Introduction to Julia

Cédric Simal
cedric.simal@unamur.be
Unamur, Naxys

CISM Training Sessions
09/11/22

Follow along!

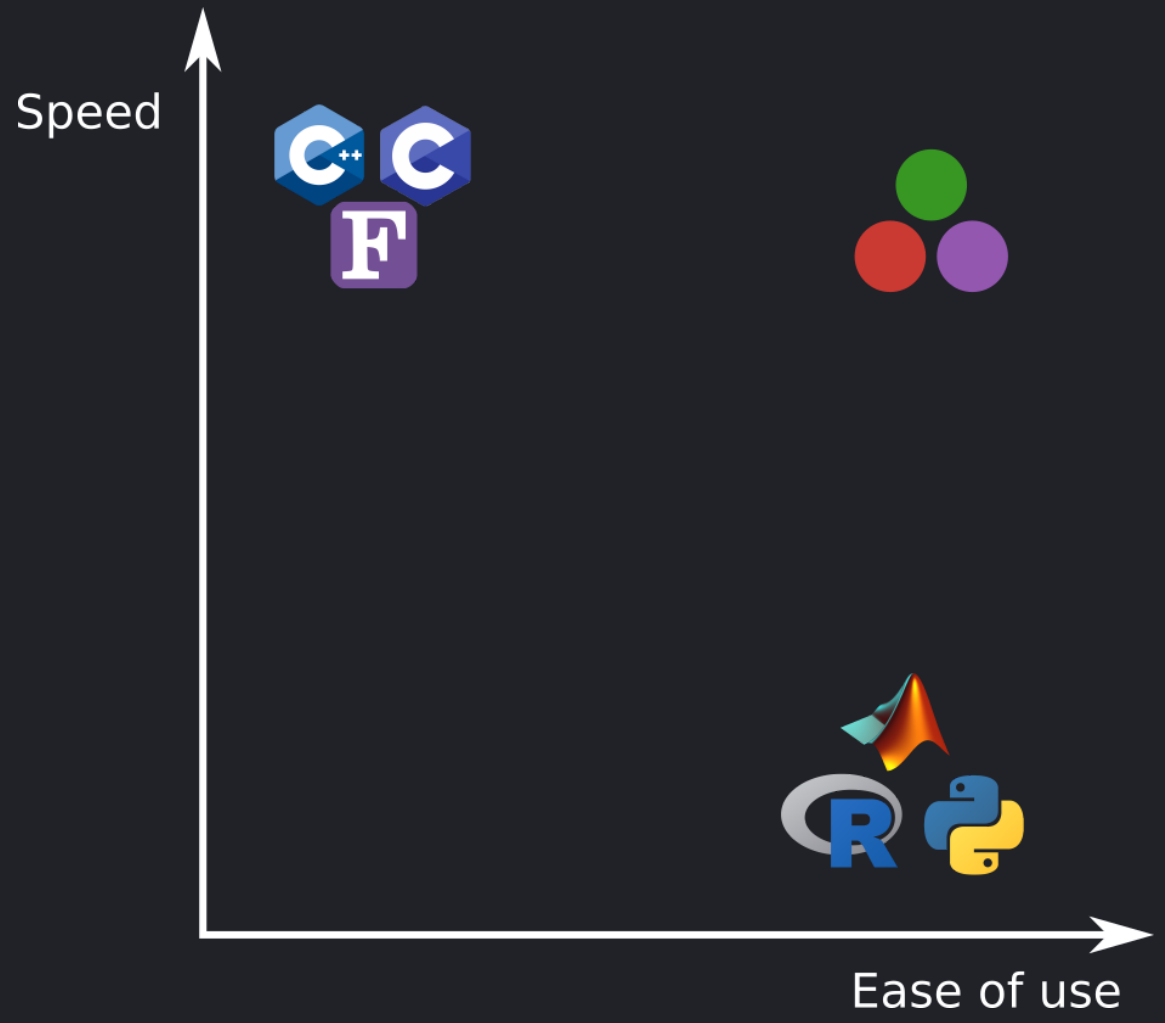
https://github.com/csimal/Julia-CISM

Outline

1. Basics of Julia
2. Multiple dispatch
3. Benchmarking Julia code
4. Parallel Programming

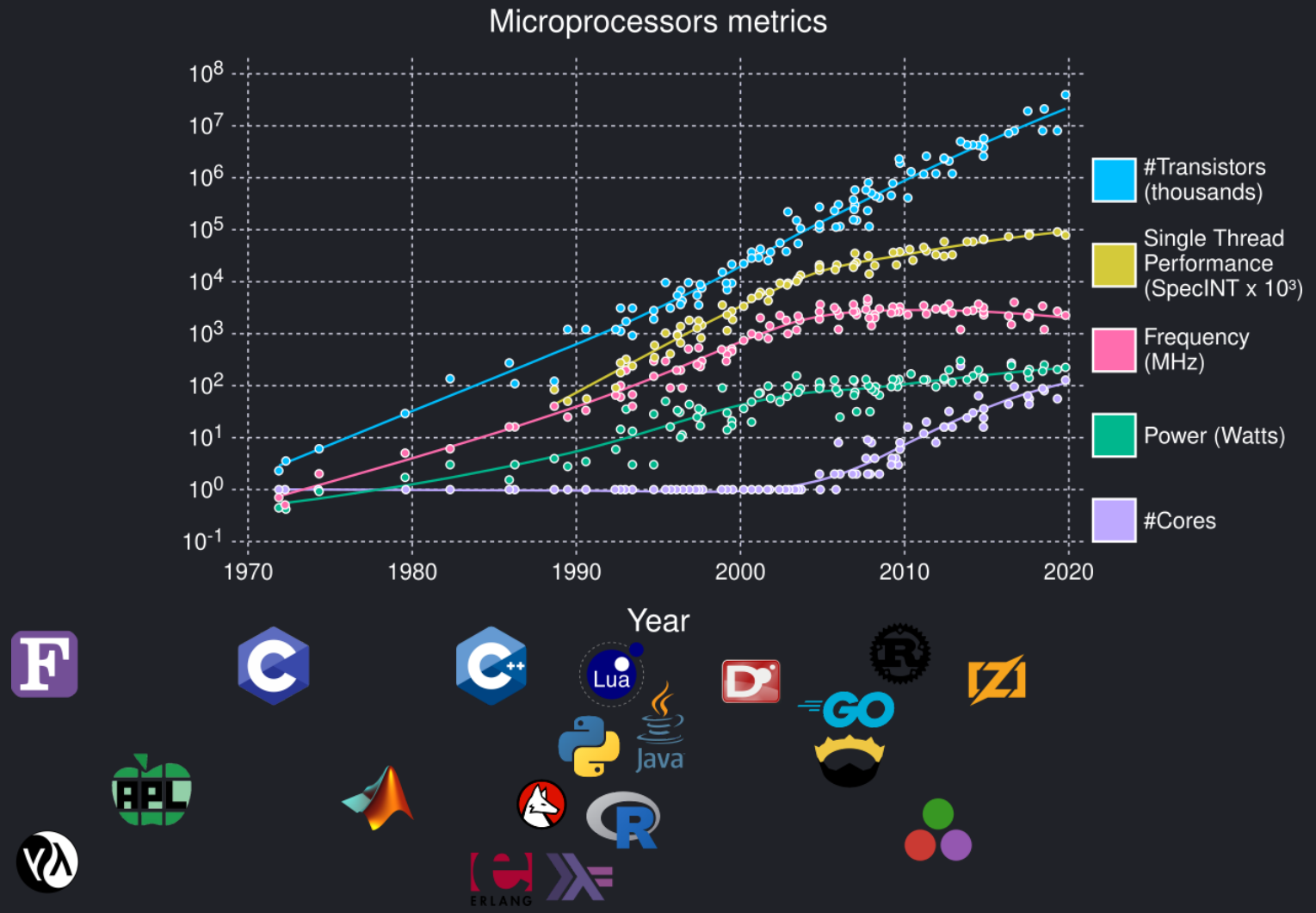
Modern Problems in Scientific Computing

The two languages problem



Modern Problems in Scientific Computing

The rise of parallel computing



data: https://github.com/karlrupp/microprocessor-trend-data

Meet Julia

Julia: A Fast Dynamic Language for Technical Computing

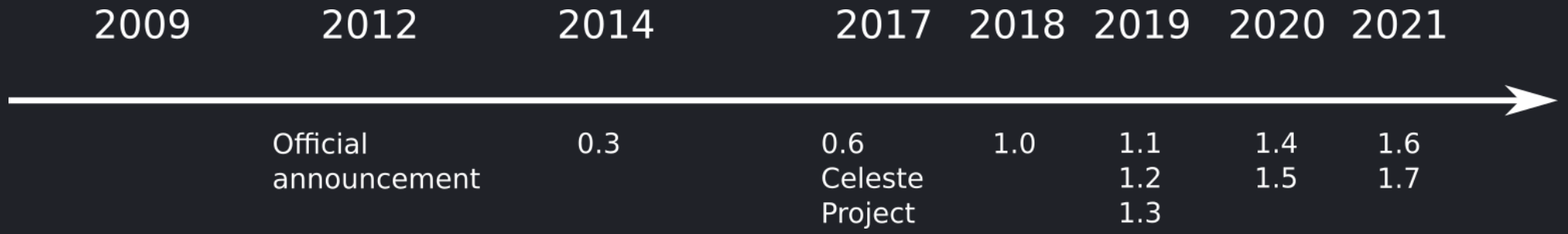
Jeff Bezanson* MIT
Stefan Karpinski† MIT
Viral B. Shah‡
Alan Edelman§ MIT

September 25, 2012

Abstract

Dynamic languages have become popular for scientific computing. They are generally considered highly productive, but lacking in performance. This paper presents Julia, a new dynamic language for technical computing, designed for performance from the beginning by adapting and extending modern programming language techniques. A design based on generic functions and a rich type system simultaneously enables an expressive programming model and successful type inference, leading to good performance for a wide range of programs. This makes it possible for much of Julia's library to be written in Julia itself, while also incorporating best-of-breed C and Fortran libraries.

A short history of Julia



Julia Joins Petaflop Club

September 12, 2017

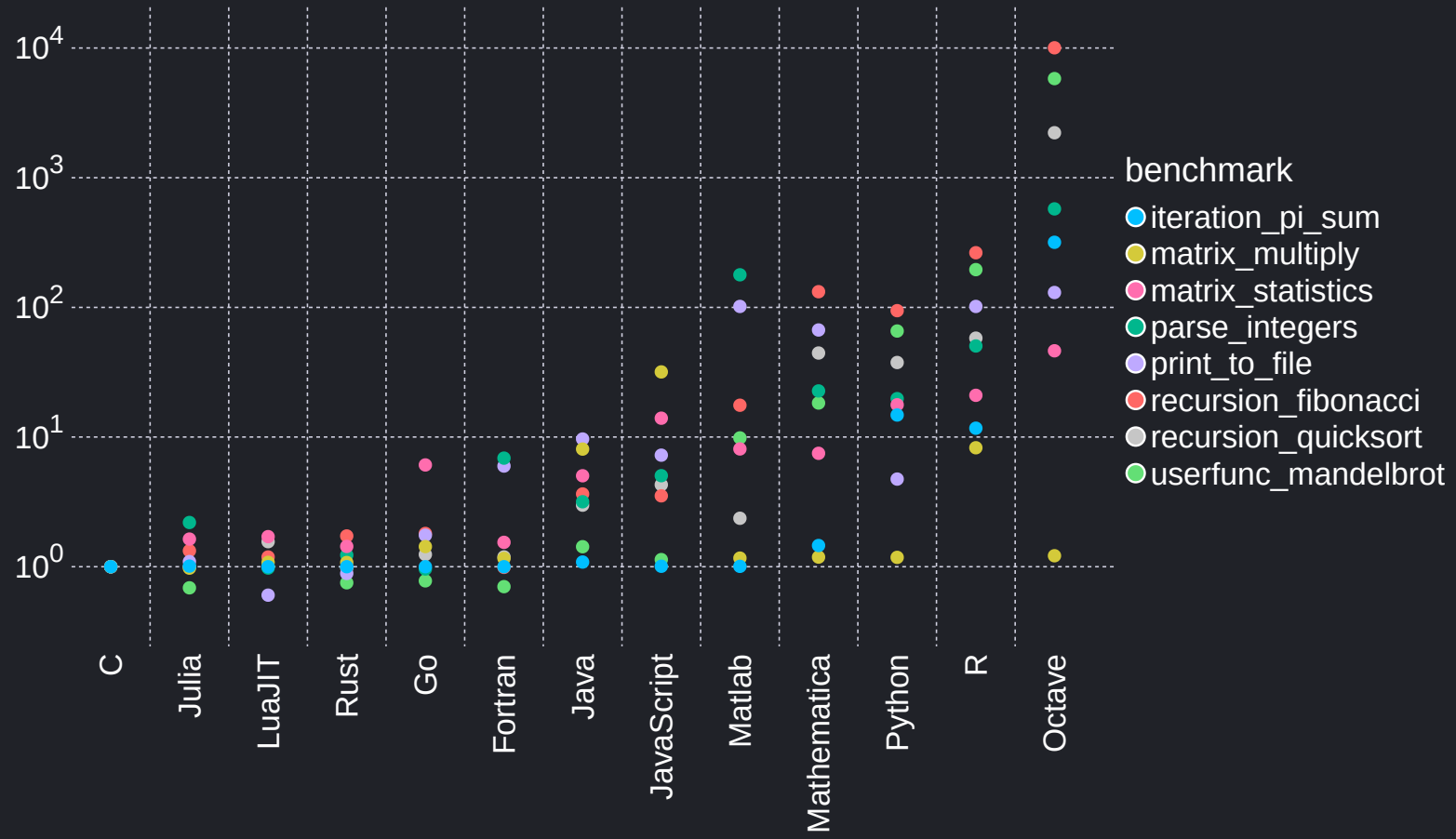
BERKELEY, Calif., Sept. 12, 2017 — Julia has joined the rarefied ranks of computing languages that have achieved peak performance exceeding one petaflop per second – the so-called 'Petaflop Club.'

The Julia application that achieved this milestone is called Celeste. It was developed by a team of astronomers, physicists, computer engineers and statisticians from UC Berkeley, Lawrence Berkeley National Laboratory, National Energy Research Scientific Computing Center (NERSC), Intel, Julia Computing and the Julia Lab at MIT.

https://juliacomputing.com/case-studies/celeste/

Why you might want to use Julia

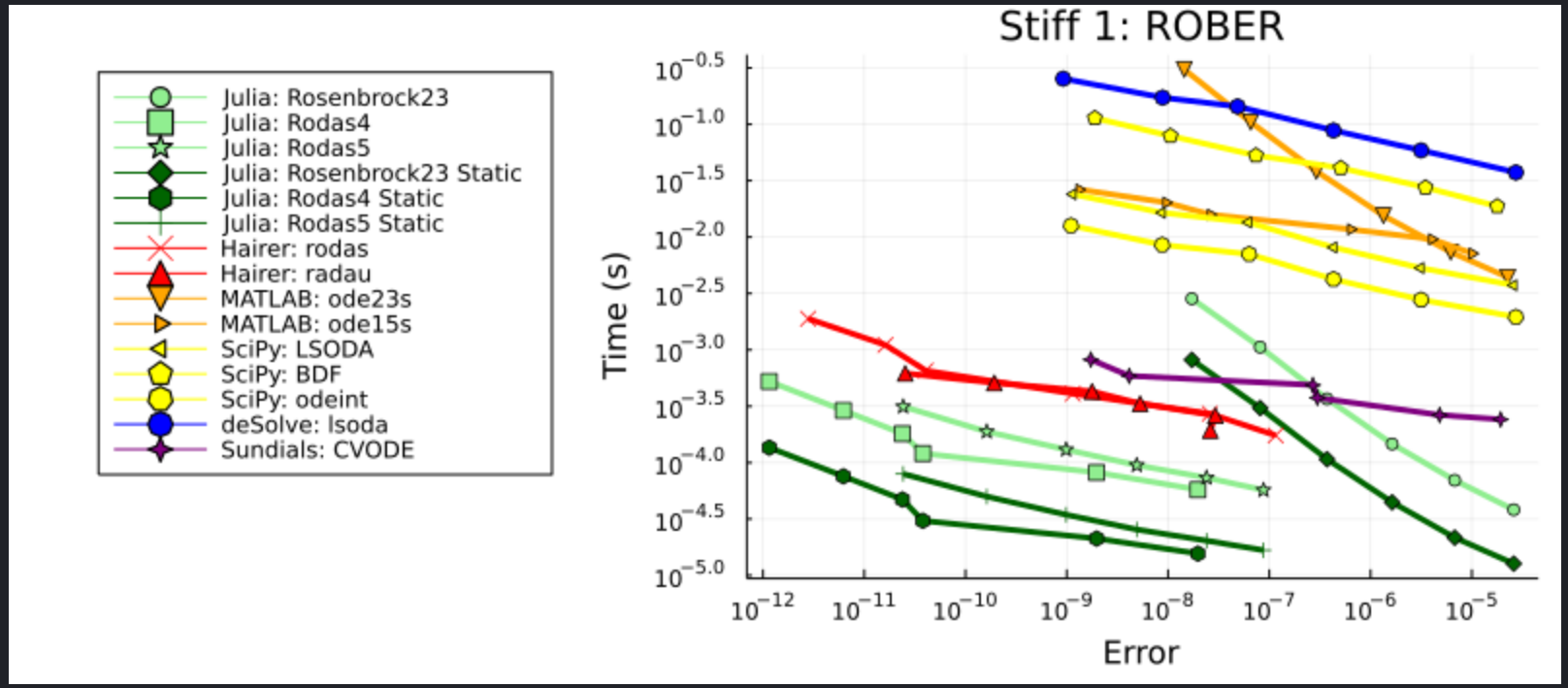
Julia is fast



https://julialang.org/benchmarks/

Why you might want to use Julia

DifferentialEquations.jl is SOTA



https://benchmarks.sciml.ai/stable/MultiLanguage/ode_wrapper_packages/

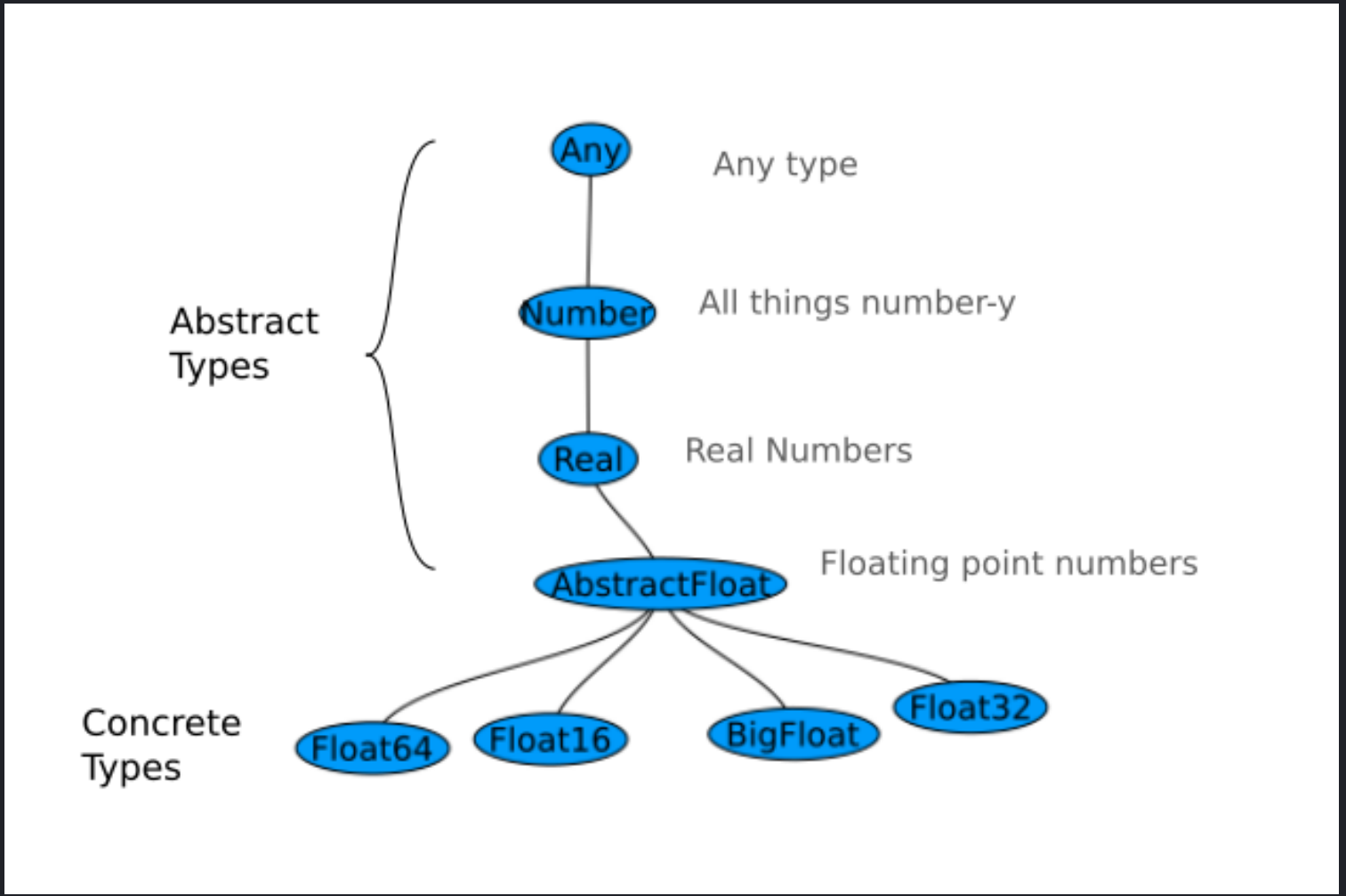
Why you might want to use Julia

- Open Source (MIT license)
  - It's free to use!
  - Faster development than proprietary languages
- Made for numerical computing
- Unicode
  - Code looks like math
- It's secretly a LISP

Why you might *NOT* want to use Julia

- Open Source
  - Documentation can suck
  - Projects with only one maintainer
- Time to first X
  - Parts of your programs have to be recompiled on startup
- Error messages are not beginner-friendly
- Tiny compared to Python
- Array indices start at 1

Types


Abstract Types
Any
Any type
Number
All things number-y
Real
Real Numbers
AbstractFloat
Floating point numbers
Concrete Types
Float64
Float16
BigFloat
Float32

Type System

Julia's type system is

- Dynamic, with optional type annotations `x::Int`
- Parametric `Vector{T}` (*generic types*)
- Hierarchical (subtyping) `Float64 <: Real`

First class citizens

In Julia, the following objects are *first class citizens*

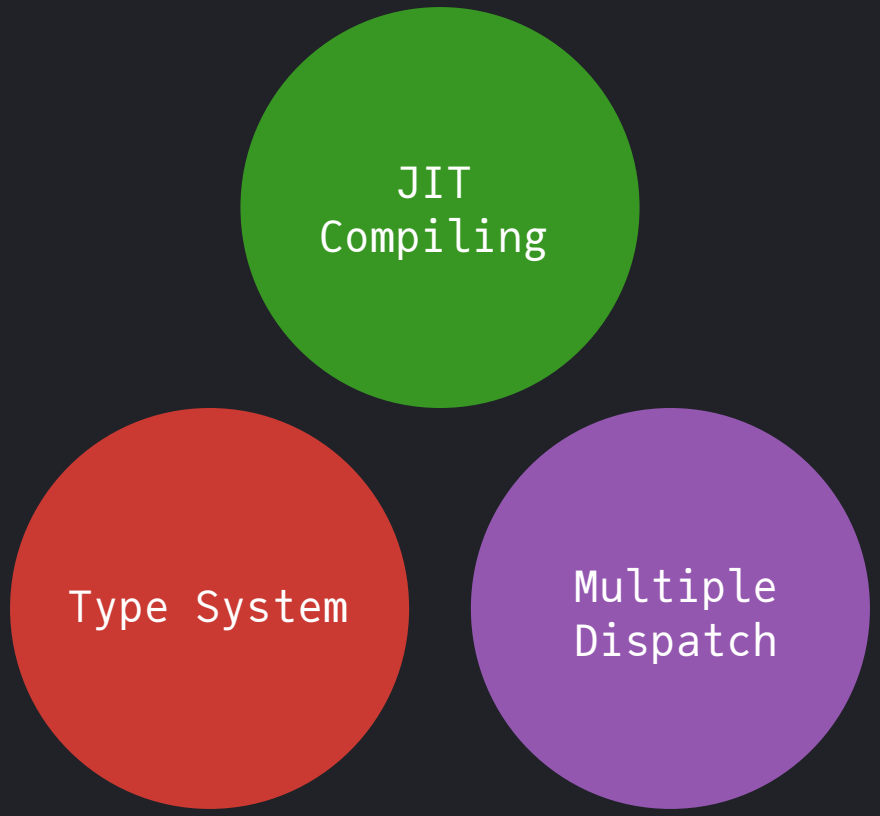
- Functions ($\Rightarrow$ functional programming)
- Types (`Type{T}`)
- Julia Expressions ($\Rightarrow$ Metaprogramming)

Multiple Dispatch

```
julia> +
+ (generic function with 198 methods)

julia> methods(+)
# 198 methods for generic function "+":
[1] +(x::Bool, z::Complex{Bool}) in Base at complex.jl:282
[2] +(x::Bool, y::Bool) in Base at bool.jl:96
[3] +(x::Bool) in Base at bool.jl:93
[4] +(x::Bool, y::T) where T<:AbstractFloat in Base at bool.jl:104
[5] +(x::Bool, z::Complex) in Base at complex.jl:289
[6] +(a::Float16, b::Float16) in Base at float.jl:398
[7] +(x::Float32, y::Float32) in Base at float.jl:400
[8] +(x::Float64, y::Float64) in Base at float.jl:401
[9] +(z::Complex{Bool}, x::Bool) in Base at complex.jl:283
[10] +(z::Complex{Bool}, x::Real) in Base at complex.jl:297
...
```

The secret sauce behind Julia's speed



https://github.com/csimal/Julia-CISM

cedric.simal@unamur.be