Swift

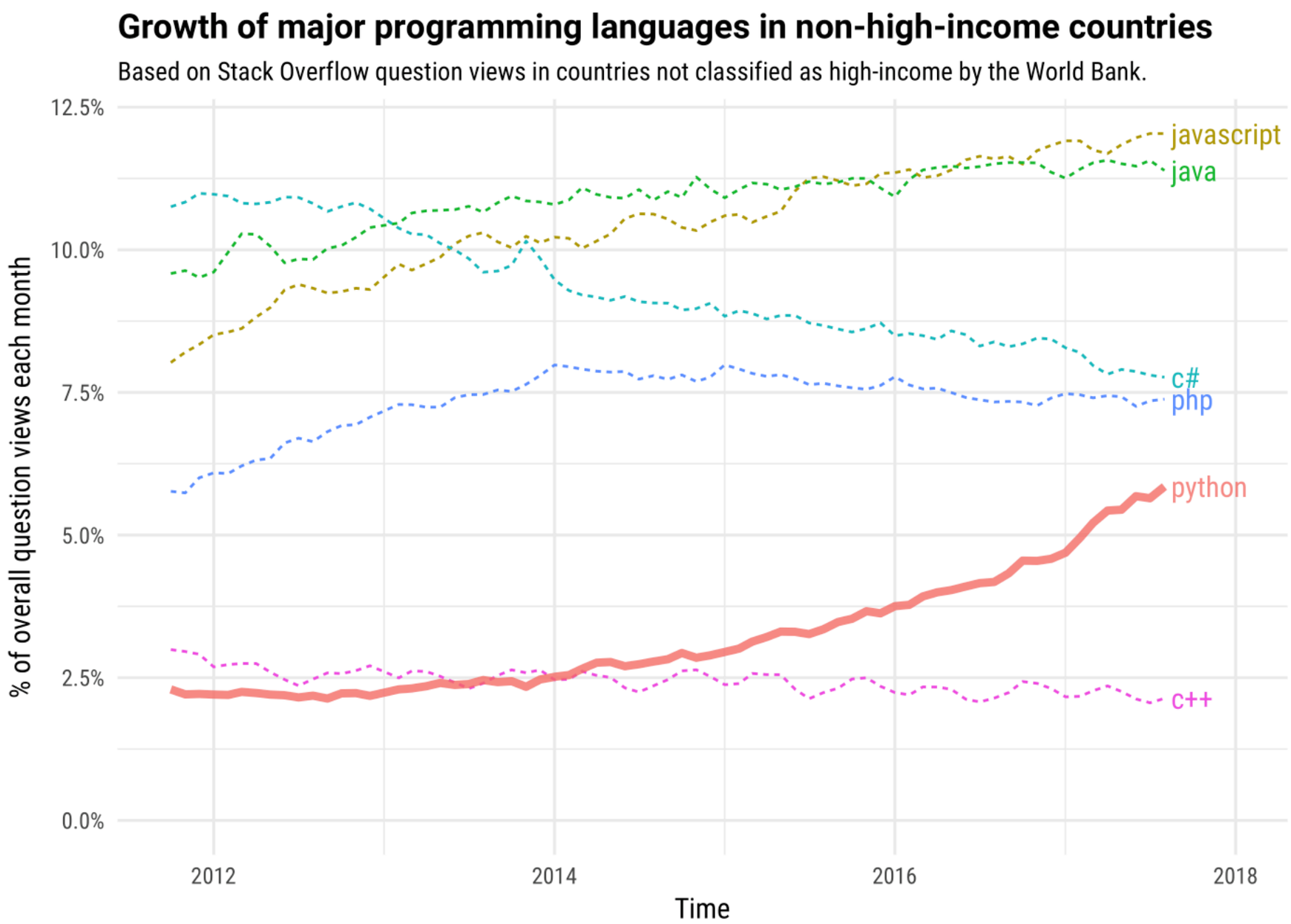
python™

SEBASTIAN BODENSTEIN

WILL JULIA OR SWIFT TAKE PYTHONS MACHINE LEARNING CROWN?

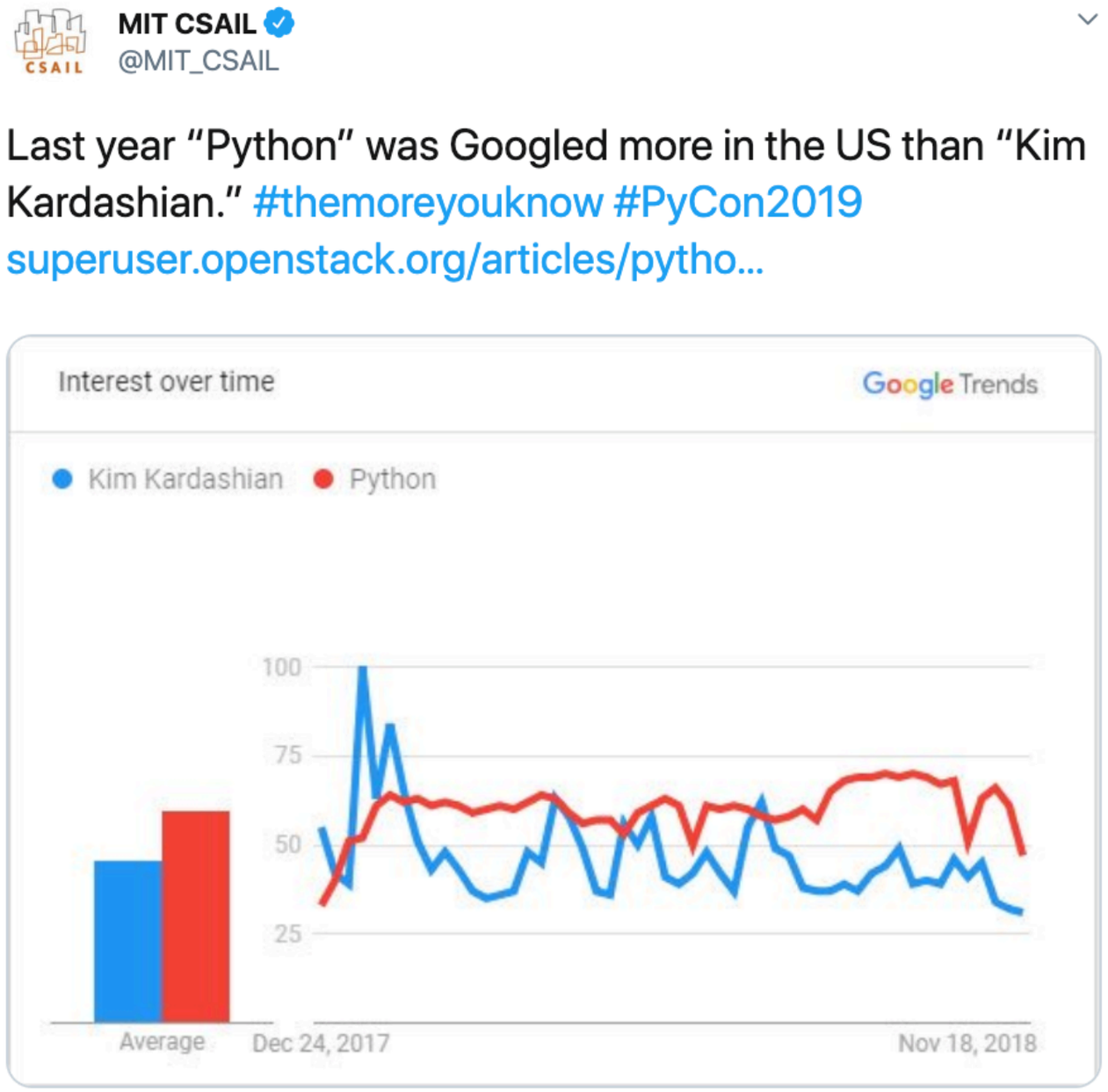
PART 1: PYTHON

PYTHON IS EATING THE WORLD

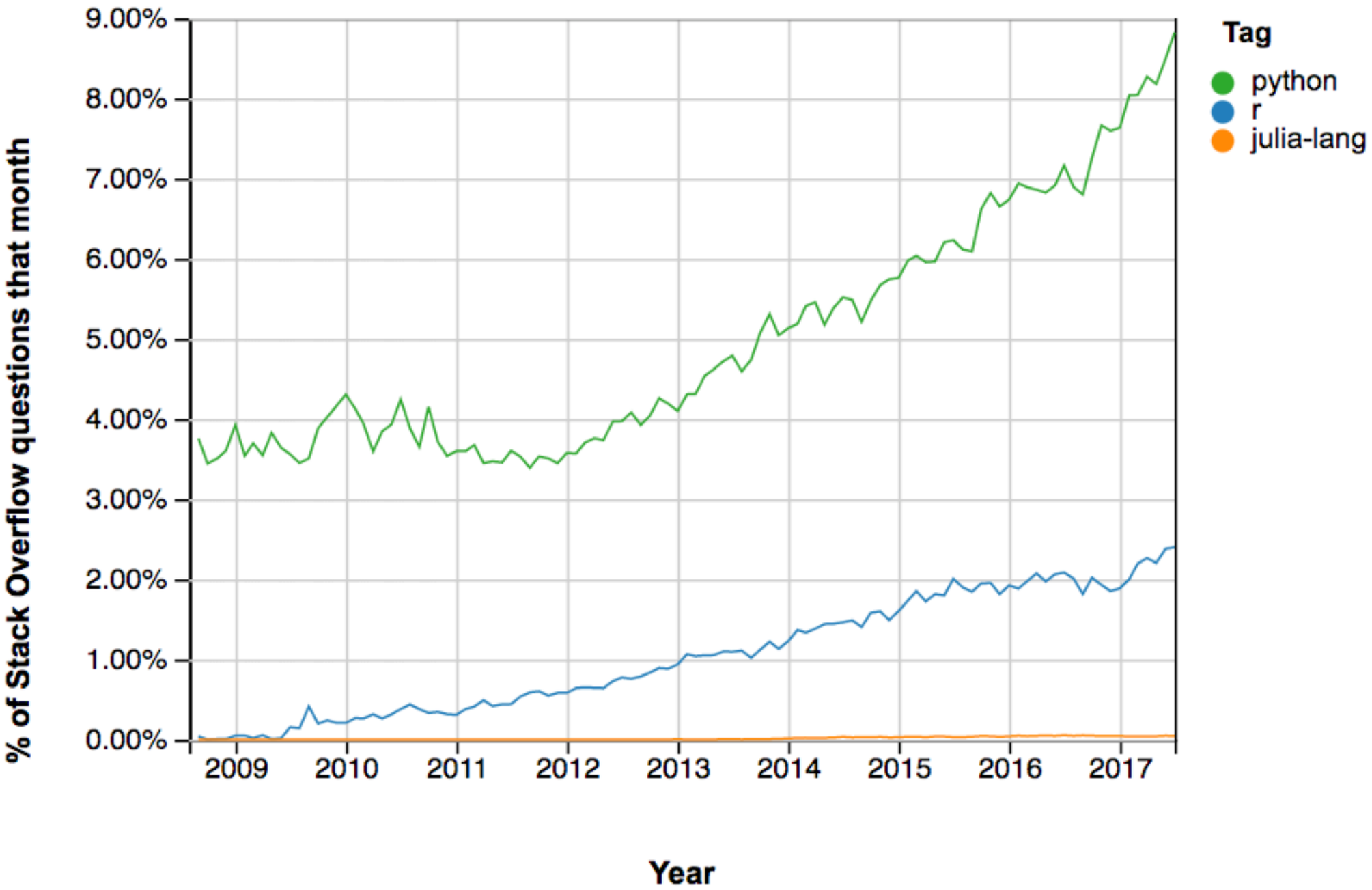


Taken from: https://stackoverflow.blog/2017/09/06/incredible-growth-python/

PYTHON IS EATING THE WORLD



ITS COMPETITORS



Taken from: https://stackoverflow.blog/2017/09/06/incredible-growth-python/

PYTHON PROBLEMS: EXECUTION SPEED

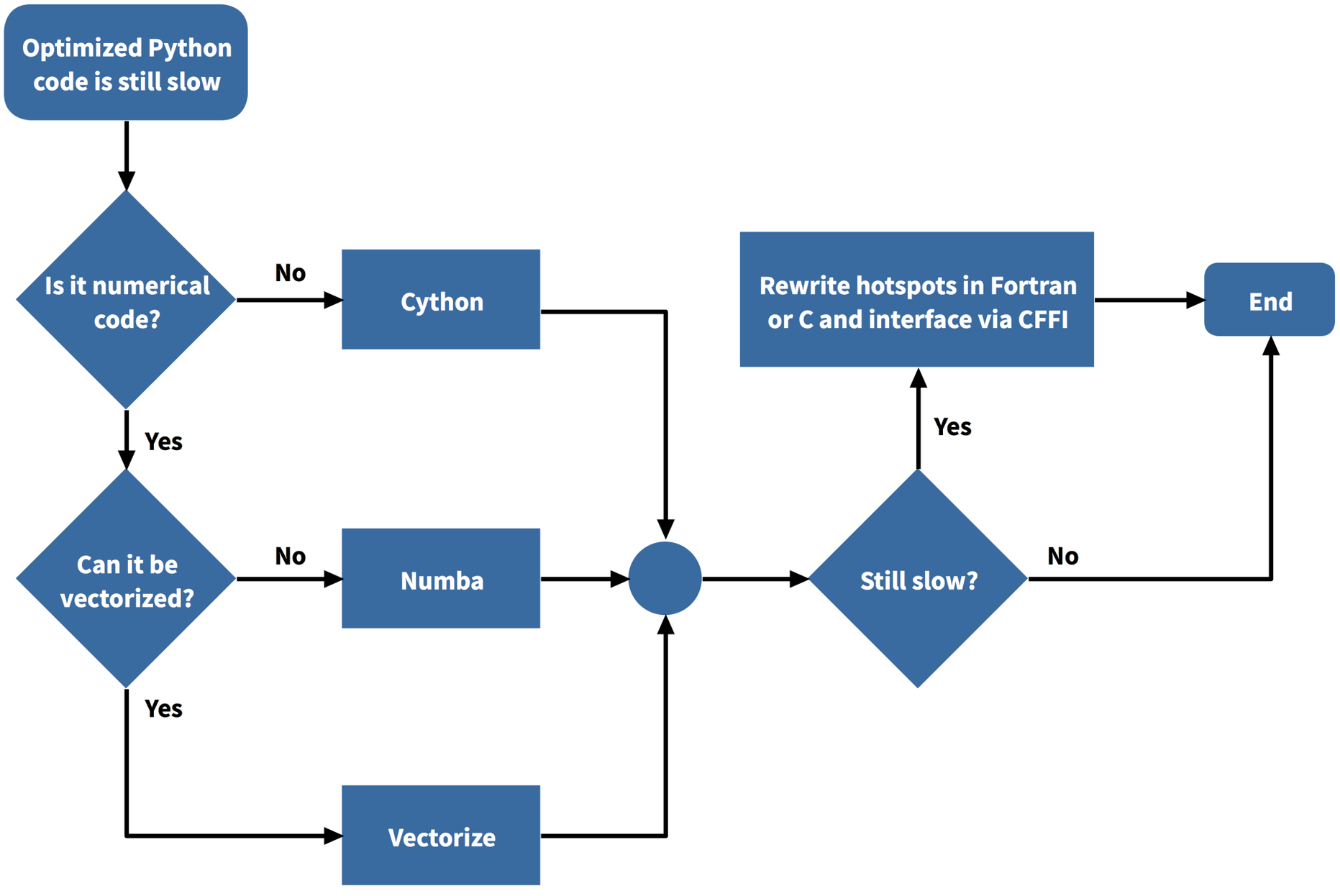
```python
# Python
def fib(n):
    if n<2:
        return n
    return fib(n-1)+fib(n-2)

# C
int fib(int n) {
    return n < 2 ? n : fib(n-1) + fib(n-2);
}

# Julia
fib(n) = n < 2 ? n : fib(n-1) + fib(n-2)
```

- **Question**: how much longer will it take Python to compute fib(20) than C?
  - ~100x

PYTHON PROBLEMS: EXECUTION SPEED

- Writing fast Python code effectively requires a second language:
  - C/C++ (eg. PyTorch and TensorFlow are mostly written in C++)
  - There is Numba, Pythran, etc: allow the compilation of a small subset of Python.
    - Need to learn what this subset is, and only use that!
- This is the infamous **two-language problem**
  - bad for developer productivity

PYTHON PROBLEMS: EXECUTION SPEED



PYTHON PROBLEMS: GIL

As a few of you might know, C Python has a Global Interpreter Lock (GIL)

```
>>> import that
The Unwritten Rules of Python

1. You do not talk about the GIL.
2. You do NOT talk about the GIL.
3. Don't even mention the GIL. No seriously.
...
```

Taken from: *Understanding the Python GIL*, David Beazley, PyCon 2010

PYTHON PROBLEMS: GIL

```
import threading
import time

def countdown(n):
    while n > 0:
        n -= 1

COUNT = 100000000 # 100 million

# This take ~5s
countdown(COUNT)

# Q: How long does this take?
t1 = threading.Thread(target=countdown,args=(COUNT//2,))
t2 = threading.Thread(target=countdown,args=(COUNT//2,))
t1.start(); t2.start()
t1.join(); t2.join()
```

PYTHON PROBLEMS: GIL

- **ANSWER**:
  - ~5s
- **Why**?
  - The GIL makes sure that only one thread runs in the interpreter at once
  - Simplifies low-level details, eg. memory management
- **Single-threaded Moores Law is dead**:
  - need parallelism to take advantage of all future hardware speedups

PART 2: JULIA

JULIA MISSION STATEMENT

"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...

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.

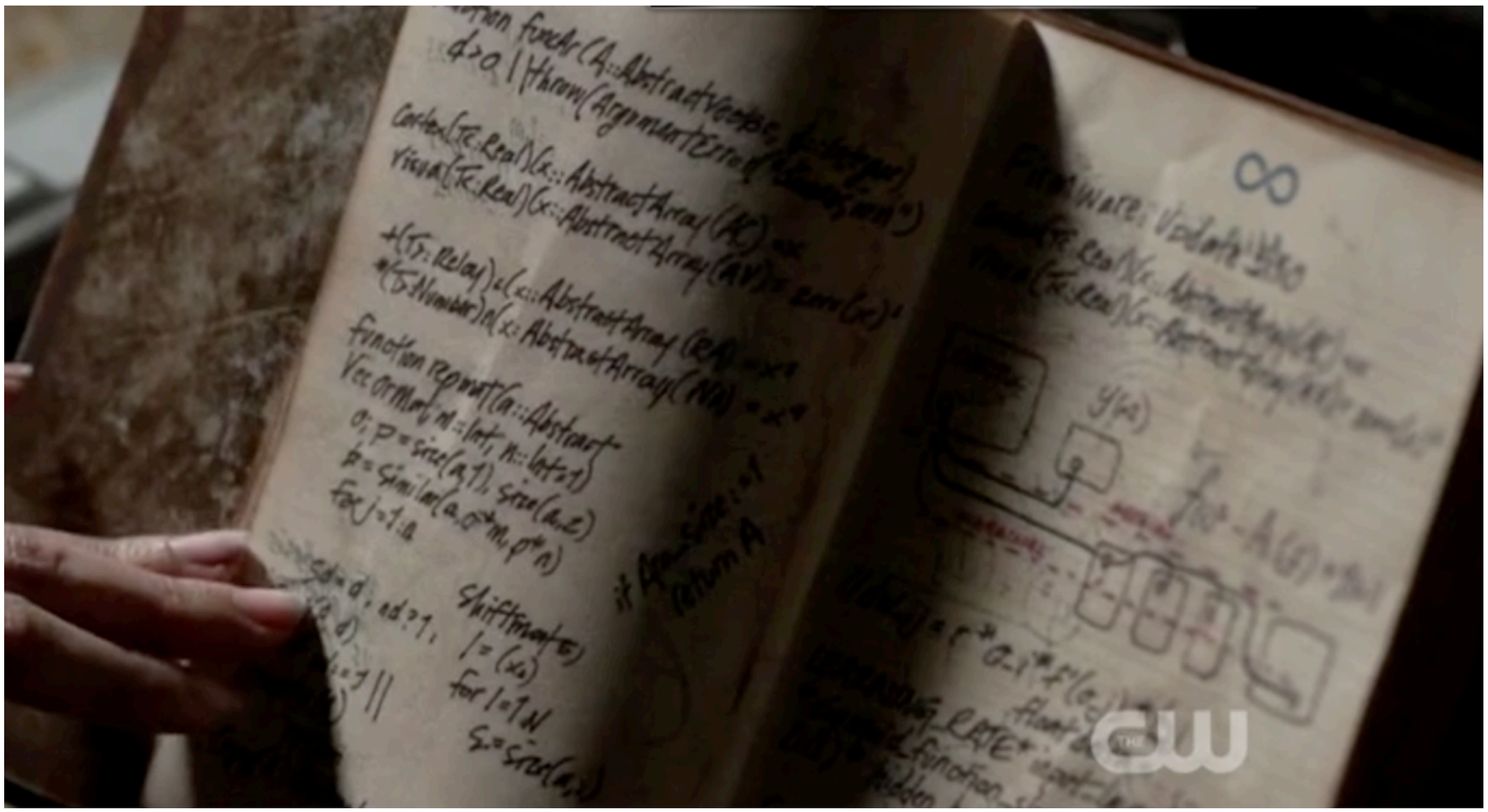
We want a language that's open source, with a liberal license. We want the speed of C with the dynamism of Ruby... We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab... Something 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?)" ~ **Why We Created Julia**, J Bezanson et al

JULIA MISSION STATEMENT

- Designed to solve the two-language problem
  - virtually all Julia packages are written in pure Julia!
- Release Dates:
  - **Python**: 1991
  - **Julia**: 2012
- Julia downloads to date:
  - 3.2 million

THE 100



The 100 is a SciFi set in 150 years time. The source code of one of the AIs was Julia! https://juliacomputing.com/communication/2017/09/19/julia-the-ai-language-for-next-150-years.html

DEMO

ADVANTAGES

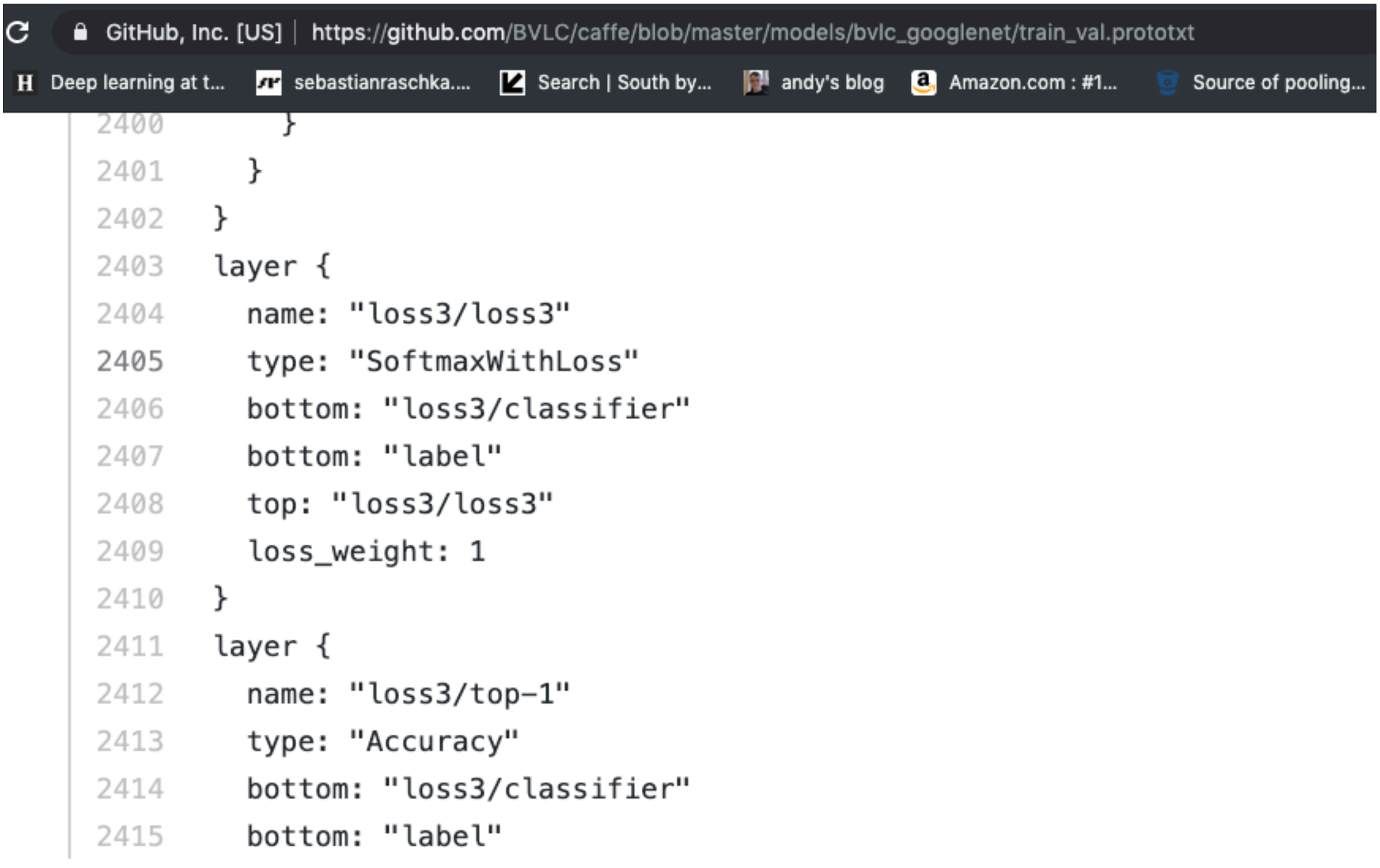
- The compilability of Julia mainly benefits package developers:
  - Eg. the Julia numerical differential equations package is almost certainly the best of any language (including Matlab, Python, Mathematica)
    - http://docs.juliadiffeq.org/latest/
  - In other languages, need expert C/C++/Fortran programmers to write performance critical parts of ODE solvers.
  - Ugly situation with callbacks: sure, your fast code is C/C++/Fortran. But for ODEs, you want custom Python code for computing Jacobians, logging, etc. This interacts terribly with C/C++/Fortran
  - A great post on this:
    - http://www.stochasticlifestyle.com/why-numba-and-cython-are-not-substitutes-for-julia/

USED BY CELESTE PROJECT

- "Celeste had to be fast, so we considered C++, a blend of Python and Cython, and Julia. Julia let us write most of our program in a high-level, math-inspired syntax, without requiring us to pass data structures between programming languages," says Jeffrey Regier (UC Berkeley Statistics), lead author on the paper presenting the method.

PART 3: DIFFERENTIABLE PROGRAMMING

THE OLD DAYS (2013): CAFFE



TODAY: PYTORCH, TF 2.0

- Move towards differentiating through programs
  - still hits the Python interpreter after every operation

```
x = torch.randn(3, requires_grad=True)

y = x * 2
while y.data.norm() < 1000:
    y = y * 2

print(y)
```

THE FUTURE: JULIA + SWIFT

- Build automatic differentiation into the language at the compiler level
  - differentiate any program!

THE FUTURE: JULIA + SWIFT

The really powerful advance is this: pervasive differentiability means all these techniques snap together like lego bricks. Rather than always writing new programs for ML, we can incorporate existing ones, enabling physics engines inside deep learning-based robotics models. Where current reinforcement learning algorithms need to build a detailed model of the external world from only a coarse-grained reward signal (which sounds like a brute force problem if anything does), we can instead just drop in detailed, precise knowledge of physical systems before training even begins.

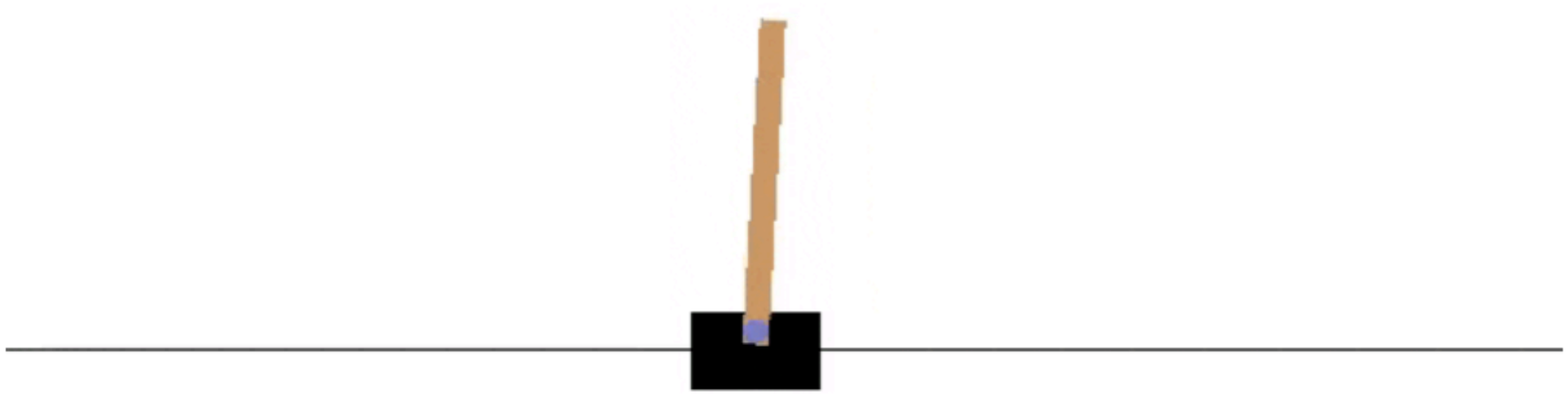
project. But advances in language and compiler technology, especially automatic differentiation, are bringing us closer to the holy grail: "just differentiate my game engine, please."

From: https://fluxml.ai/2019/02/07/what-is-differentiable-programming.html

THE FUTURE: JULIA + SWIFT

- Making CartPole environment
  differentiable makes training vastly faster

The results speak for themselves. Where RL methods need to train for hundreds of episodes before solving the problem, the DP model only needs around 5 episodes to win conclusively.



From: https://fluxml.ai/2019/03/05/dp-vs-rl.html

THE FUTURE: JULIA ZYGOTE

Zygote

Welcome! Zygote extends the Julia language to support differentiable programming. With Zygote you can write down any Julia code you feel like – including using existing Julia packages – then get gradients and optimise your program. Deep learning, ML and probabilistic programming are all different kinds of differentiable programming that you can do with Zygote.

At least, that's the idea. We're still in beta so expect some adventures.

Taken from: https://fluxml.ai/Zygote.jl/latest/

TODAY: SWIFT FOR TENSORFLOW

- **Swift**: released 2014
- Similar to Julia: compiles to LLVM
- Designed by Chris Lattner
  - the main author of LLVM!
  - he now heads the Swift for TensorFlow project at Google

TODAY: SWIFT FOR TENSORFLOW

- **Swift for TensorFlow**
  - build automatic differentiation into the Swift compiler
  - "Differentiable programming gets first-class support in a general-purpose programming language. Take derivatives of any function, or make custom data structures differentiable at your fingertips."
- Google calls it "a next generation platform for deep learning and differentiable programming". Possibly a successor to TensorFlow v2?
- https://www.tensorflow.org/swift
- Excellent motivation document: https://github.com/tensorflow/swift/blob/master/docs/WhySwiftForTensorFlow.md

TODAY: SWIFT FOR TENSORFLOW

fast.ai Embracing Swift for Deep Learning

Written: 06 Mar 2019 by *Jeremy Howard*

> *Note from Jeremy*: If you want to join the next deep learning course at the University of San Francisco, discussed below, please apply as soon as possible because it's under 2 weeks away! You can apply here. At least a year of coding experience, and deep learning experience equivalent to completing Practical Deep Learning for Coders is required.

Today at the TensorFlow Dev Summit we announced that two lessons in our next course will cover Swift for TensorFlow. These lessons will be co-taught with the inventor of Swift, Chris Lattner; together, we'll show in the class how to take the first steps towards implementing an equivalent of the fastai library in Swift for TensorFlow. We'll be showing how to get started programming in Swift, and explain how to use and extend Swift for TensorFlow.

CASE STUDY: NEURAL ODE

- Won best paper award at NeurIPS 2018

Neural Ordinary Differential Equations

Ricky T. Q. Chen*, Yulia Rubanova*, Jesse Bettencourt*, David Duvenaud
University of Toronto, Vector Institute
{rtqichen, rubanova, jessebett, duvenaud}@cs.toronto.edu

CASE STUDY: NEURAL ODE

- Very simple idea: suppose you have an ODE

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$$

- The Euler discretization is:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t)$$

- Which is a ResNet! So ODE is result of making number ResNet layers continuous

CASE STUDY: NEURAL ODE

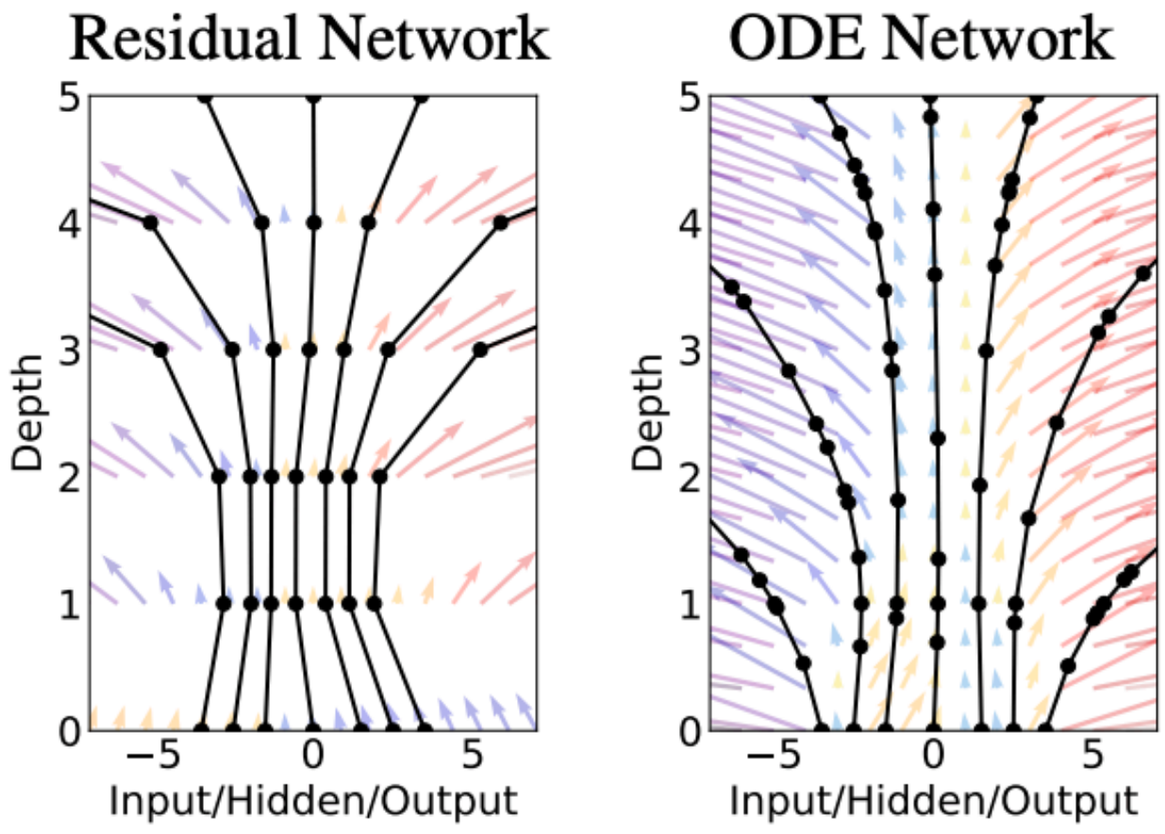


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

CASE STUDY: NEURAL ODE

- A neat property for generative modelling:

The discretized equation (1) also appears in normalizing flows (Rezende and Mohamed, 2015) and the NICE framework (Dinh et al., 2014). These methods use the change of variables theorem to compute exact changes in probability if samples are transformed through a bijective function f:

$$\mathbf{z}_1 = f(\mathbf{z}_0) \implies \log p(\mathbf{z}_1) = \log p(\mathbf{z}_0) - \log \left| \det \frac{\partial f}{\partial \mathbf{z}_0} \right| \tag{6}$$

Surprisingly, moving from a discrete set of layers to a continuous transformation simplifies the computation of the change in normalizing constant:

Theorem 1 (Instantaneous Change of Variables)**.** *Let $\mathbf{z}(t)$ be a finite continuous random variable with probability $p(\mathbf{z}(t))$ dependent on time. Let $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$ be a differential equation describing a continuous-in-time transformation of $\mathbf{z}(t)$. Assuming that f is uniformly Lipschitz continuous in $\mathbf{z}$ and continuous in t, then the change in log probability also follows a differential equation,*

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathrm{tr}\left(\frac{df}{d\mathbf{z}(t)}\right) \tag{8}$$

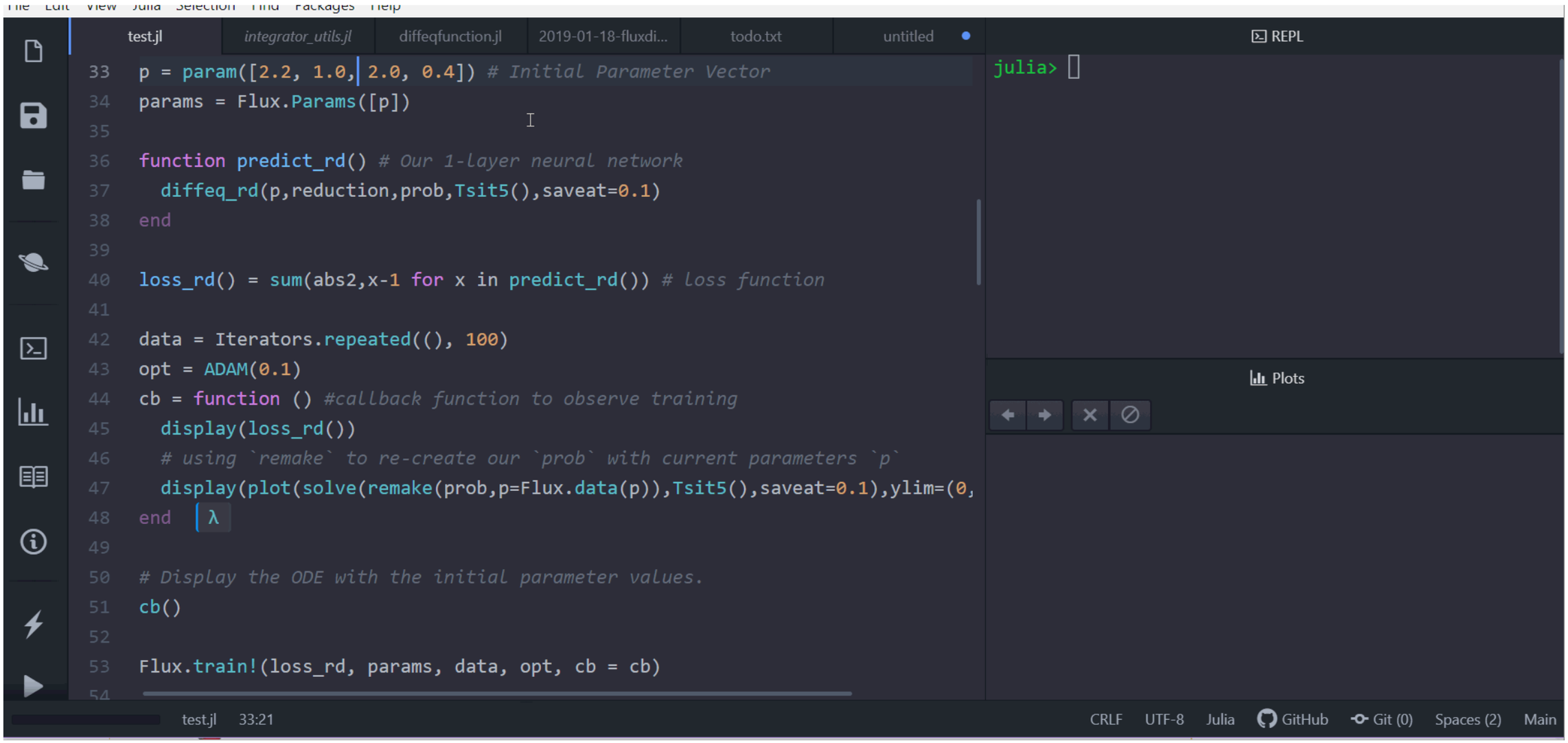
CASE STUDY: NEURAL ODE

The core technical challenge: backpropagation through differential equation solvers

Let's end by explaining the technical issue that needed a solution to make this all possible. The core to any neural network framework is the ability to backpropagate derivatives in order to calculate the gradient of the loss function with respect to the network's parameters. Thus if we stick an ODE solver as a layer in a neural network, we need to backpropagate through it.

- Impossible to do in Python without reimplementing ODE solvers
  - Very easy in Julia: without needing to rewrite anything, can immediately differentiate through any of ODE solvers in DifferentialEquations.jl!

CASE STUDY: NEURAL ODE



Taken from: https://julialang.org/blog/2019/01/fluxdiffeq

PART 4: IS PYTHON DOOMED?

ECOSYSTEM

- Pythons ecosystem is vastly superior to Julia and Swift
- Lack of an ecosystem doomed Lua Torch before...

BUT: JULIA AND SWIFT HAVE GREAT PYTHON SUPPORT

Seamless Python Interoperability

```
[4]  import TensorFlow
     import Python
     %include "EnableIPythonDisplay.swift"
```

```
let plt = Python.import("matplotlib.pyplot")
let np = Python.import("numpy")

IPythonDisplay.shell.enable_matplotlib("inline")

let x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```

Swift for TensorFlow

IT DEPENDS

Soumith Chintala @soumithchintala · Apr 7

A lot of folks ask me: what's the next mainstream ML software? What is PyTorch-Next?
There's no magic answer, it depends on where the field goes. Software, Research and Hardware go hand-in-hand -- iteratively doing exploration and exploitation.
Predictions as of today:
(1/4)

9 97 384

Soumith Chintala @soumithchintala · Apr 7

- If we go big on GPs and PGMs, then I expect a mainstream Pyro / Edward.
- If we go back to 2nd order methods, something like Jax.
- If Graph ConvNets, then Julia -- for it's ability to build efficient fundamental data structures in an interactive language.
(2/4)

6 33 202

IT DEPENDS

Soumith Chintala @soumithchintala · Feb 19

Julia, Swift are great viable options.
Or, one can make Python cool enough ;-)
With MyPy-style static checking, torch.jit style compilation, the benefits might be realized while staying in Python.

7 4 139

Yann LeCun
@ylecun

Replying to @soumithchintala and @jeremyphoward

Compiling Python (or a subset of it) is one (Lush-like) way to do it.
Julia and Swift are nice.
another option, with the advantage of safe
lism

Facebook's chief AI scientist: Deep learning may need a new programming l...
Deep learning may need a new programming language that's more flexible and easier to work with than Python, Facebook AI Research director Yann ...
venturebeat.com

skip · A programming language to skip the things you have alread...
A programming language to skip the things you have already computed
skiplang.com

Feb 19, 2019 · Twitter for Android

CONCLUSION + QUESTIONS