JULIA

A programming language *specifically* made for scientists

Carsten Bauer @ University of Cologne, October 2018

DISCLAIMER

- This workshop will be opinionated!
- There a many good programming languages out there!
- This workshop won't be about comparing those in detail!
- Find out what works best for your use case!

WHAT DOES SCIENCE NEED FROM A PROGRAMMING LANGUAGE?

TYPICAL WORKFLOW

1. How can I answer the question using computers?
2. Implement those ideas
3. Run the code
4. Analyze and visualize results
5. Realize that your results are full of nonsense
6. Debug
7. Run the code again

WHAT WE WANT

- Great performance of "doing research"
- Great performance of computation
- Source code that is concise and easy to read & understand
- Code that is easy to install, share, use, and replicate
- Free and open source

COMPILED

- `C`, `Java`, `Fortran`, `Go`, `Haskell`, `Rust`
- **Static**: you can't change anything once compiled
- **Fast**: many things are known about objects
- Industry standard
- **Critique**: too low-level, no interactivity

INTERPRETED

- `MATLAB`, `Python`, `R`, `Mathematica`, `JavaScript`
- **Dynamic**: write → (interpretation-layers) → run
- **Slow**: Few is known about objects (hard to optimize)
- Easy to code and interactive
- **Critique**: too slow, black box

TWO-LANGUAGE PROBLEM

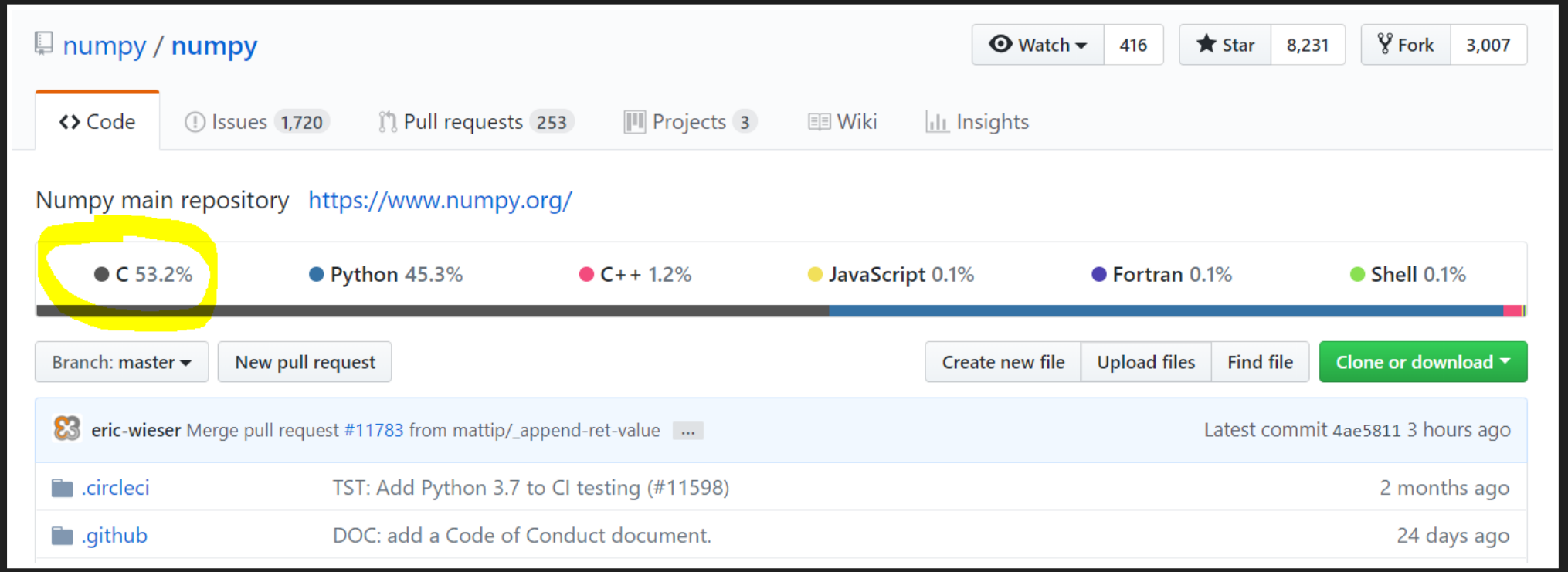
WRITING SCIENTIFIC CODE

1. Brain-storm and prototype in an interpreted language for exploration and testing.
2. Deliver a performant final-version in a compiled language.

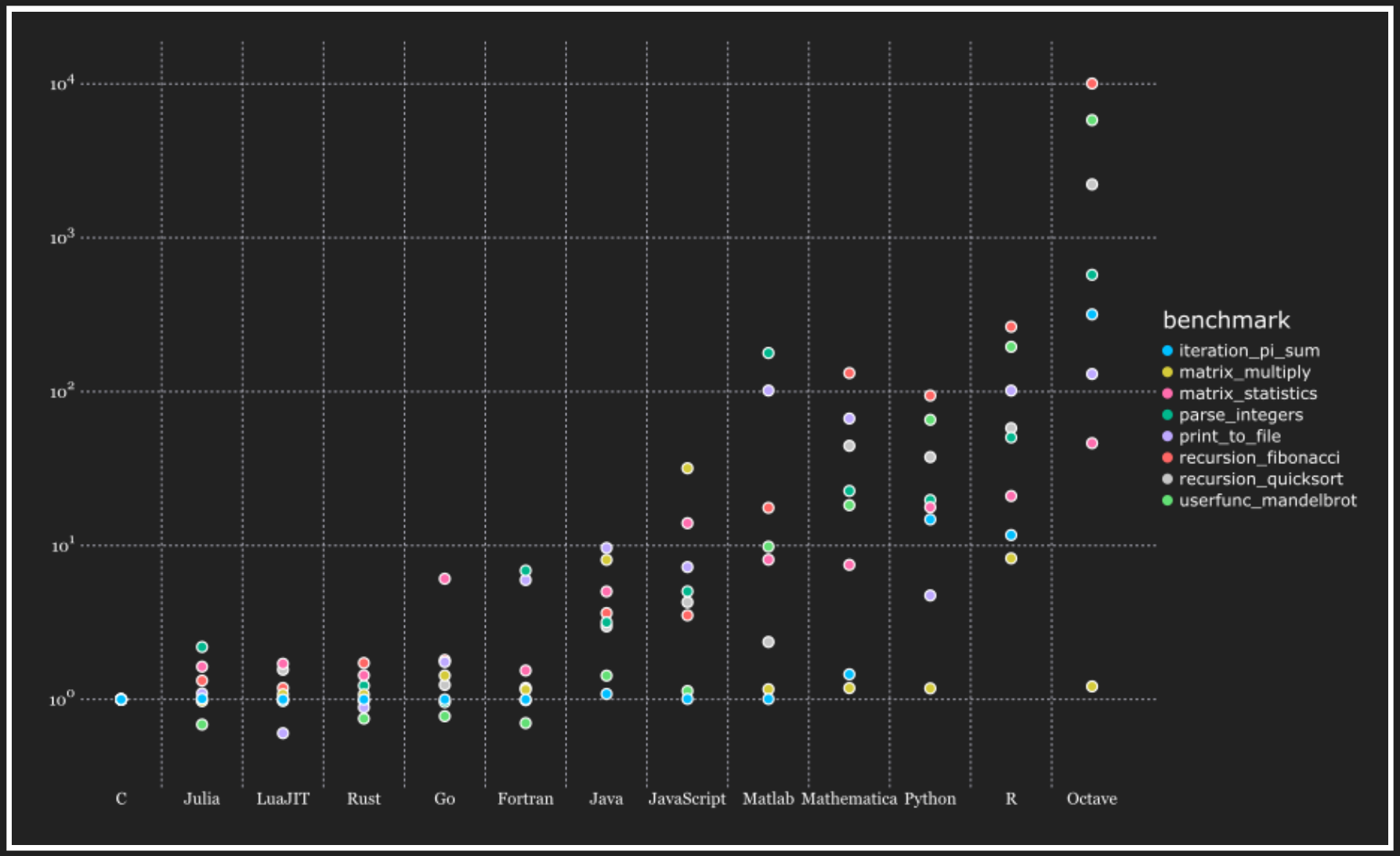
BLACK BOXES IN INTERPRETED LANGUAGES

```
Does a package/library do exactly
what you need?
              ├── Yes:   Great!
              ├── Dunno: Damn it, it's written in C/Fortran.
              └── No:    You need to (learn to) code in C/Fortran.
```

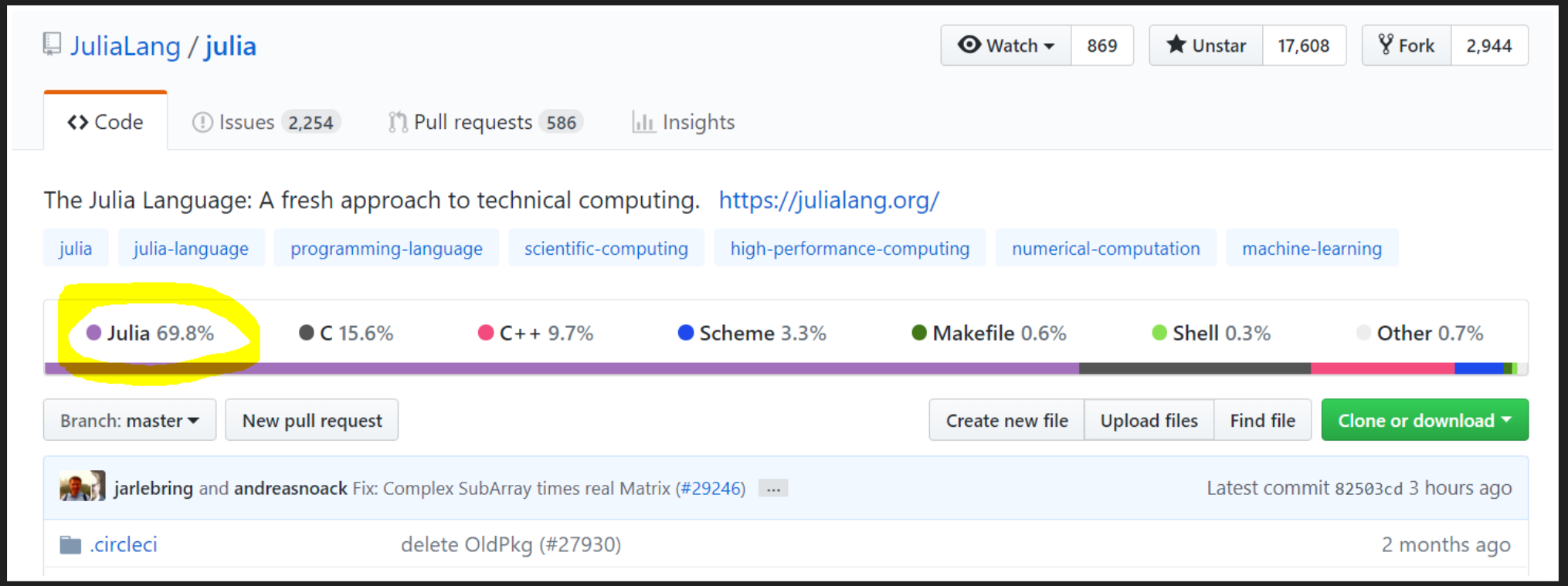
NUMPY CODEBASE



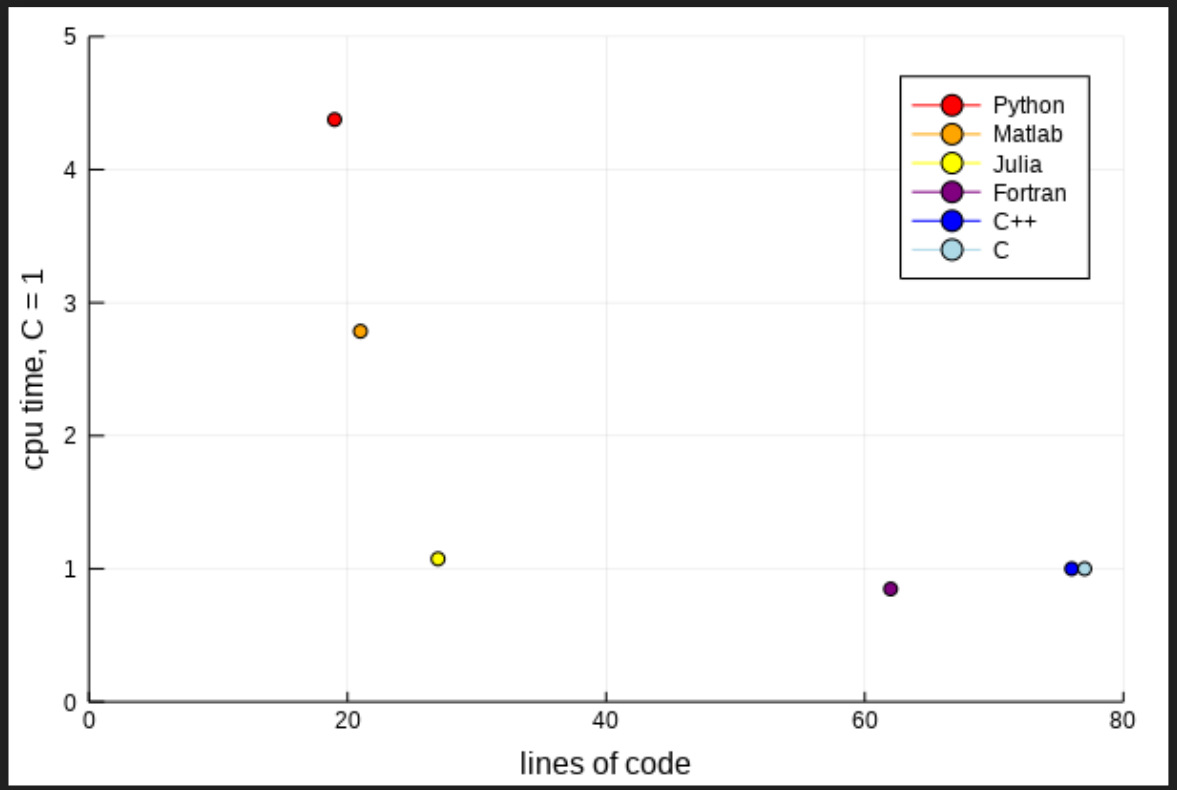
WHAT IF THE INTERPRETED LANGUAGE IS FAST ENOUGH?



JULIA CODEBASE



EXPRESSIVE AND FAST



THE JULIA PROJECT

- Begun in 2009 as part of Jeff Bezanson's PhD thesis @ MIT.
- First major release of Julia 1.0 in August 2018.
  - already > 2 million downloads
- ~750 core language contributors
- 1600+ external packages

COMMUNITY

- Developed by scientists
- Very inviting and excellent
- Many experts that used to code in C/Fortran

DYNAMIC & FAST, HOW?

1. Just-in-time compilation (JIT): User-level code is compiled to machine code on-the-fly.
2. Meticulous type system: Designed to maximize impact of JIT.
3. Multiple dispatch: Function dispatch determined at compile time when possible, run time when not.

SYNTAX CLARITY

Looks like `Python`/`MATLAB`/`R` but with prettier syntax.

UNICODE

Math:

```
2π√3/5α₀
```

Python:

```
2*np.pi*np.sqrt(3)/(5*alpha0)
```

Julia:

```
2π*√3/5α₀
```

CUSTOM INFIX OPERATORS

```
# rotate coordinate `c` by `θ` radians
julia> ↺(c, θ) = [cos(θ) -sin(θ); sin(θ) cos(θ)]*c
↺ (generic function with 1 method)

julia> [1,0] ↺ π/2
2-element Array{Float64,1}:
 0.0
 1.0
```

EXAMPLE: TENSOR PRODUCT

```
⊗ # Tensor product operator (\otimes)
ψx = gaussianstate(b_position, x0, p0_x, σ)
ψy = gaussianstate(b_position, y0, p0_y, σ)
ψ = ψx ⊗ ψy
```

UNITS

It took a beetle 36 seconds to walk 25 cm. How many days will it take it to walk 3 km?

```
v = 25cm / 36s
t = 3km / v
uconvert(d, t)
```

5 days

<!--- ---

MISSING

The concept of `missing` and `NaN` is treated correctly:

```
julia> NaN + 1          = NaN

julia> missing + 1      = missing
```

```
julia> true | missing  = true

julia> false | missing = missing

julia> true & missing  = missing

julia> false & missing = false
```

INTERFACING

Call C and Fortran libraries with **zero overhead**!

Full access to libraries and functionalities you already know.

`PyCall.jl`, `Cxx.jl`, `RCall.jl`, `Mathematica.jl`, ...

PyCall.jl

```
using PyCall
@pyimport numpy as np
x = rand(2,2)                    # Note, that this is a Julia array!
result = np.linalg.eigs(x)
```

FREE & OPEN SOURCE

- Enables highly specialized and niche solutions and tools
- No "black boxes", everything is within reach
- Language design decision are transparent and democratic

DISADVANTAGES

It's still relatively young…

TEETHING PROBLEMS

- Ecosystem (e.g. some packages, IDEs, debugger) not as mature as in other environments.
- Many pure Julia implementations missing
- Harder to find answers online.

STILL WORKING OUT ALL THE KINKS

- Loading some packages is still a bit slow.
- Plotting works but hasn't settled yet.
- Recent release of `v1.0.0`: many minor inconveniences.

JULIA IS GREAT

- People come to `Julia` because of its speed, but stay for the type-dispatch system
- A number of libraries already far out-perform their equivalents in other languages.
- Most compiler optimizations are yet to come!

HIGHLY RECOMMENDED TALKS

- Nick Eubank: What Julia Offers Academic Researchers
- George Datseris: Why Julia is the most suitable language for science


julia