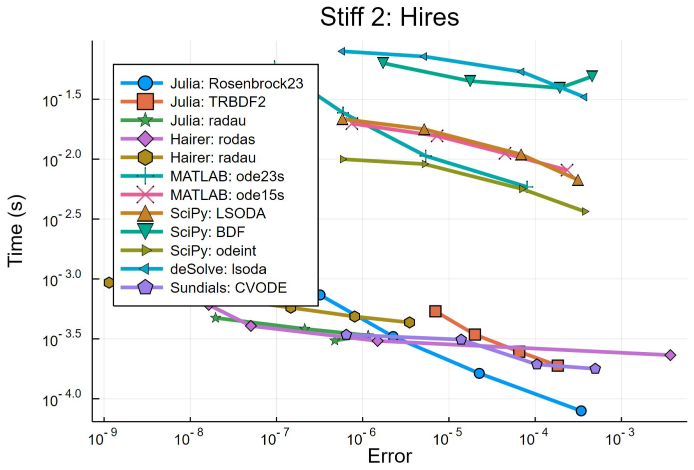
Rackauckas, Christopher, and Qing Nie. "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia." Journal of Open Research Software 5.1 (2017).

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

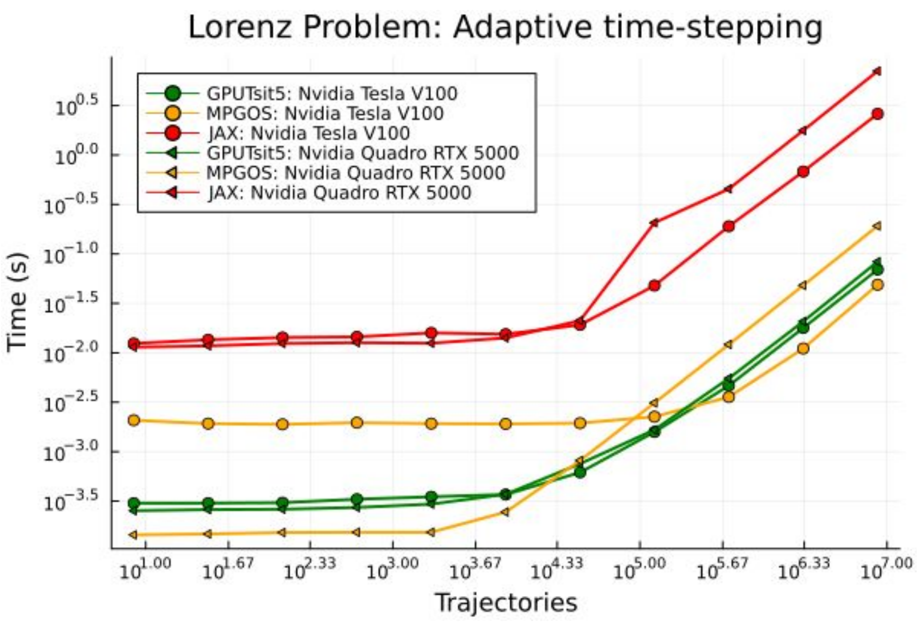
Non-Stiff ODE: Rigid Body System

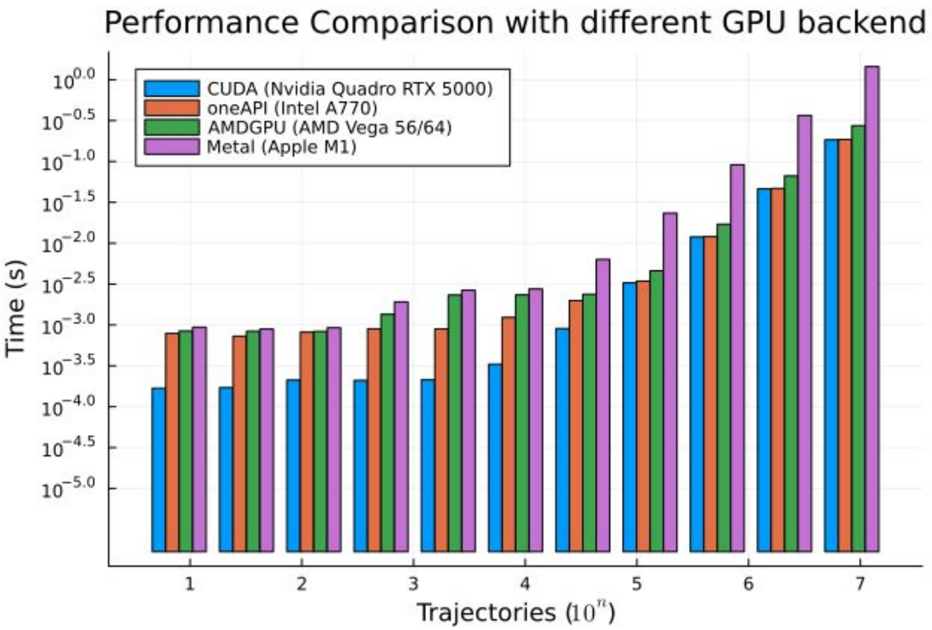


8 Stiff ODEs: HIRES Chemical Reaction Network



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

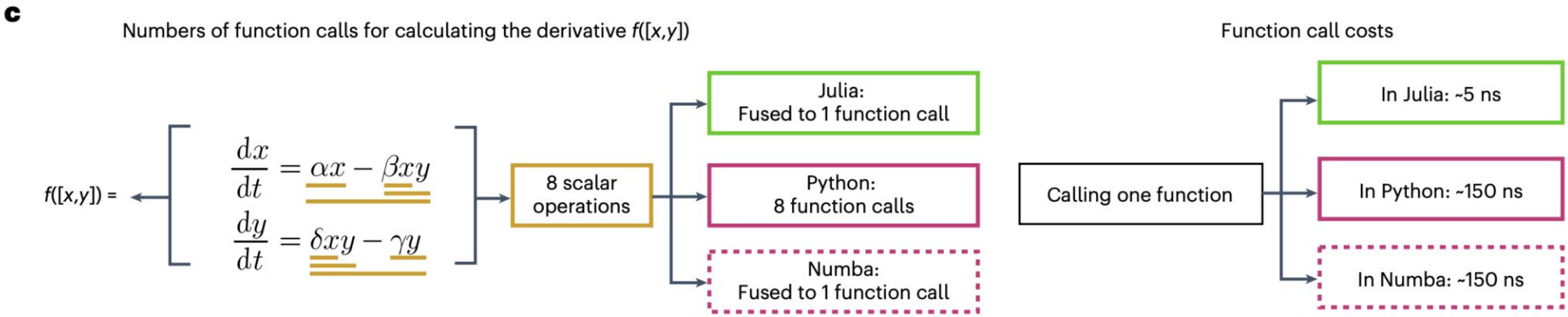




Matches State of the Art on CUDA, but also works with AMD, Intel, and Apple GPUs

Paper coming soon…

Understanding Julia’s Performance: Why is a JIT on Python not enough?



Julia for Biologists (Nature Methods)

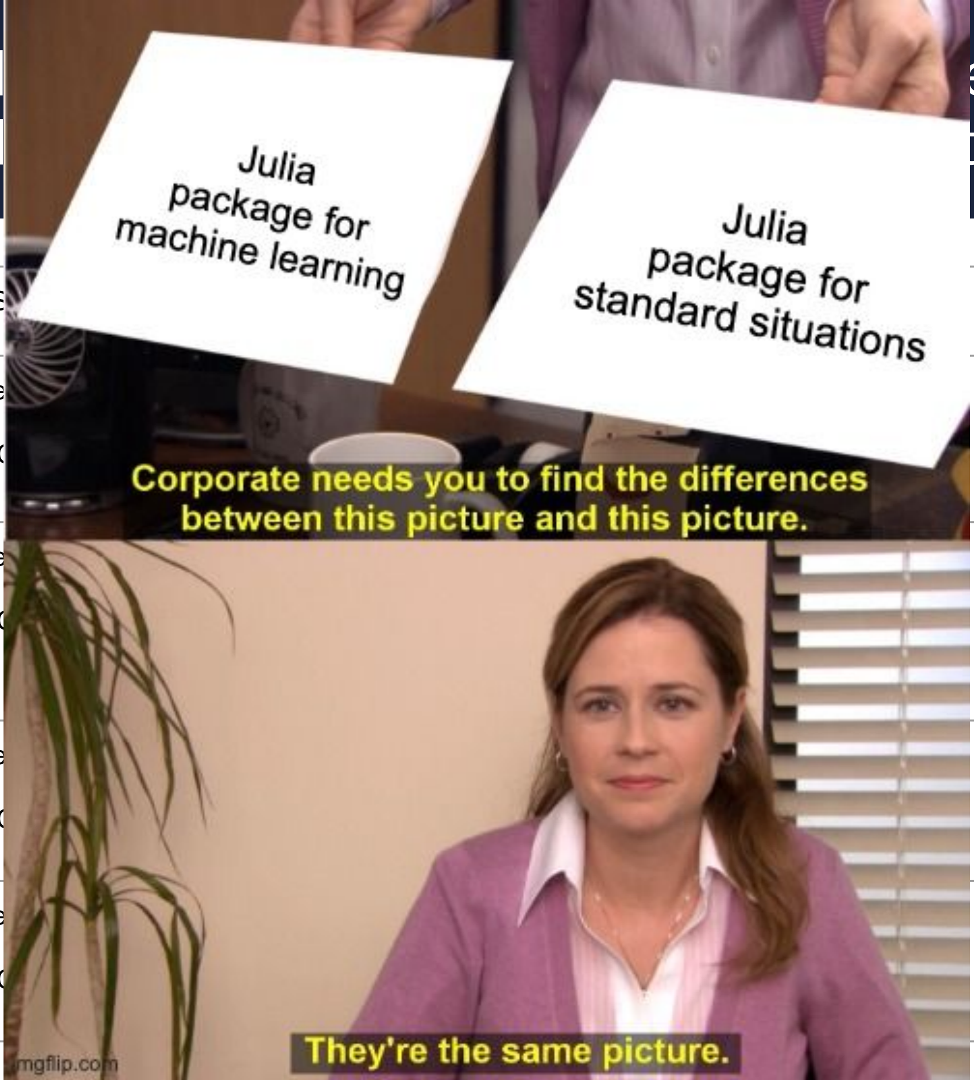
Theoretically inferred and real-time calculation of *f*([*x*,*y*])

	Time of array allocation	+	Time of floating point operations	+	Time of function calls	=	Inferred time	Real time
Julia			8 × 2 ns	+	1 × 5 ns	=	21 ns	20 ns
Python	300 ns	+	8 × 2 ns	+	8 × 150 ns	=	1,516 ns	1,510 ns
Numba	300 ns	+	8 × 2 ns	+	1 × 150 ns	=	466 ns	425 ns

Understanding Julia’s Development Speed: Why are Julia packages growing faster and better tested?

	Julia Scientific Computing	Julia Machine Learning	Normal Python	PyTorch
ODE Solver	DifferentialEquations.jl >100 Contributors	DifferentialEquations.jl >100 Contributors	SciPy.odeint ~5 developers	Torchdiffeq 1 contributor with more than one contribution
SDE Solver	DifferentialEquations.jl >100 Contributors	DifferentialEquations.jl >100 Contributors	N/A	TorchSDE 2 contributors with more than one contribution (last commit July 2021)
DDE Solver	DifferentialEquations.jl >100 Contributors	DifferentialEquations.jl >100 Contributors	N/A	N/A
DAE Solver	DifferentialEquations.jl >100 Contributors	DifferentialEquations.jl >100 Contributors	N/A	N/A

Und… eed:
Why are Jul… tter tested?



	Julia S…	PyTorch
ODE Solver	Differe… >100 …	Torchdiffeq 1 contributor with more than one contribution
SDE Solver	Differe… >100 …	TorchSDE 2 contributors with more than one contribution (last commit July 2021)
DDE Solver	Differe… >100 …	N/A
DAE Solver	Differe… >100 …	N/A

Understanding Julia’s Package Ecosystem: Can Composability of Features be Automatic?

julia

DifferentialEquations.jl and Measurements.jl

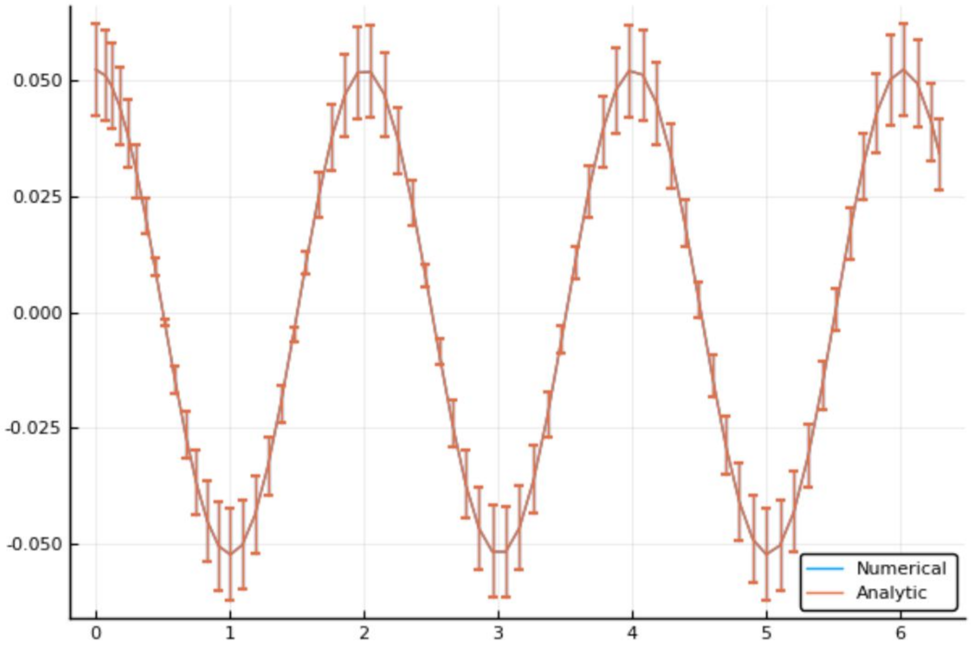
Usage diffeq

giordano Oct '17

Today I was asked whether it was possible to solve in Julia differential equations involving numbers with uncertainties. Of course the answer is yes. What I find really amazing about Julia is that the two packages don't know anything about each other, yet they can work together without any effort. Here is an short example based on this tutorial 23: https://nbviewer.jupyter.org/gist/giordano/e82a3959d8f64301129d64d004e10b4e 99

Understanding Julia’s Package Ecosystem: Can Composability of Features be Automatic?

```
using DifferentialEquations, Measurements, Plots

pyplot()

g = 9.79 ± 0.02; # Gravitational constants
L = 1.00 ± 0.01; # Length of the pendulum

#Initial Conditions
u₀ = [0 ± 0, π / 60 ± 0.01] # Initial speed and initial angle
tspan = (0.0, 6.3)

#Define the problem
function simplependulum(du,u,p,t)
    θ  = u[1]
    dθ = u[2]
    du[1] = dθ
    du[2] = -(g/L)*θ
end

#Pass to solvers
prob = ODEProblem(simplependulum, u₀, tspan)
sol = solve(prob, Tsit5(), reltol = 1e-6)

# Analytic solution
u = u₀[2] .* cos.(sqrt(g / L) .* sol.t)

plot(sol.t, getindex.(sol.u, 2), label = "Numerical")
plot!(sol.t, u, label = "Analytic")
```



One Language: More Performance

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

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

Covariates

Requires special fitting procedures (Pumas)

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

Structural Model (pre)

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

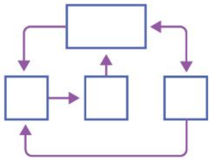
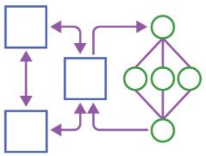
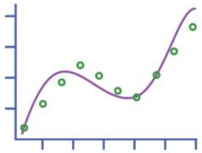
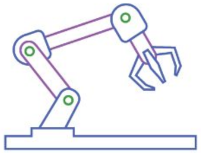
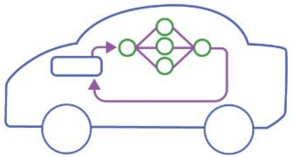
Dynamics

JuliaHub

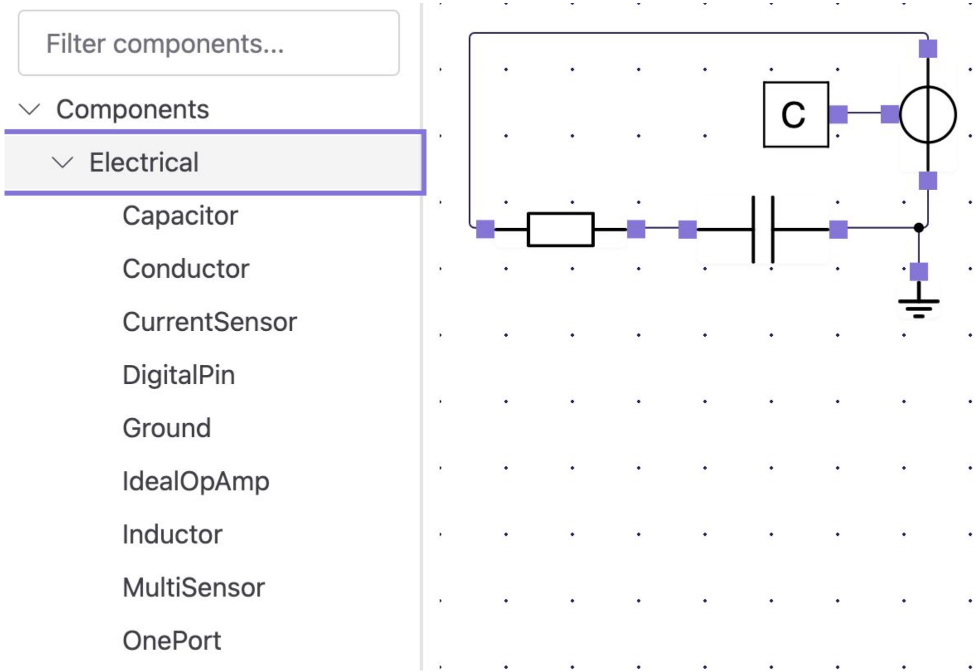
Julia is Building Tools for High-Tech Enterprise

April, 2023

JuliaSim at a Glance

Design	Discover	Calibrate	Control	Surrogatize
• Build realistic physical models with minimal code • Run simulations 100x faster	• Use Machine Learning to autocomplete models • Discover missing physics	• Turn models into Digital Twins • Robust nonlinear fitting with automatic differentiation	• Build robust nonlinear controls • Deploy Model-Predictive Controllers (MPC)	• Train neural networks to match models • Accelerate fast simulations by another 100x
				

All in a point-and-click GUI



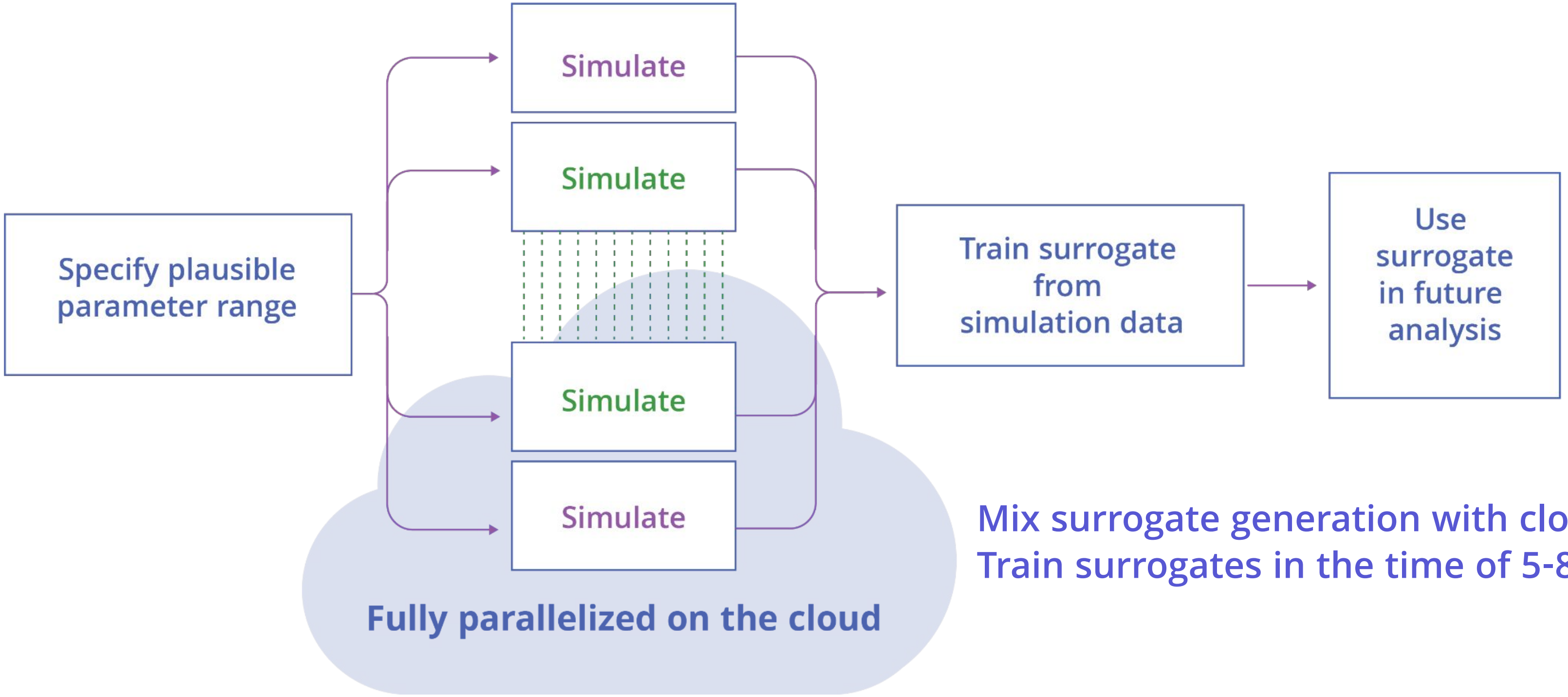
Generate Digital Twins and calibrate models

System parameters

ID	Lower Bound	Upper Bound	Nominal	Tunable
k1	1	100	10	☑
k2	10	50	25	☐
k3	5	250	125	☑
k_drug	10	50	25	☐

Tune nonlinear controllers
Deploy to embedded hardware

Power of the Cloud: Point and Click GUI Doesn't Sacrifice Performance



Mix surrogate generation with cloud compute:
Train surrogates in the time of 5-8 runs!

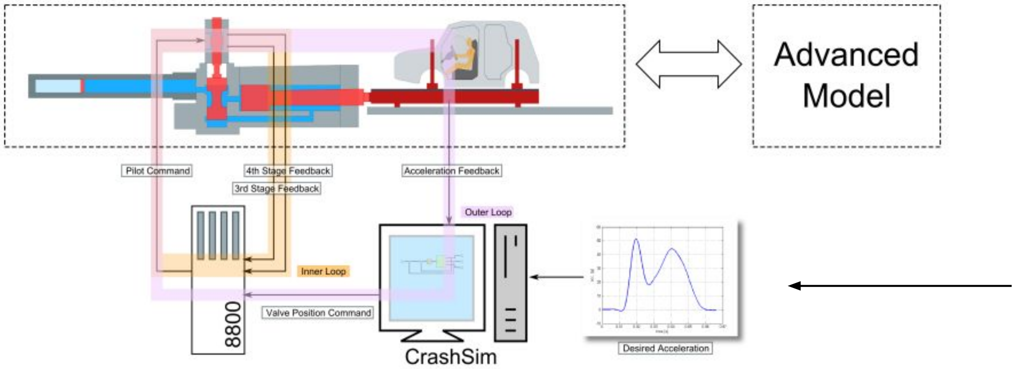
Catapult Project

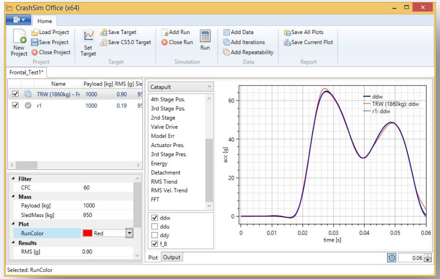
10/11/2022

Brad Carman

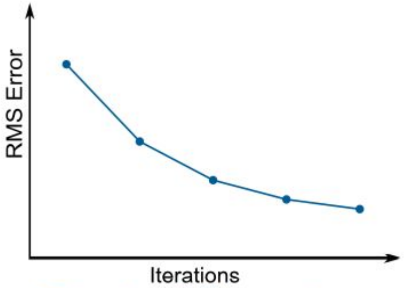
Julia SIM

Instron Hydropuls Catapult Introduction

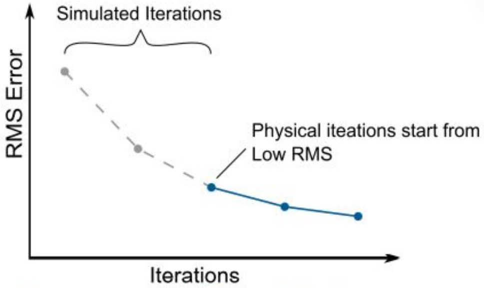




prediction software



Physical Iterations



Improved Prediction

Model History: >10,000x over Simulink, and Beyond

2.5kHz → 10kHz →

2000

- Inverse Model: Transfer functions
- Forward Model: **Simulink**

2014

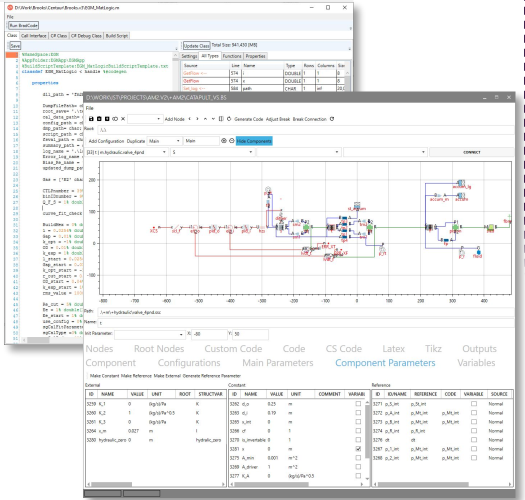
- I joined Instron
- Built Implicit Newton/Euler Equation Based model in pure **Matlab** with inverse and subset model generator using Symbolic Toolbox
- Increased model accuracy with elimination of assumptions and increased complexity
- Worked well, but…
  - *Slow*
  - *Hard to update and maintain*

2017

- Attempted to move to **SimScape**
- Successfully transitioned model with improved speed, but required many workarounds and hacks
- *Never released…*

2020

- Moved to **Julia**
- Developed *EmbeddedJulia* library, *ModelingToolkitComponents.jl* and successfully transitioned model to **ModelingToolkit.jl**



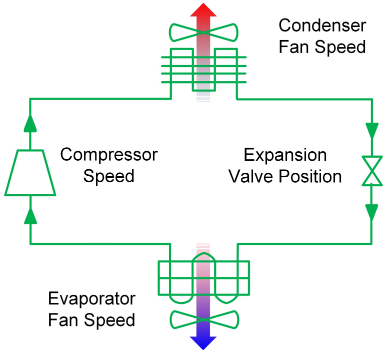
Matlab2CSharp and SimScape Manager

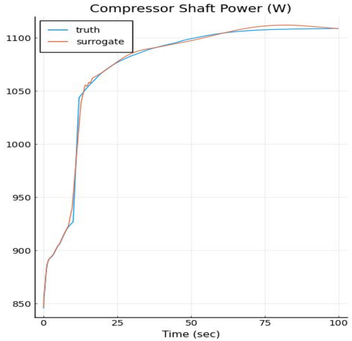
>1000x performance improvements over Simulink!

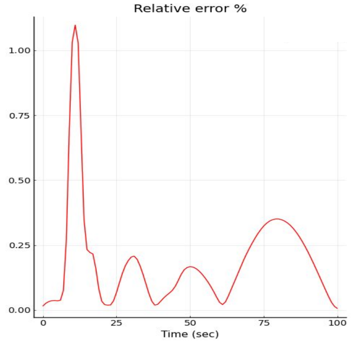
ARPA-E
Accelerated Simulation of Building Energy Efficiency

8,000 ODE Highly stiff vapor-compression cycle model

MITSUBISHI ELECTRIC
Changes for the Better







The Julia implementation is 6x faster than Dymola for the full cycle simulation.

- Dymola reference model: 35.3 s
- Julia (as close to) equivalent model: 5.8 s
- Could be due to details such as the linear solvers, the refrigerant property libraries, etc. More benchmarking to come.

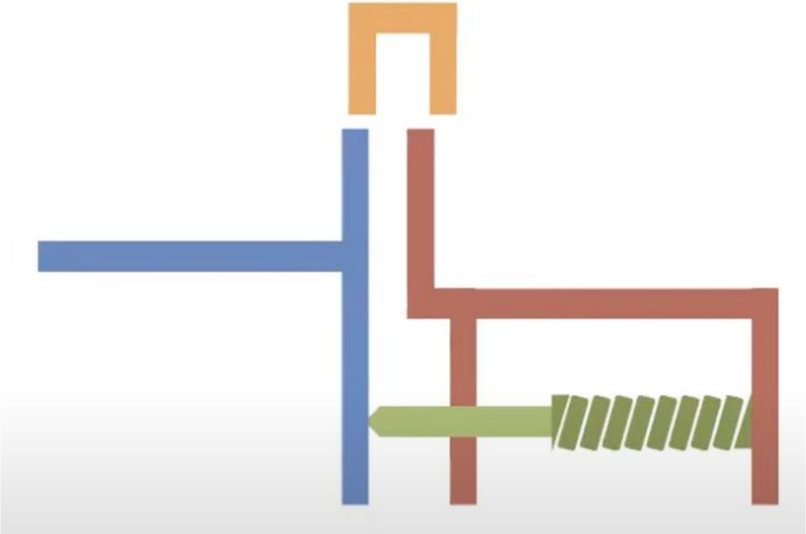
Using CTESNs as surrogates improves simulation times between 10x-95x over the Julia baseline. Acceleration depends on the size of the reservoir in the CTESN. **The surrogate approximates 20 of the observables.**

Training set size	Reservoir size	Prediction time	Speedup over baseline
100	1000	0.06 s	95x
1000	2000	0.56 s	10x

Error is < 5% in all cases.

Total speedup over Dymola: 60-570x

NASA Launch Services:
Deploying Julia to Replace Simulink



Day long cluster compute analysis turned into an interactive webapp!
Youtube: Modeling Spacecraft Separation Dynamics in Julia - Jonathan Diegelman

US Air Force Research Laboratory

1. Robust Controls
2. Optimal control under uncertainty
3. Deployment onto embedded hardware
4. Nonlinear control of unmanned vehicles (UAVs / Drones)

INFORMATION SYSTEMS

Year of autonomy in Alaskan glaciers, flight, Earth orbit, cislunar space and Mars

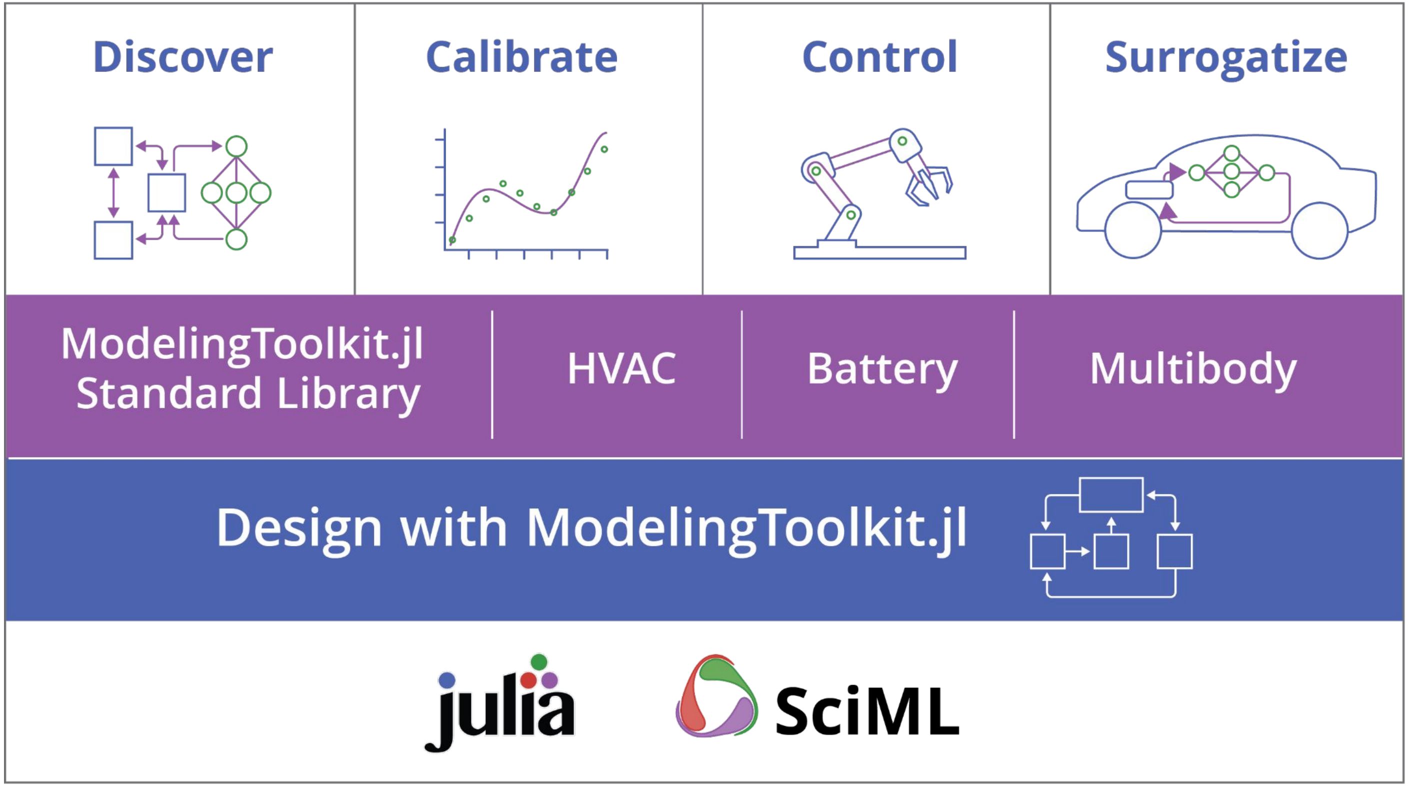
BY KERIANNE HOBBS

The **Intelligent Systems Technical Committee** works to advance the application of computational problem-solving technologies and methods to aerospace systems.

In June, the **U.S. Air Force Research Laboratory's Intelligent Control and Evaluation of Teams** flight test program flew an uncrewed aerial system in coordination with ground systems to provide aerial support in virtually contested environments. The flight test team was able to demonstrate this on a **vertical takeoff and landing vehicle** with both electric and conventional fuel propulsion systems onboard. The UAS was able to plan and execute these missions autonomously using onboard hardware. It was the first time the **Julia programming language** was flown on the embedded hardware — algorithms were precompiled ahead of time. The algorithms used to perform the various missions involved feedback control, mixed-integer linear programming and optimal control.

In November, NASA's **Cislunar Autonomous Positioning System Technology Operations and Navigation Experiment** arrived at its near-rectilinear halo orbit around the moon. Mission controllers regained control of CAPSTONE in October after the spacecraft began spinning in September, likely due to a stuck thruster valve. Among the mission objectives are demonstrating autonomous orbit determination in cislunar space. Using ranging measurements from the **Lunar Reconnaissance Orbiter**, the CAPSTONE spacecraft can determine its orbital position and perform stationkeeping maneuvers without the need for Earth-based localization, paving a way for greater numbers of more independent cislunar and deep space probes.

JuliaSim Architecture



SciML Open Source Software Organization
sciml.ai

- DifferentialEquations.jl: 2x-10x Sundials, Hairer, …
- DiffEqFlux.jl: adjoints outperforming Sundials and PETSc-TS
- ModelingToolkit.jl: 15,000x Simulink
- Catalyst.jl: >100x SimBiology, gillespy, Copasi
- DataDrivenDiffEq.jl: >10x pySindy
- NeuralPDE.jl: ~2x DeepXDE* (more optimizations to be done)
- NeuralOperators.jl: ~3x original papers (more optimizations required)
- ReservoirComputing.jl: 2x-10x pytorch-esn, ReservoirPy, PyRCN
- SimpleChains.jl: 5x PyTorch GPU with CPU, 10x Jax (small only!)
- DiffEqGPU.jl: Some wild GPU ODE solve speedups coming soon

And 100 more libraries to mention…

If you work in SciML and think optimized and maintained implementations of your method would be valuable, please let us know and we can add it to the queue.

Democratizing SciML via pedantic code optimization
Because we believe full-scale open benchmarks matter

Sioux Hot or Not
ASML & Sioux & Eindhoven

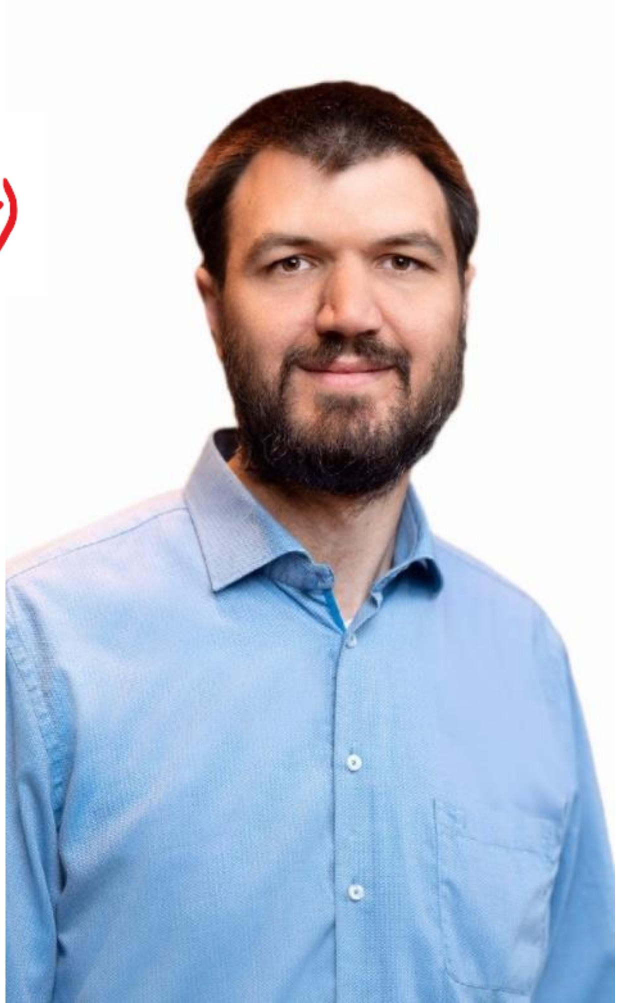
Keith Myerscough & Matthijs Cox

April 21, 2023

CREATION DATE: YYYY-MM-DD

Keith Myerscough

Mathware Designer



Matthijs Cox

Physicist & Metrology Architect

julia
eindhoven

www.meetup.com/
julialang-eindhoven

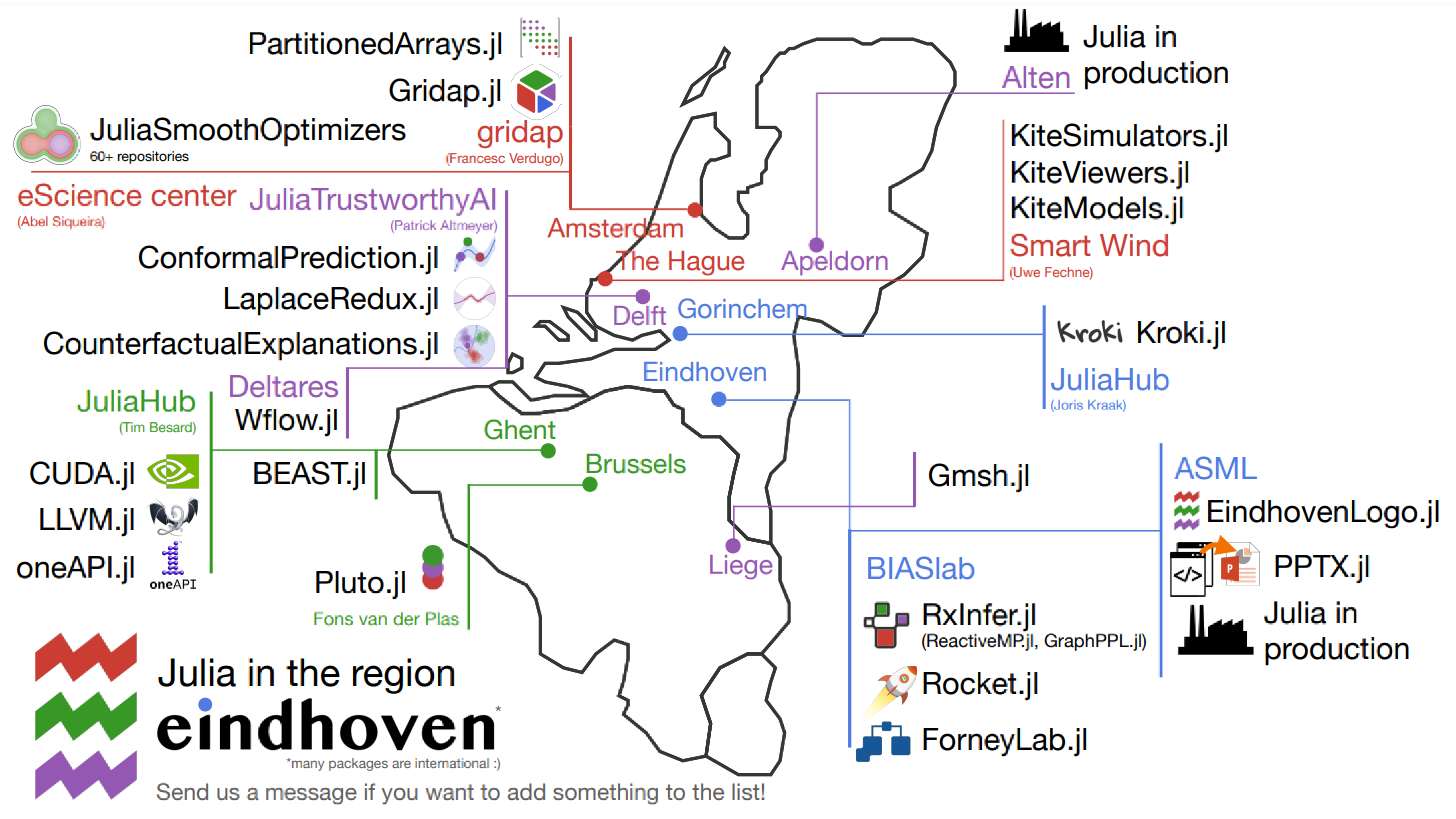
Events

Sep ’22	Embracing Change	HTC	50
Nov ‘22	How to Julia	TU/e	46
Jan ’23	Julia for Reals	Sioux Labs	37
Feb ’23	Julia Bonus Meetup	Philips Stadium	60*
May ‘23	???	???	>50

* Capacity limited!

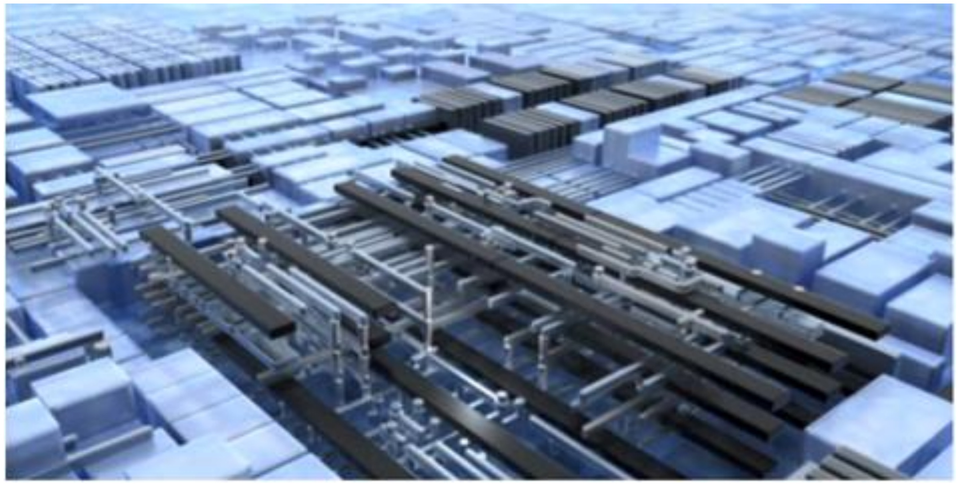
Currently looking for new hosts!

Low cost, low key: high fun, high value!



PartitionedArrays.jl
Gridap.jl
gridap
(Francesc Verdugo)
JuliaSmoothOptimizers
60+ repositories
eScience center
(Abel Siqueira)
JuliaTrustworthyAI
(Patrick Altmeyer)
ConformalPrediction.jl
LaplaceRedux.jl
CounterfactualExplanations.jl
Deltares
Wflow.jl
JuliaHub
(Tim Besard)
CUDA.jl
LLVM.jl
oneAPI.jl
oneAPI
BEAST.jl
Pluto.jl
Fons van der Plas
Amsterdam
The Hague
Apeldorn
Delft
Gorinchem
Eindhoven
Ghent
Brussels
Liege
Julia in production
Alten
KiteSimulators.jl
KiteViewers.jl
KiteModels.jl
Smart Wind
(Uwe Fechne)
Kroki
Kroki.jl
JuliaHub
(Joris Kraak)
Gmsh.jl
ASML
EindhovenLogo.jl
</>
P
PPTX.jl
Julia in production
BIASlab
RxInfer.jl
(ReactiveMP.jl, GraphPPL.jl)
Rocket.jl
ForneyLab.jl
Julia in the region
eindhoven*
*many packages are international :)
Send us a message if you want to add something to the list!

Why Julia for ASML?



ASML makes big systems for tiny patterns

Software and algorithms play a central role
to optimize the machines and processes

Typical ASML Algorithm Development Experience

Why does it take so long?
How can we accelerate?

We have a proven algorithm prototype in research

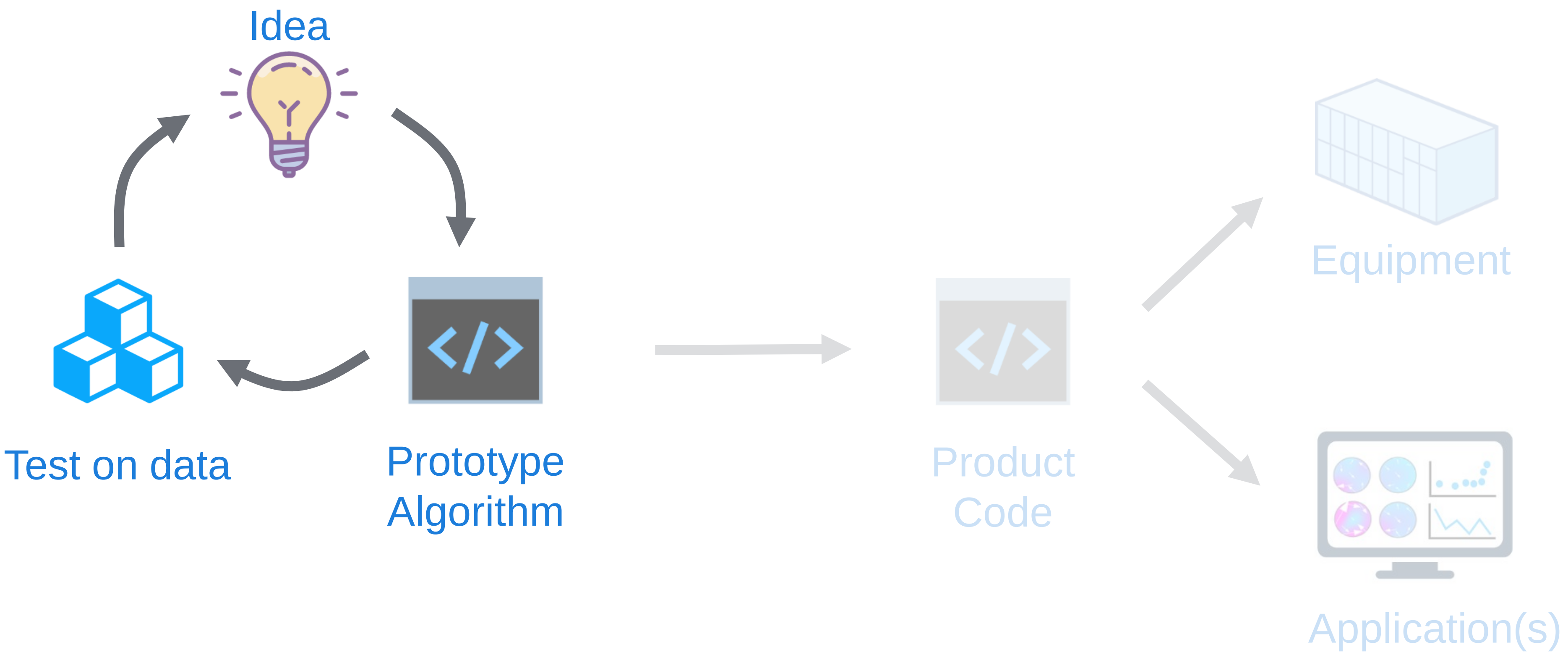
Then we spend years turning it into a product

Excited customer

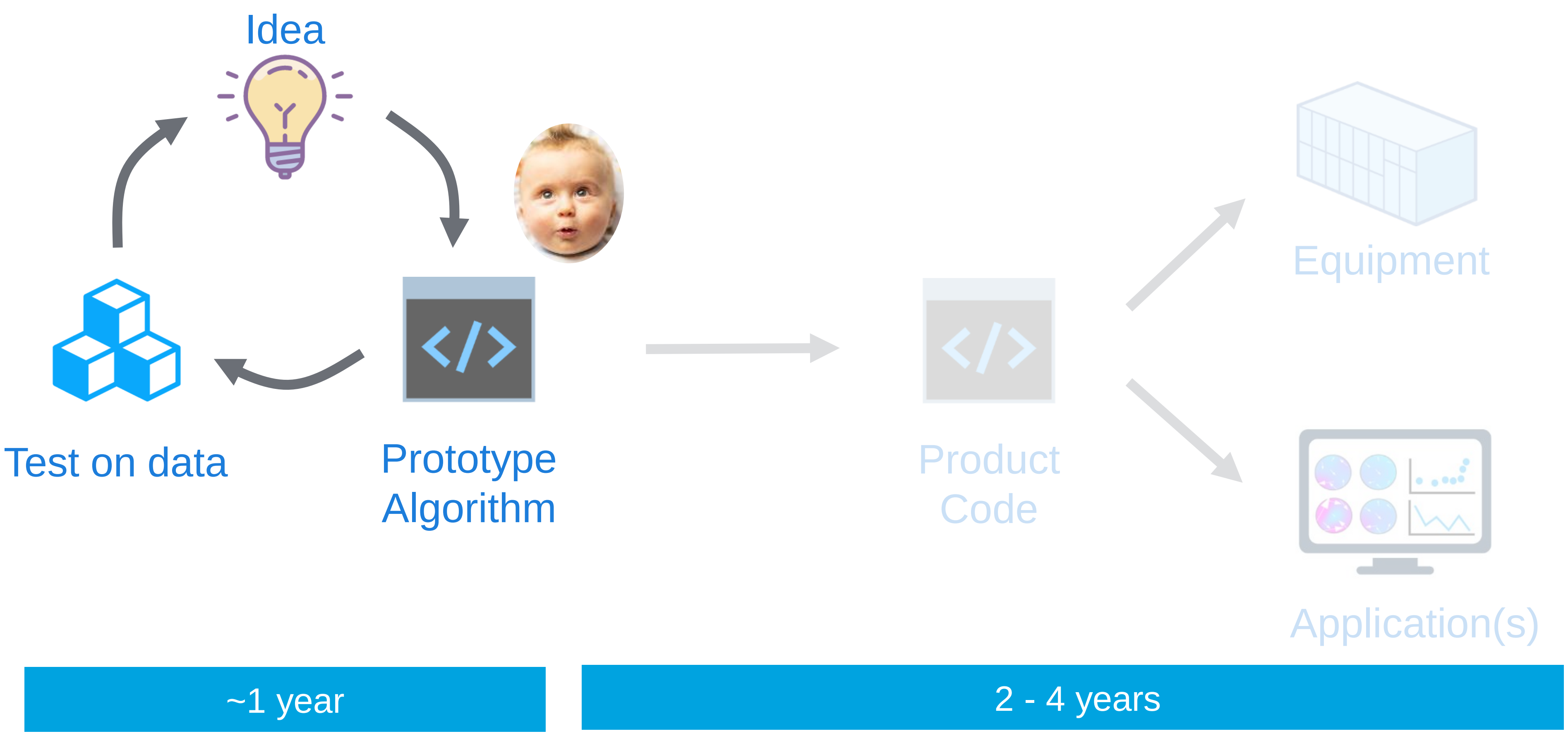
years later

Not so excited customer

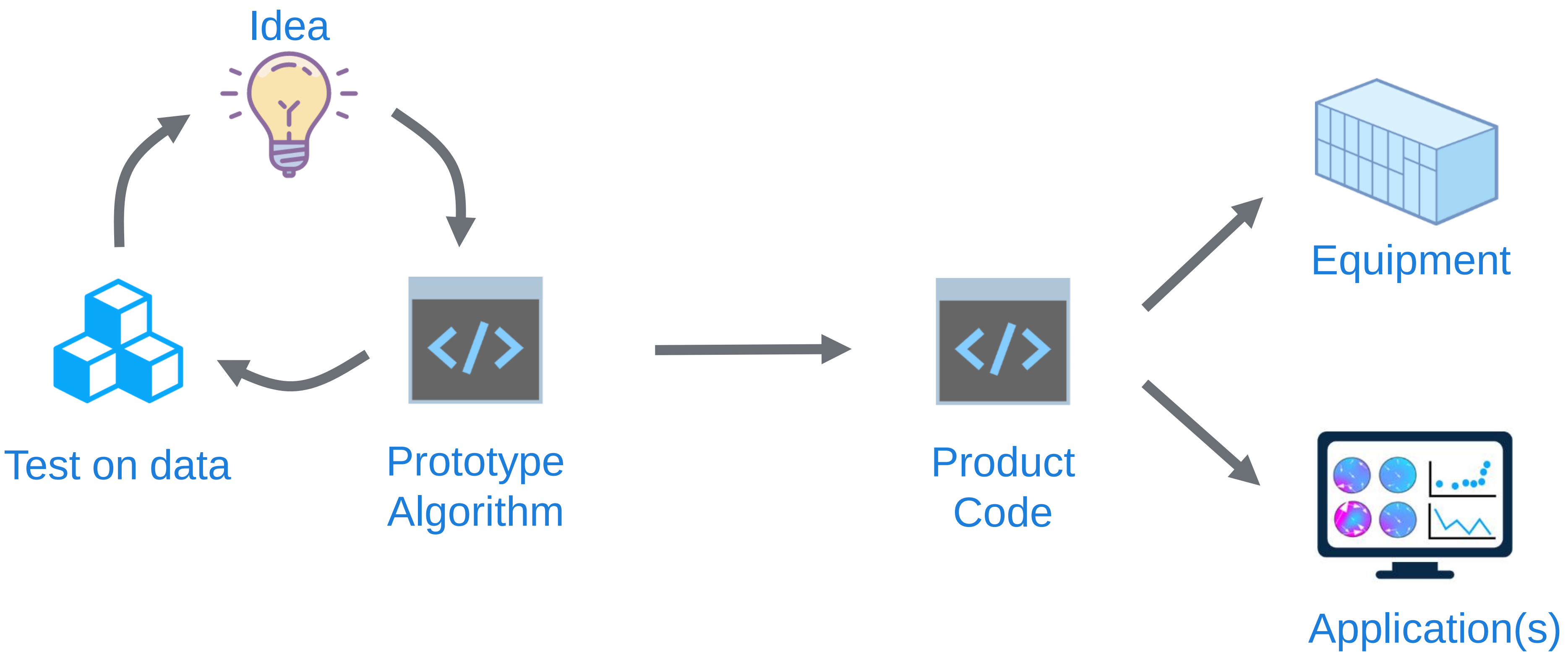
The Life of an Algorithm



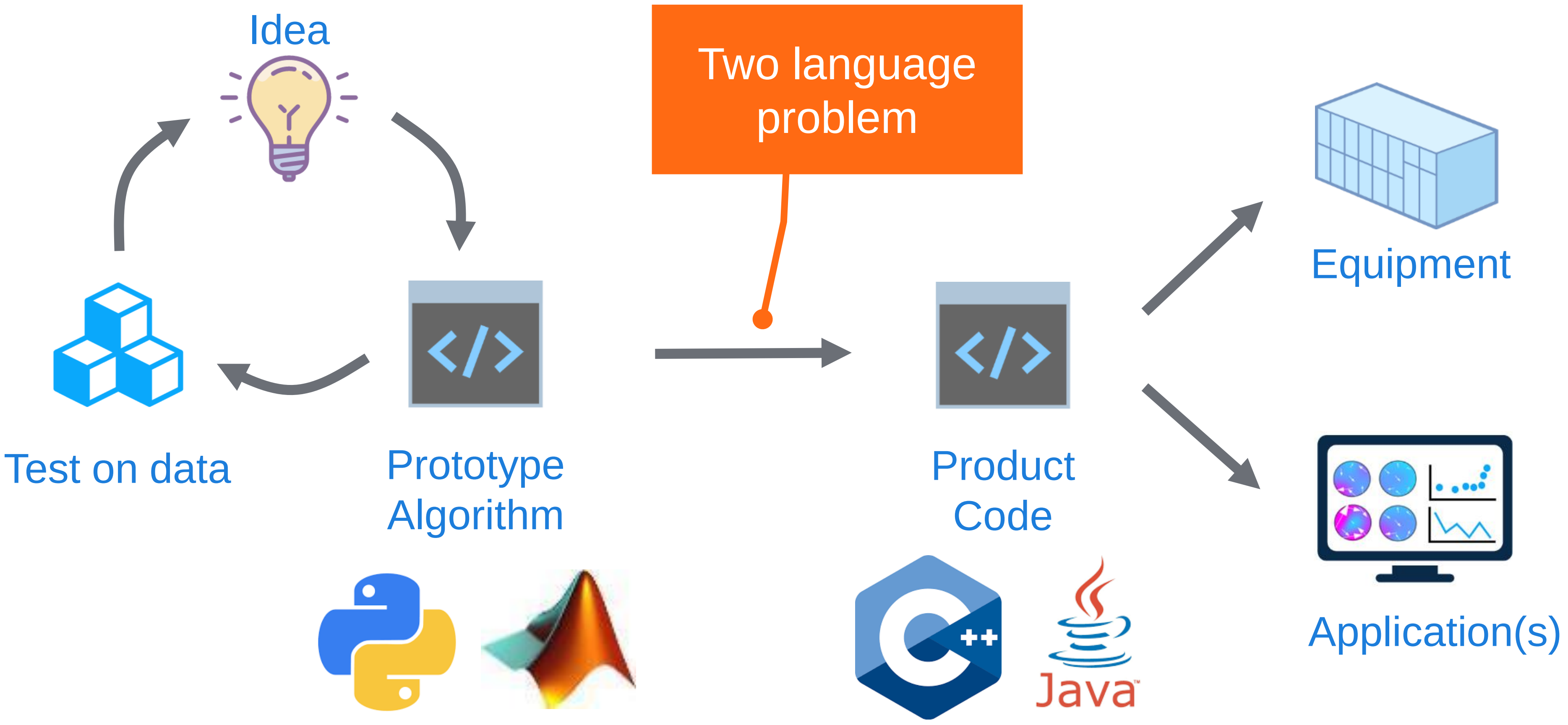
The Life of an Algorithm



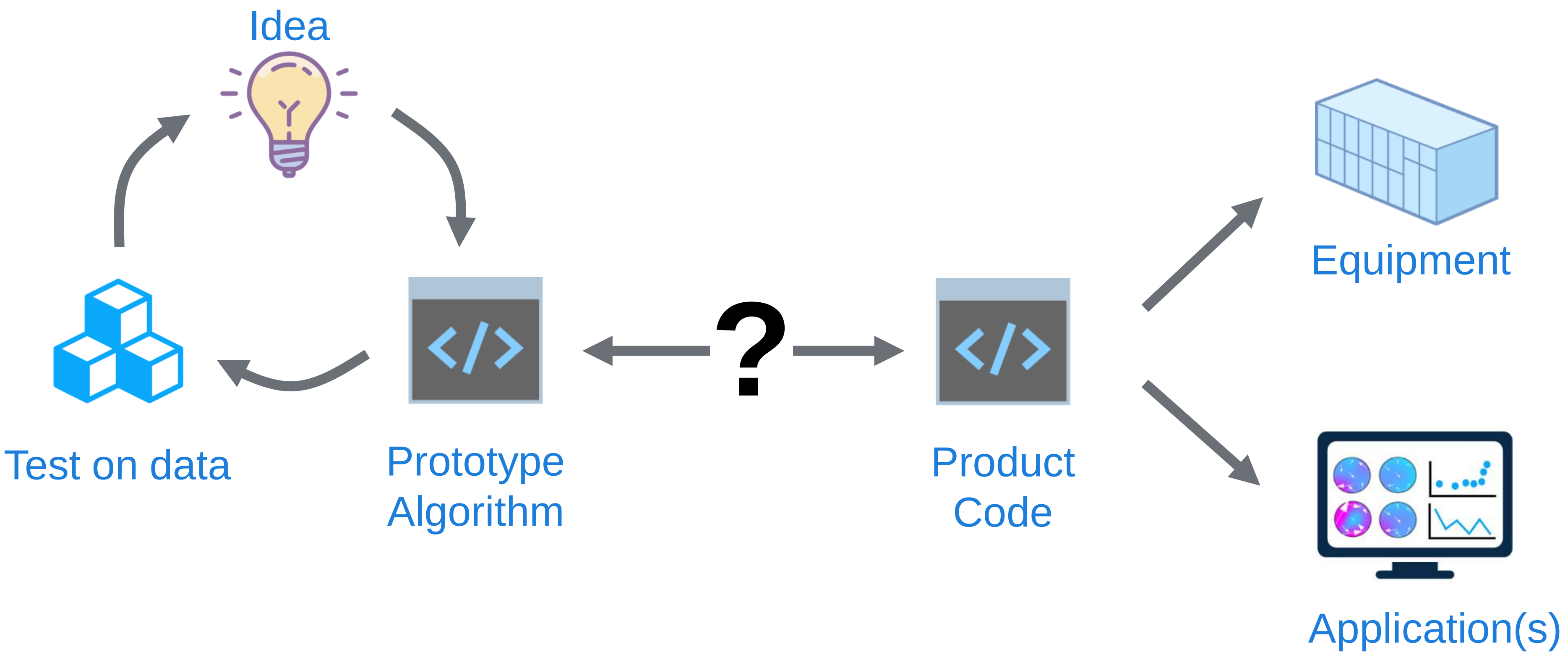
The Life of an Algorithm



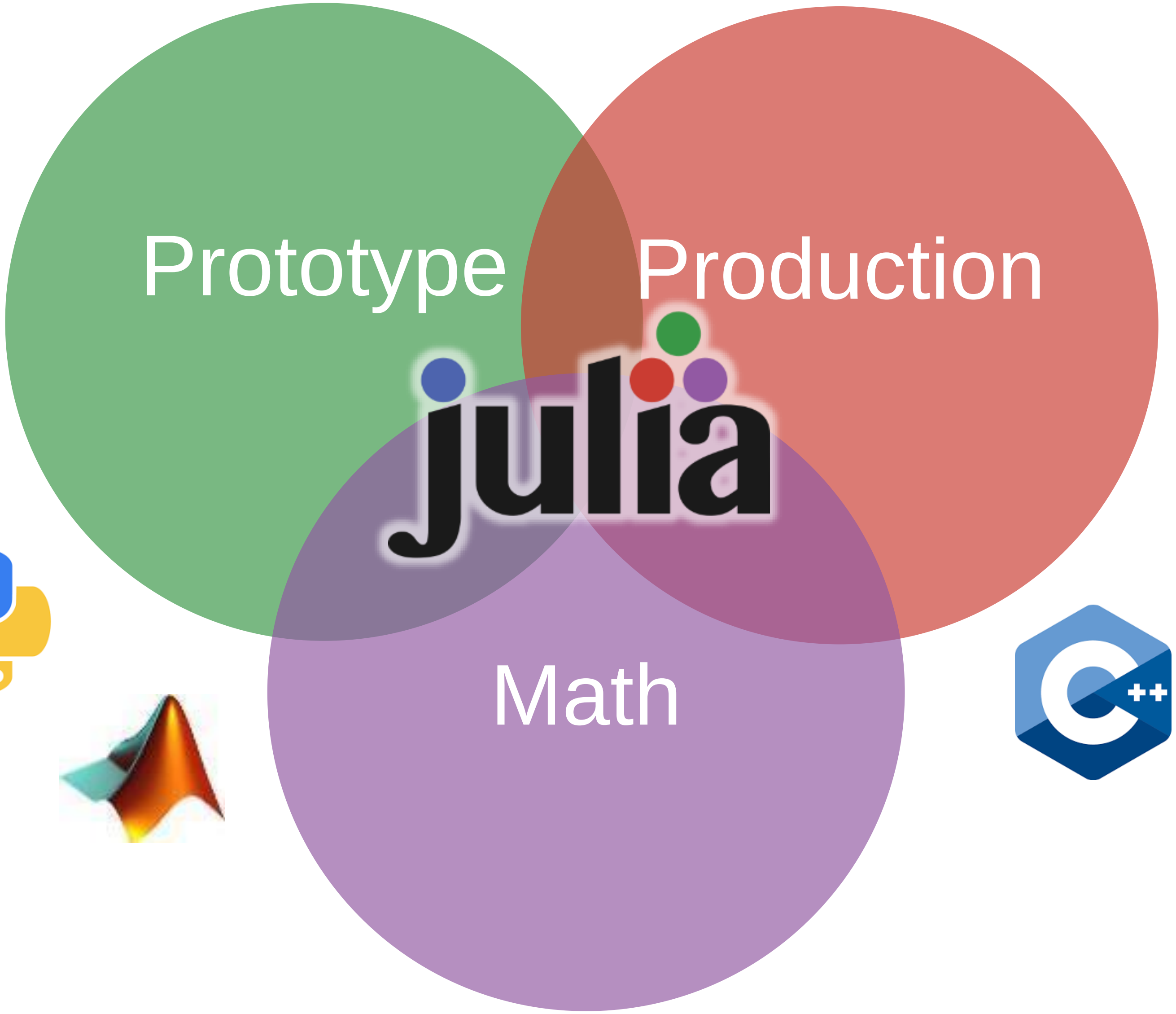
The Life of an Algorithm



The Life of an Algorithm



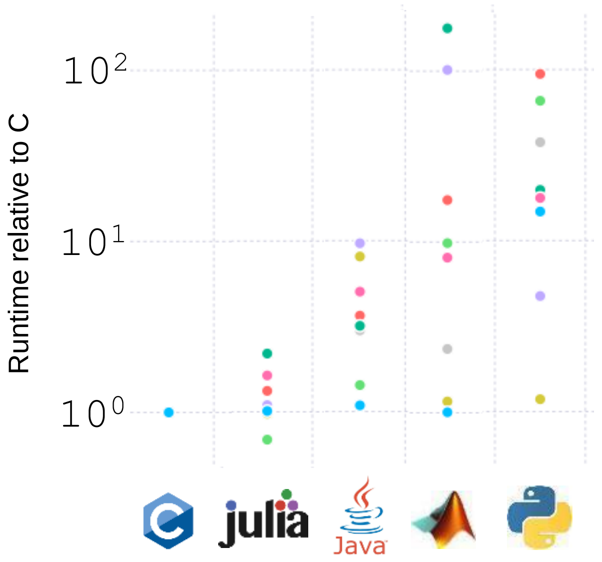
Technology needs for our algorithms



Julia is designed for the challenge we have at ASML:
fast and easy numerical computing

Fast

on a wide range of common code patterns



See more: https://julialang.org/benchmarks/

Great for Functionals

expressive and feels dynamic

```
A\b
sin.(2π*θ)
"a" ∈ [2, 4.1, "a"]
f(x, y=2) = x + y
2 ≤ x ≤ 5
```

Great for Software

typed, compiled and general purpose

```
struct Point
    x::Int
    y::String
end
function +(p::Point, z::Int)
    return Point(p.x + z, p.y)
end
```

All we need

Plots

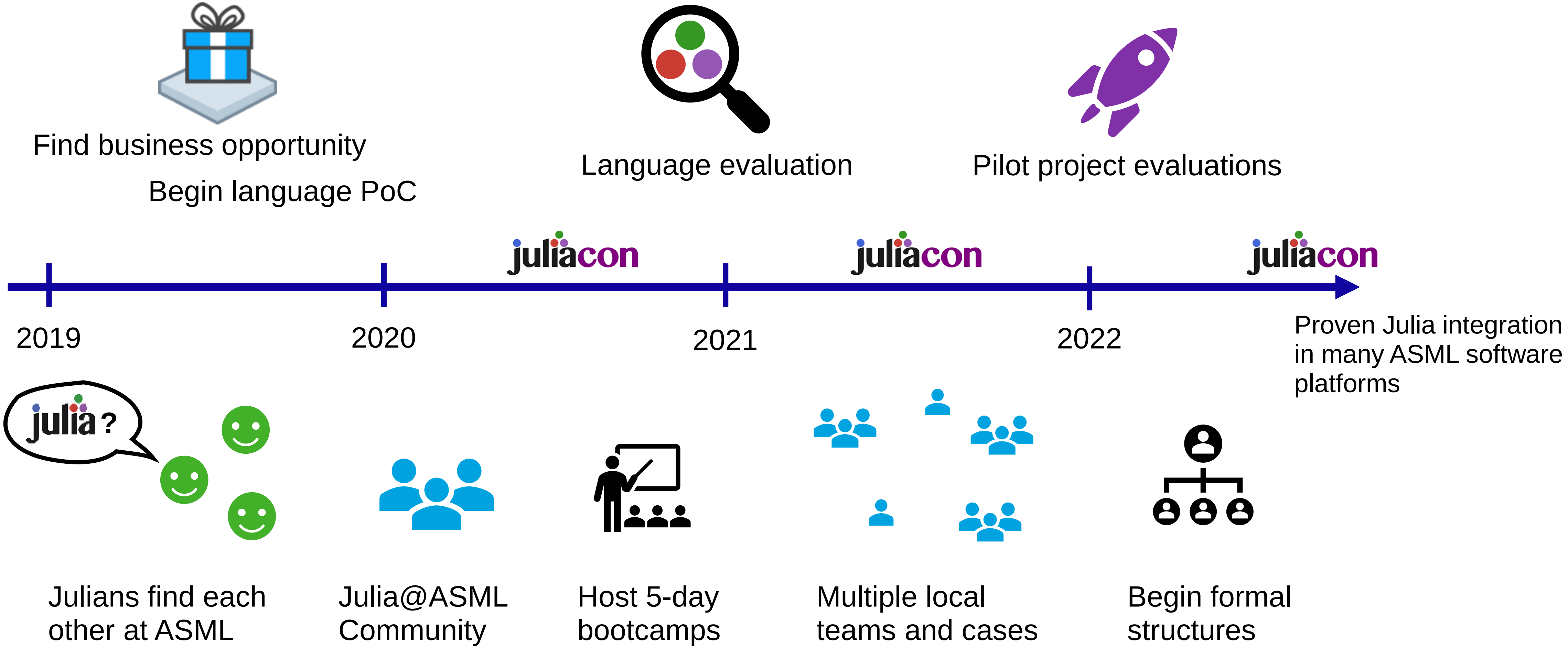
Linear Algebra

File IO

Machine Learning

And much more; all open source!

Julia progress at ASML



Our discovery of Julia’s unexpected benefits

Julia Package Management

Julia all the way down

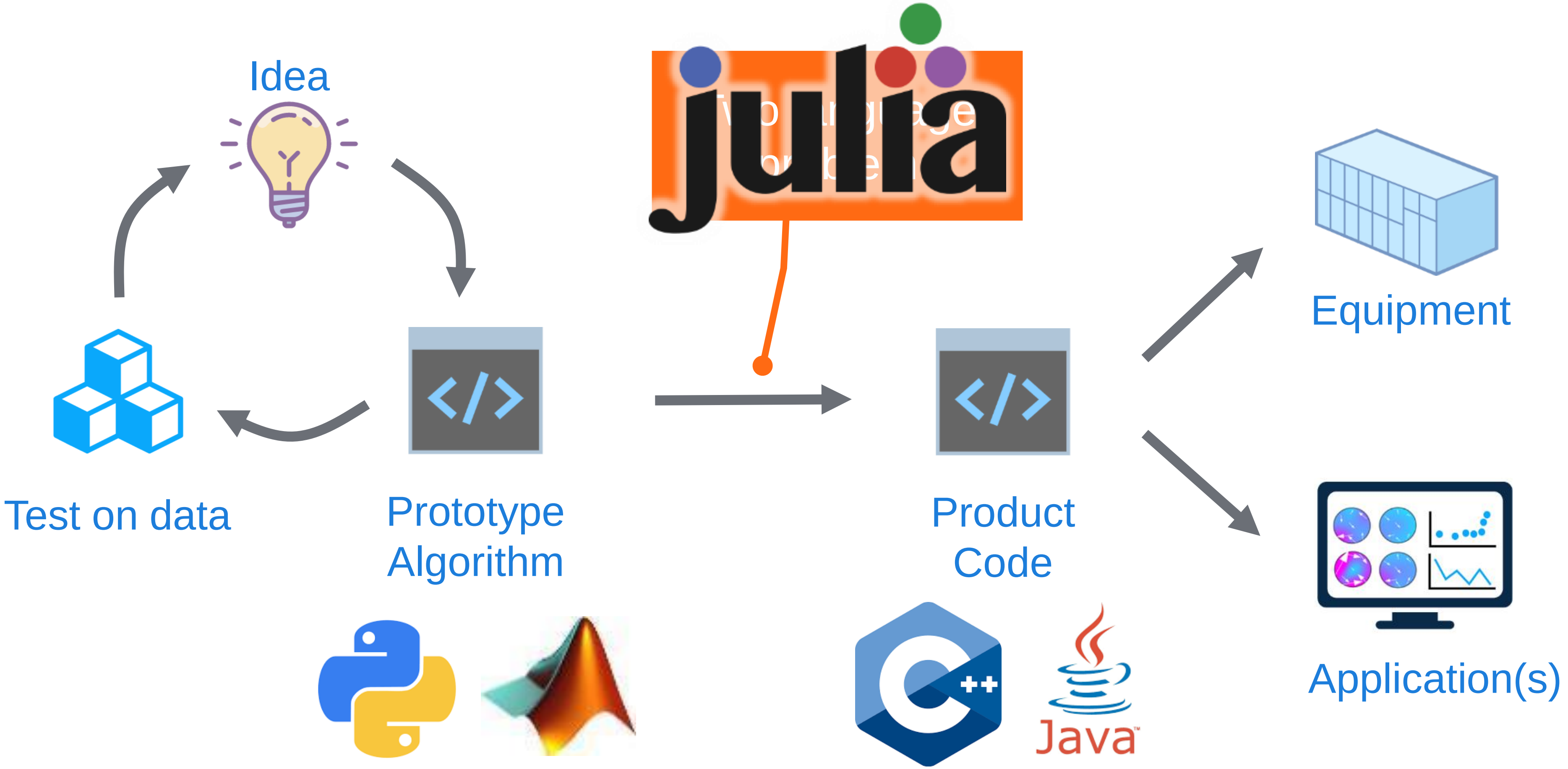
C memory alignment makes life easier

Open-source Julia contributions turn you into a legend ☺

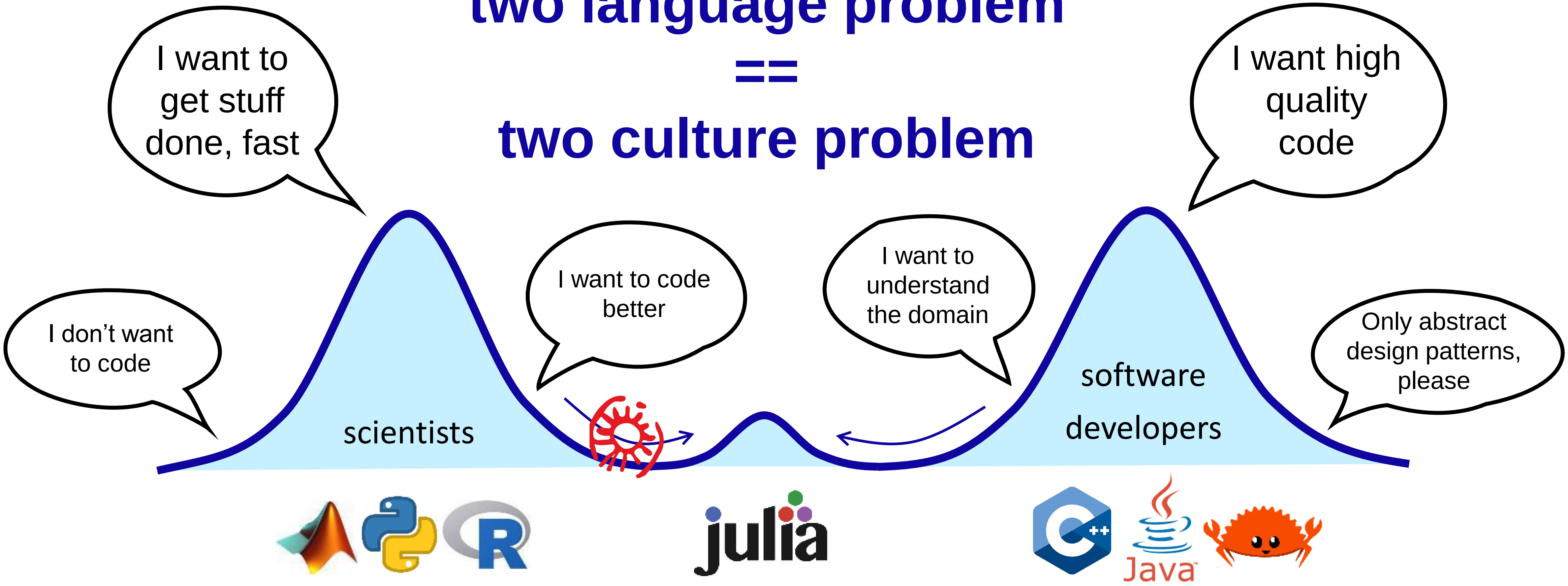
Julia’s **deployment options** are rapidly improving (with ASML funding)

But...

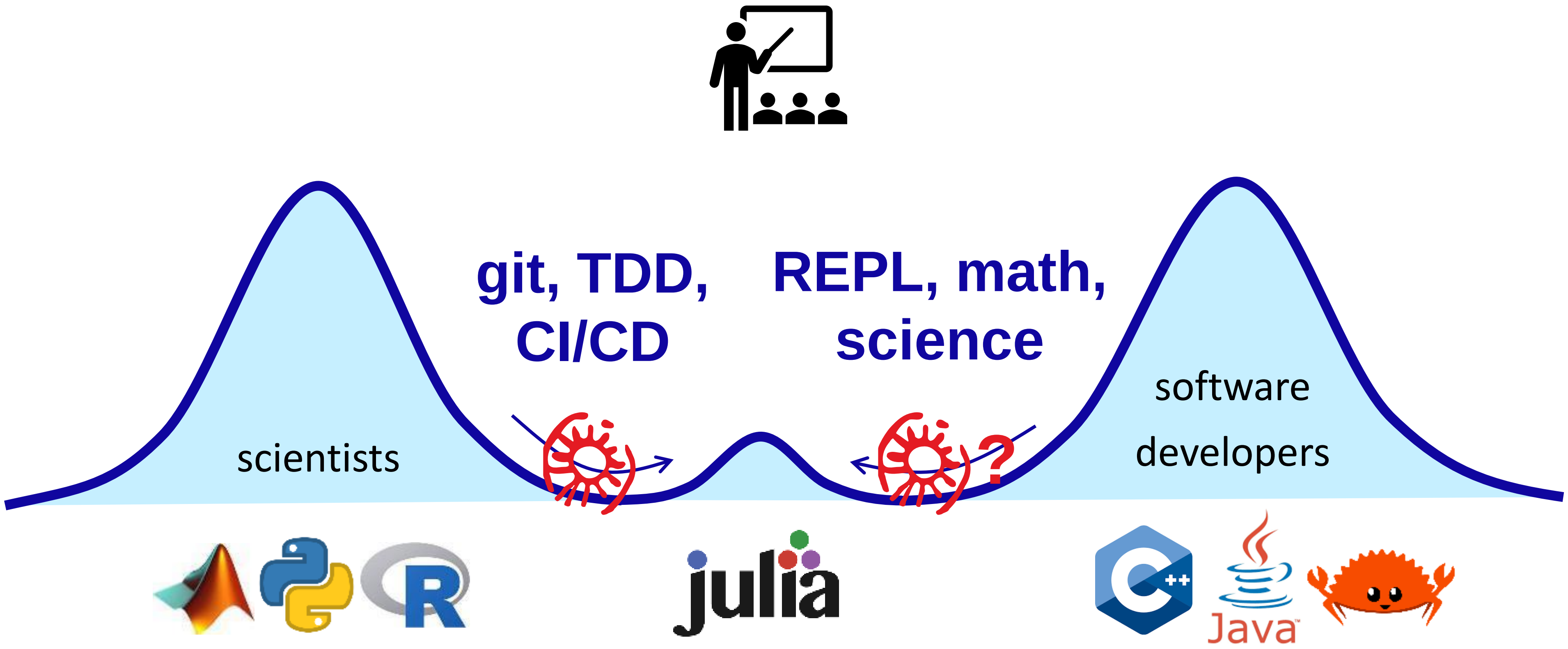
Remember: The Life of an Algorithm



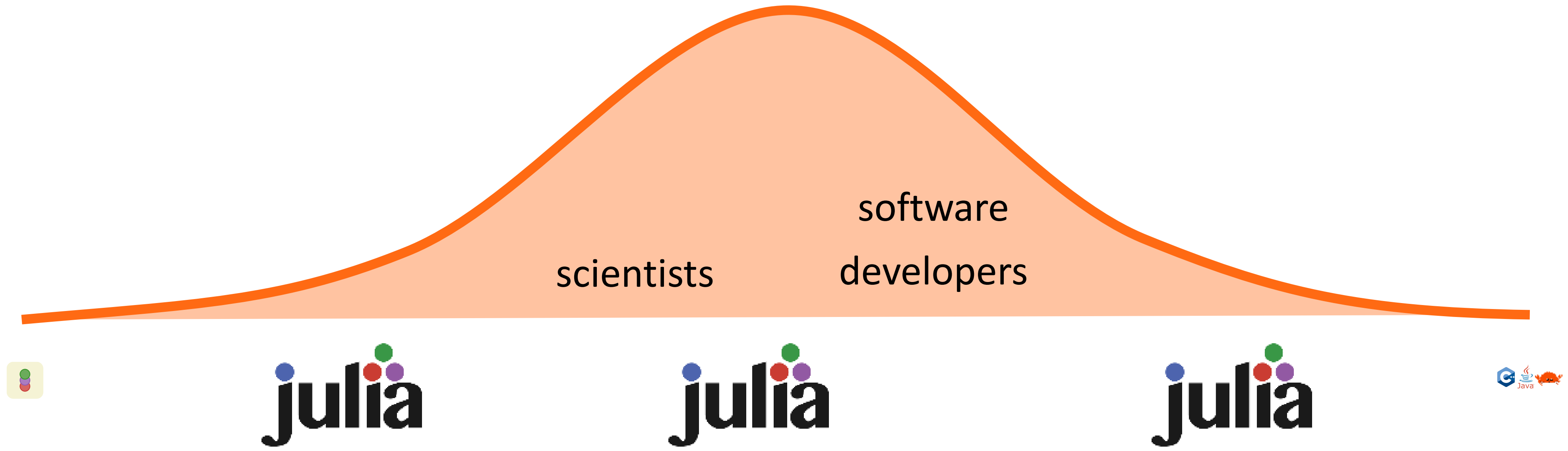
two language problem
==
two culture problem

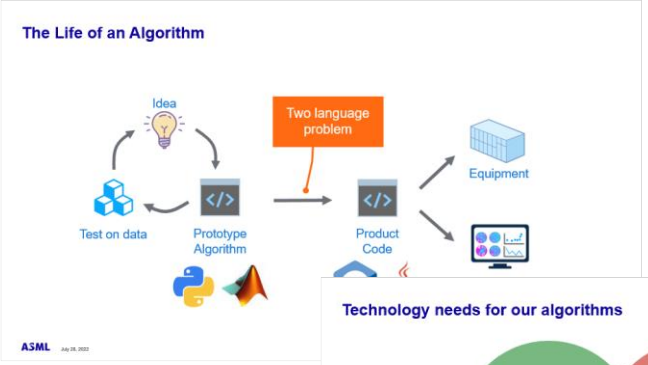


Teach more than Julia!



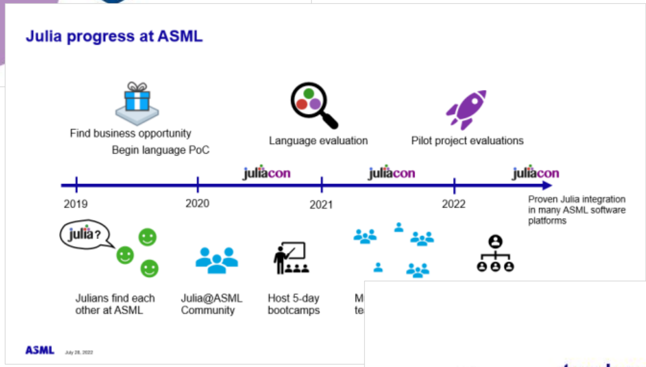
One big happy Gaussian!



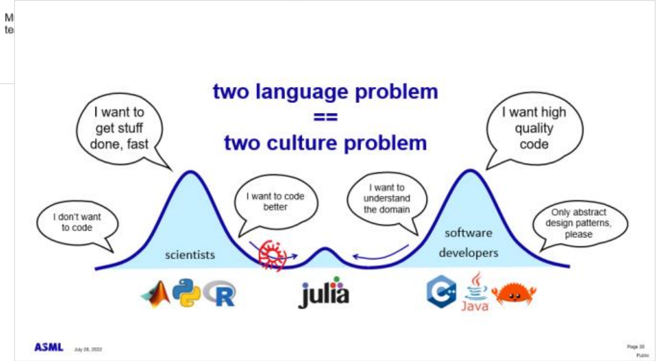


We had a two language problem in ASML

We found a technical solution



We built an internal Julia ecosystem



We are solving the two culture problem

We make it together

Functional and software engineers work together to build prototype algorithms

Excited customer

fast

Software and functional engineers can quickly deploy high quality algorithms

Happy customer

Show your hands!

Is Julia

~ HOT ~

or

~ NOT ~


Julia, C++
or Python?