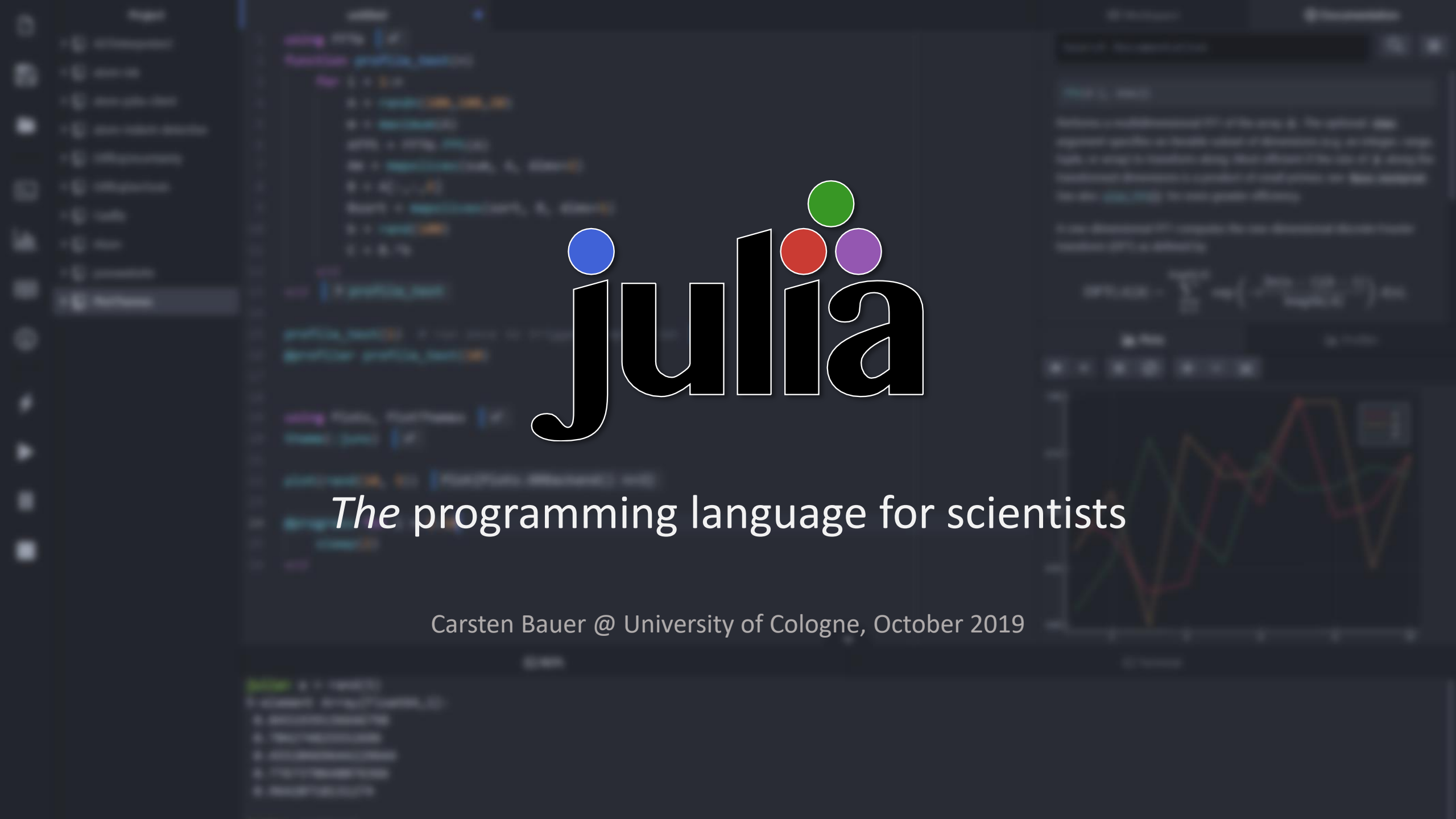


The programming language for scientists

Carsten Bauer @ University of Cologne, October 2019

https://science.raphael.poss.name/programming-levels.html

How good are you at programming?
A CEFR-like approach to measure programming proficiency

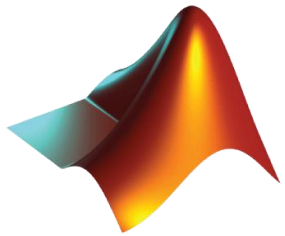
Raphael 'kena' Poss

Programming skills self-assessment matrix

		A1 Basic User	A2 Basic User	B1 Intermediate User	B2 Intermediate User	C1 Proficient User	C2 Proficient User
Writing	Writing code	I can produce a correct implementation for a simple function, given a well-defined specification of desired behavior and interface, without help from others.	I can determine a suitable interface and produce a correct implementation, given a loose specification for a simple function, without help from others. I can break down a complex function specification in smaller functions.	I can estimate the space and time costs of my code during execution. I can empirically compare different implementations of the same function specification using well-defined metrics, including execution time and memory footprint. I express invariants in my code using preconditions, assertions and post-conditions. I use stubs to gain flexibility on implementation order.	I use typing and interfaces deliberately and productively to structure and plan ahead my coding activity. I can design and implement entire programs myself given well-defined specifications on external input and output. I systematically attempt to generalize functions to increase their reusability.	I can systematically recognize inconsistent or conflicting requirements in specifications. I can break down a complex program architecture in smaller components that can be implemented separately, including by other people. I can us existing (E)DSLs or metaprogramming patterns to increase my productivity.	I can reliably recognize when under-specification is intentional or not. I can exploit under-specification to increase my productivity in non-trivial ways. I can devise new (E)DSLs or create new metaprogramming patterns to increase my productivity and that of other programmers.
	Refactoring	I can adapt my code when I receive small changes in its specification without rewriting it entirely, provided I know the change is incremental. I can change my own code given detailed instructions from a more experienced programmer.	I can determine myself whether a small change in specification is incremental or requires a large refactoring. I can change my own code given loose instructions from a more experienced programmer.	I can derive a refactoring strategy on my own code, given relatively small changes in specifications. I can change other people's code given precise instructions from a person already familiar with the code.	I can predict accurately the effort needed to adapt my own code base to a new specification. I can follow an existing refactoring strategy on someone else's code. I can take full responsibility for the integration of someone else's patch onto my own code.	I can reverse-engineer someone else's code base with help from the original specification, and predict accurately the effort needed to adapt it to a new specification.	I can reverse-engineer someone else's code base without original specification, and predict accurately the effort needed to adapt it to a new specification.

There’s a plethora of programming languages

There’s a plethora of programming languages

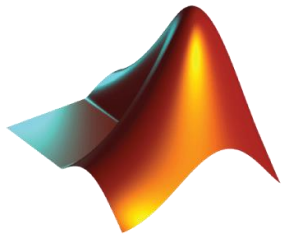


MATLAB

There’s a plethora of programming languages

Fast

Slow(ish)



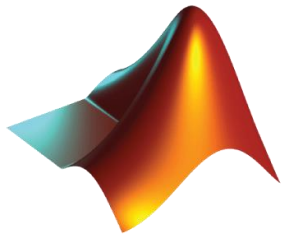
MATLAB

There’s a plethora of programming languages

Fortran

Static

Dynamic

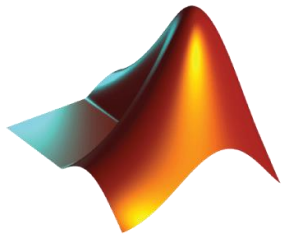


MATLAB

There’s a plethora of programming languages

Compiled

Interpreted

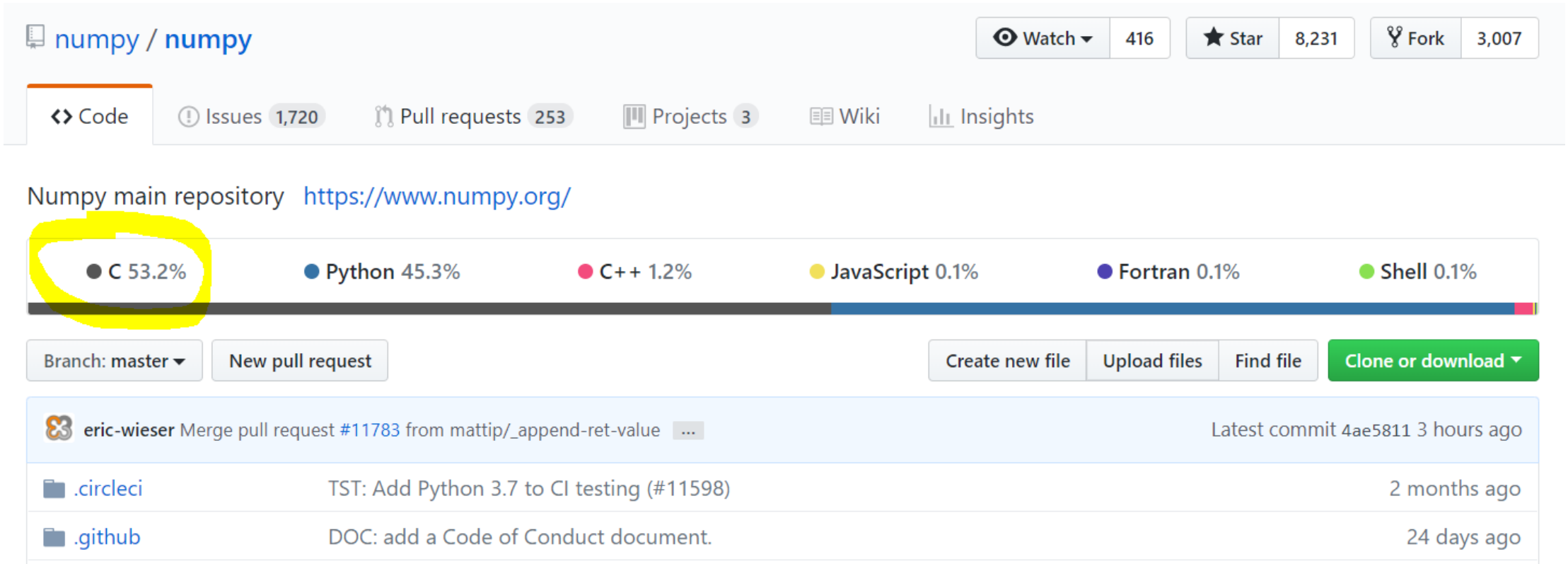


MATLAB

The two-language problem

- For **convenience**, use a scripting language
- but do all the **hard stuff** in a “systems language”

- Shows up as **black box packages** in scripting languages
- Creates a **social barrier** – a wall between users and developers


numpy / numpy
Watch 416
Star 8,231
Fork 3,007
Code
Issues 1,720
Pull requests 253
Projects 3
Wiki
Insights
Numpy main repository https://www.numpy.org/
C 53.2%
Python 45.3%
C++ 1.2%
JavaScript 0.1%
Fortran 0.1%
Shell 0.1%
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
eric-wieser Merge pull request #11783 from mattip/_append-ret-value ...
Latest commit 4ae5811 3 hours ago
.circleci TST: Add Python 3.7 to CI testing (#11598) 2 months ago
.github DOC: add a Code of Conduct document. 24 days ago

The two-language problem

static	dynamic
compiled	interpreted
user types	standard types
standalone	glue

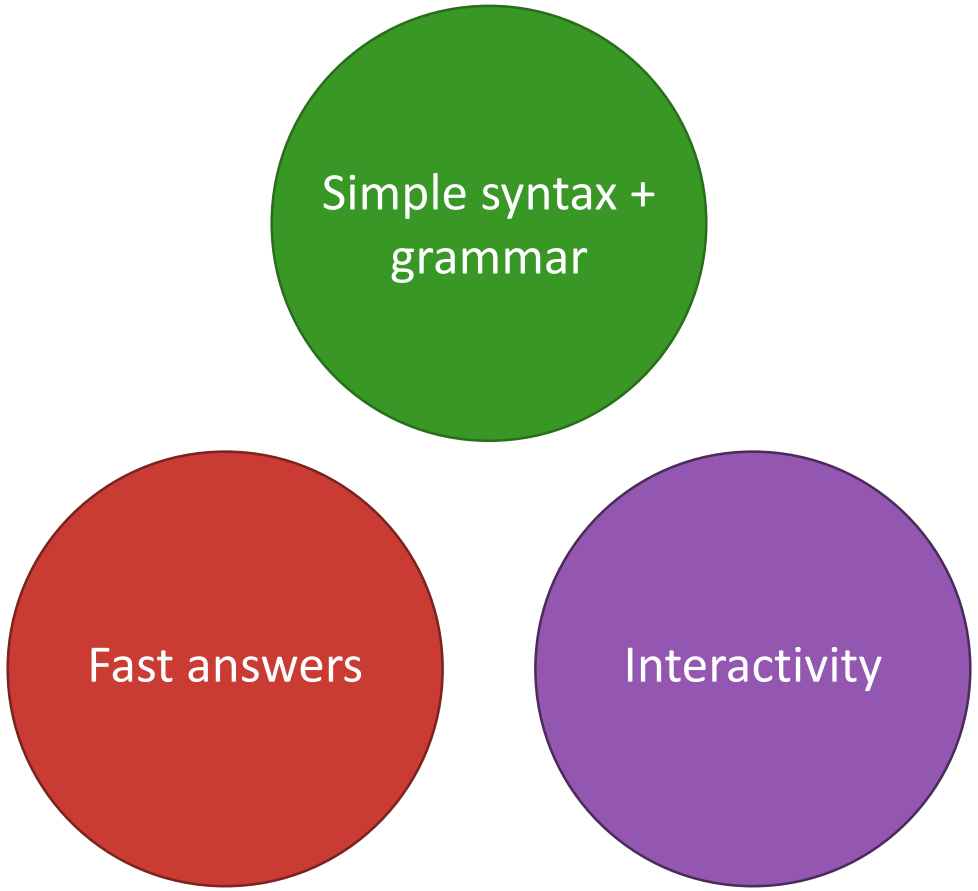
dynamic

compiled

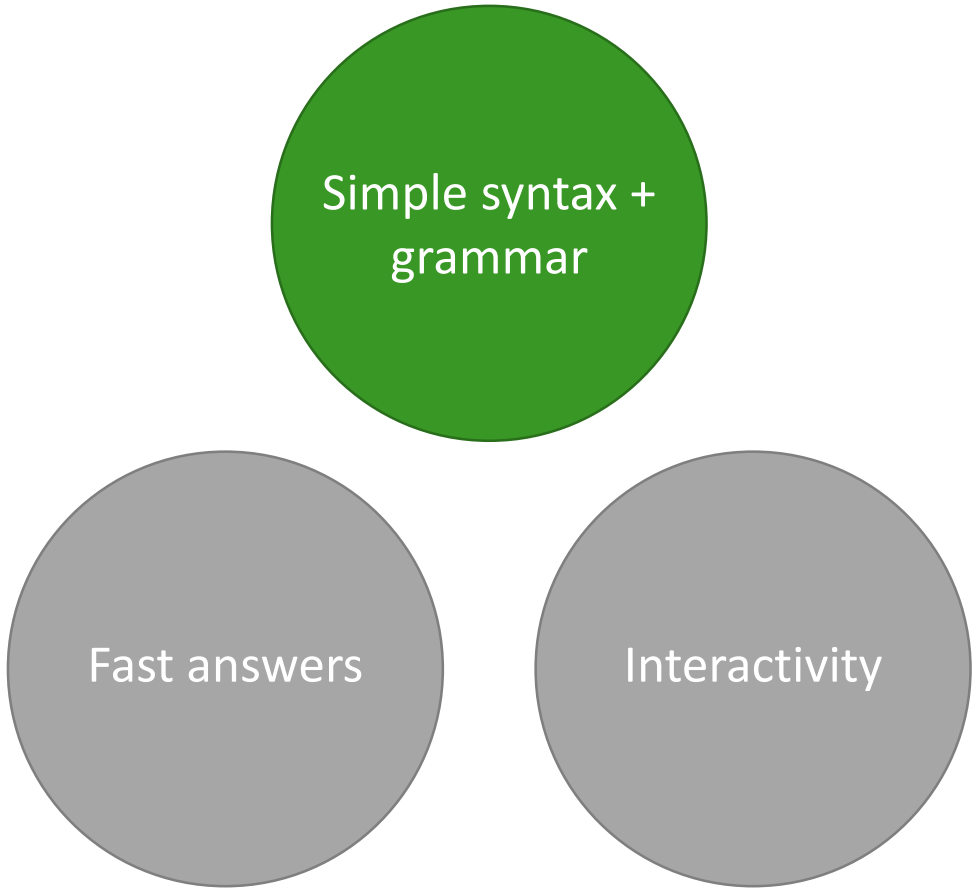
user types **and** standard types

standalone **or** glue

The julia unification



The julia unification



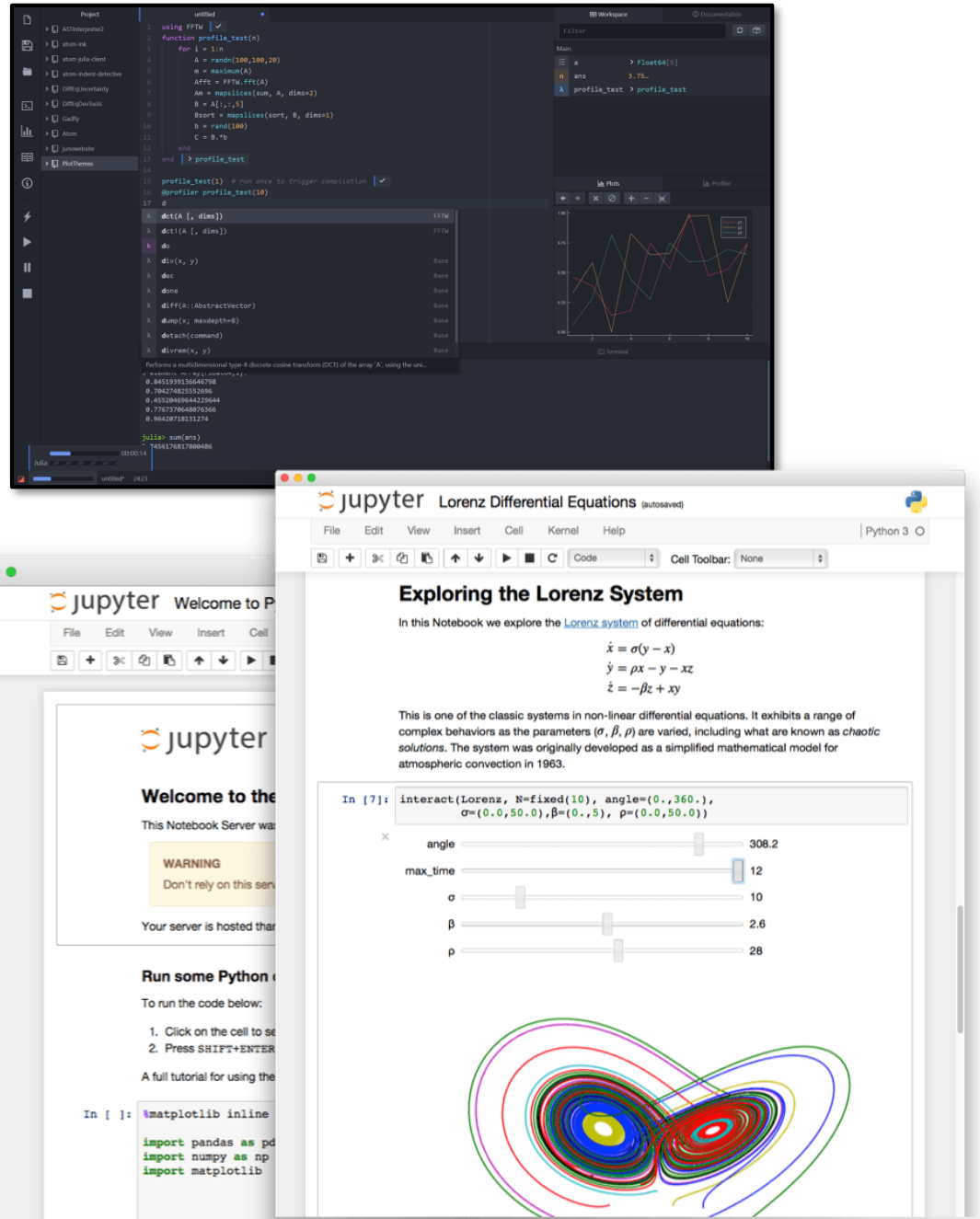

```
function babylonian(α; N = 10)
    @assert α > 0 "α must be > 0"
    t = (1+α)/2
    for i = 2:N
        t = (t + α/t)/2
    end
    t
end

babylonian(π) ≈ √π
```

The julia unification

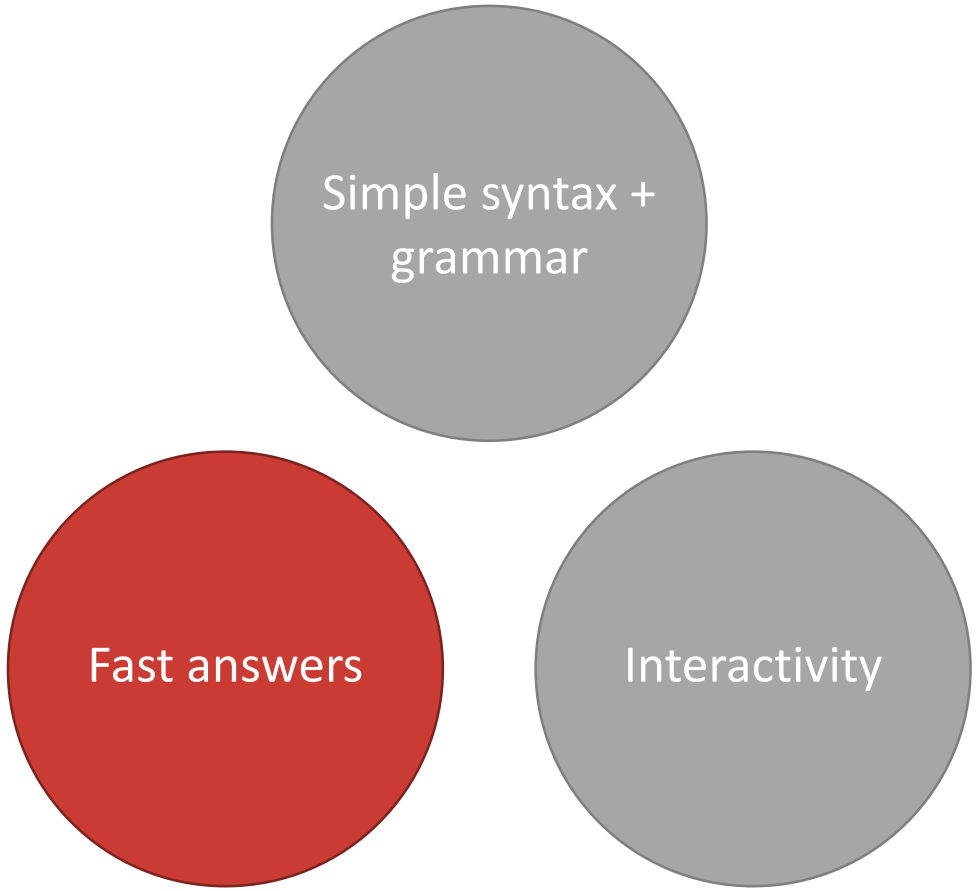
Simple syntax + grammar

Fast answers

Interactivity



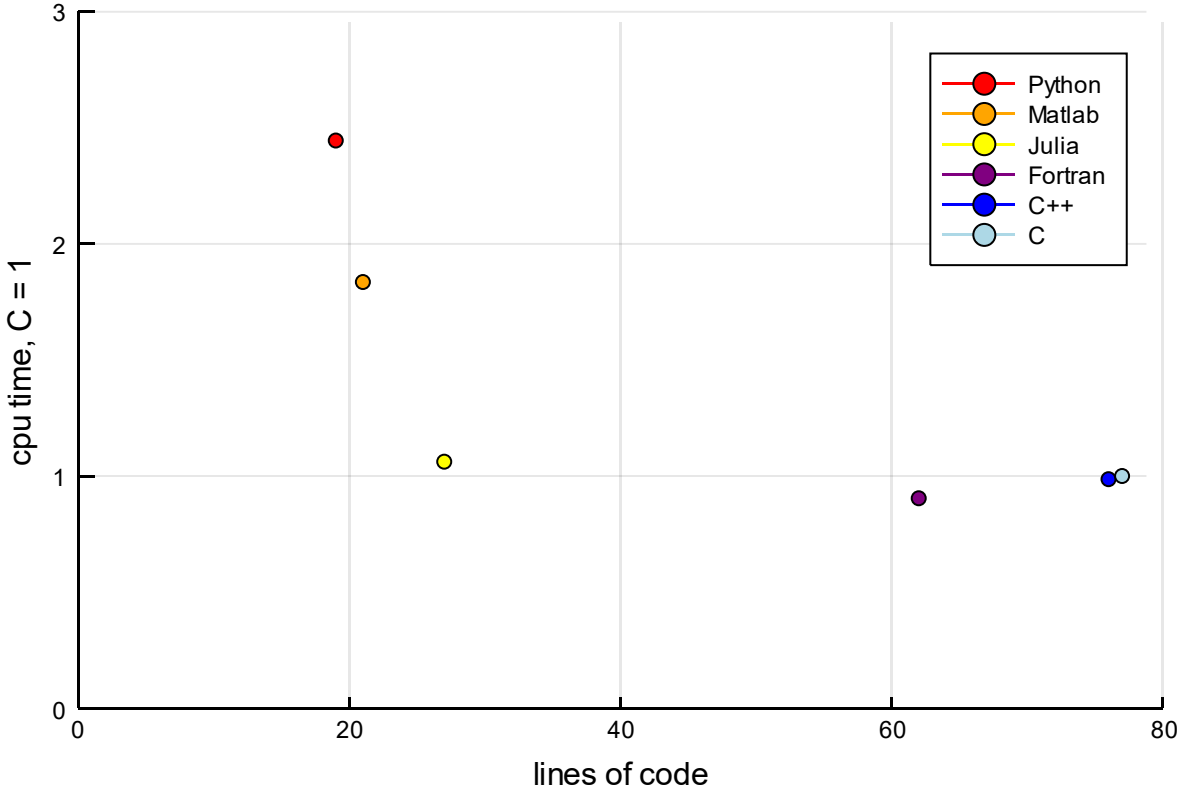
The julia unification



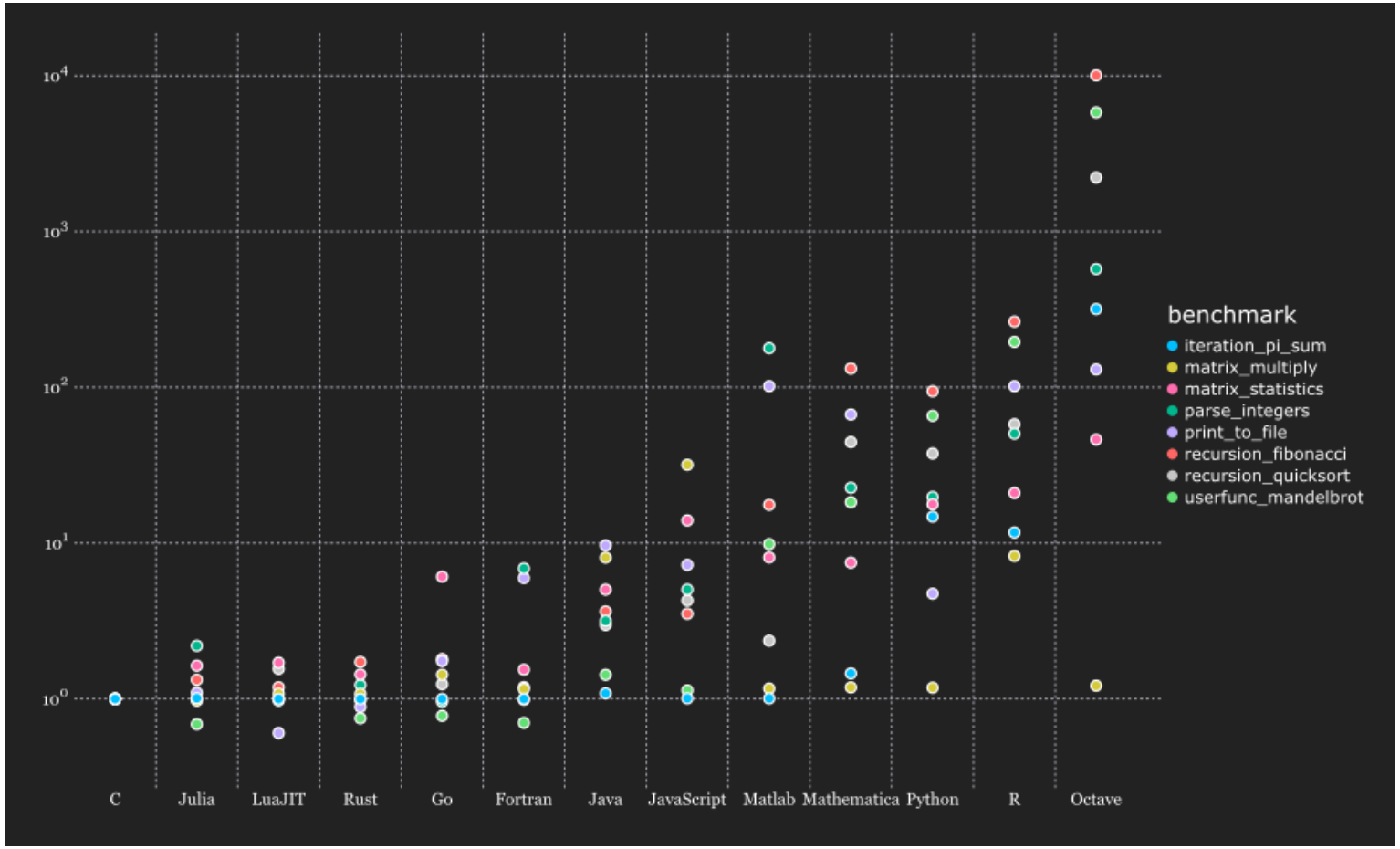

```
julia> function sumup()
           x = 0
           for i in 1:100
               x += i
           end
           x
       end
sumup (generic function with 2 methods)

julia> @code_llvm debuginfo=:none sumup()

; Function Attrs: uwtable
define i64 @julia_sumup_12626() #0 {
top:
  ret i64 5050
}
```

Just returns the answer!
(The for loop was compiled away)

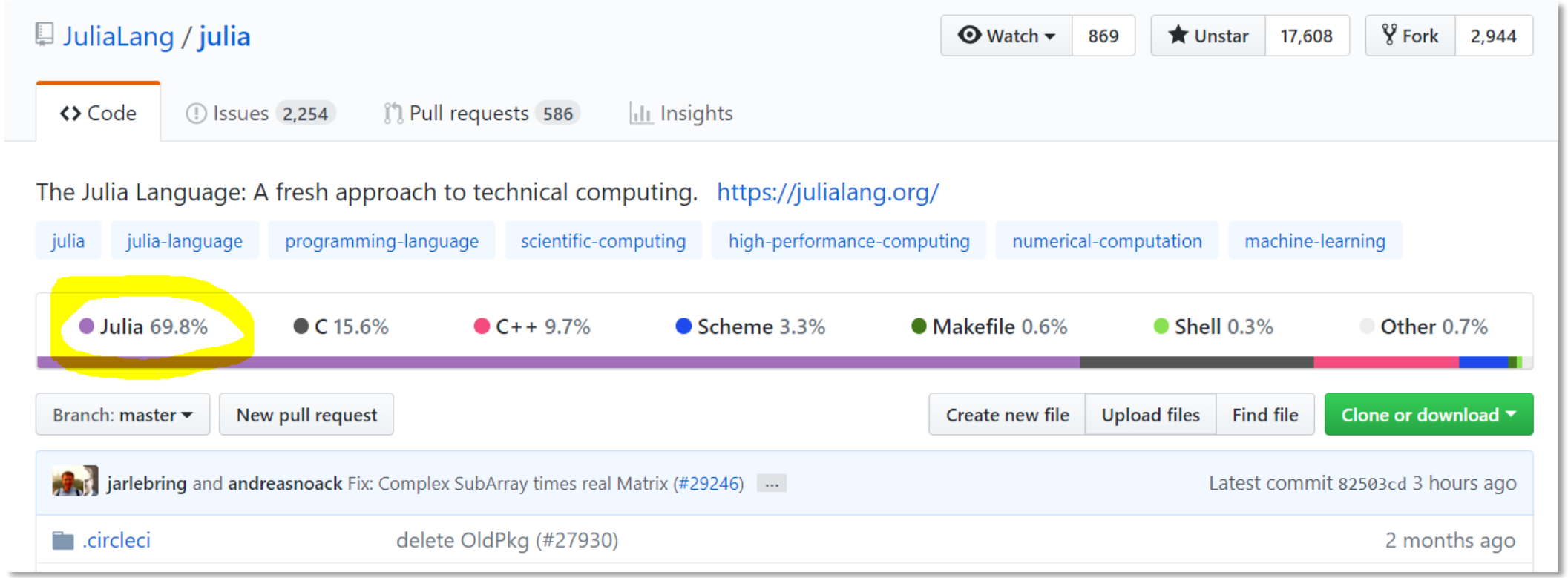
It is expressive



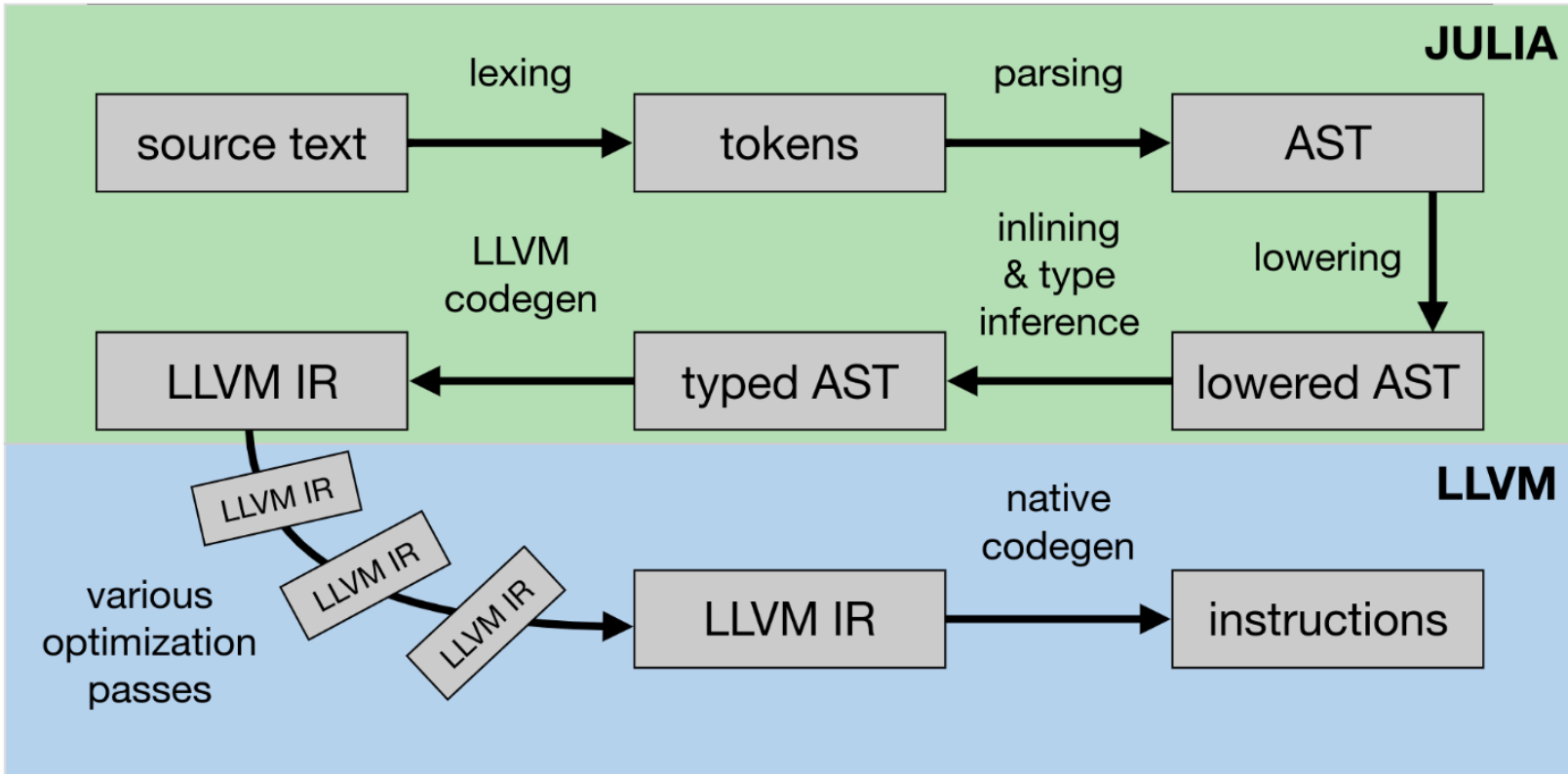
It is fast


10^4
10^3
10^2
10^1
10^0
C
Julia
LuaJIT
Rust
Go
Fortran
Java
JavaScript
Matlab
Mathematica
Python
R
Octave
benchmark
iteration_pi_sum
matrix_multiply
matrix_statistics
parse_integers
print_to_file
recursion_fibonacci
recursion_quicksort
userfunc_mandelbrot

It is free and open source



From Source to Machine Code



Recommended talks

Nick Eubank: What Julia Offers Academic Researchers

George Datseris: Why Julia is the most suitable language for science


Microsoft
Reactor London
juliacon
2018