



# A brief Introduction to the Julia Programming Language

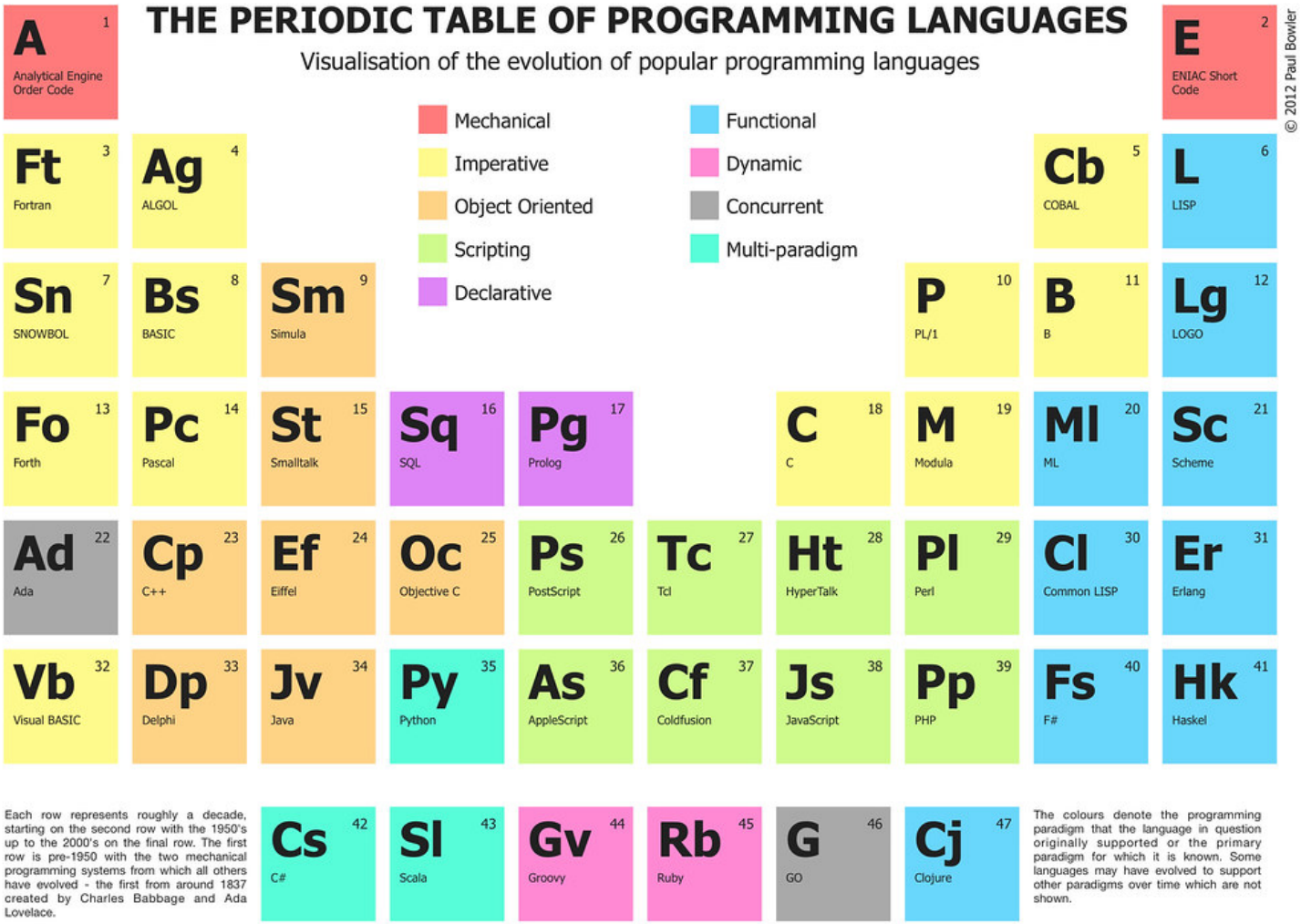
**Instructors**:
Damian Belz (DB), Albert Piwonski (AP), Rodrigo Rezende (RR).

**Scientific board**:
Mirsad Hadžiefendić, Marcus Christian Lehmann.

**Date & Location**:
June 24th 2019, 9 AM – 6 PM,
Einsteinufer 17, 10587 Berlin, Room EN-616/617.

# Table of contents

# Motivation




1

[1] *By Paul Bowler.*

# Motivation

There is still a big problem within the programming languages! It is called **The Two language problem.**

**Performant (fortran, c, asm)**

- lower level
- efficient machine code
- running quickly

**Productive (python, ruby, MatLab)**

- higher level
- allow the user to write the code quickly and easily

**performance vs. productivity vs. generality**

# Motivation

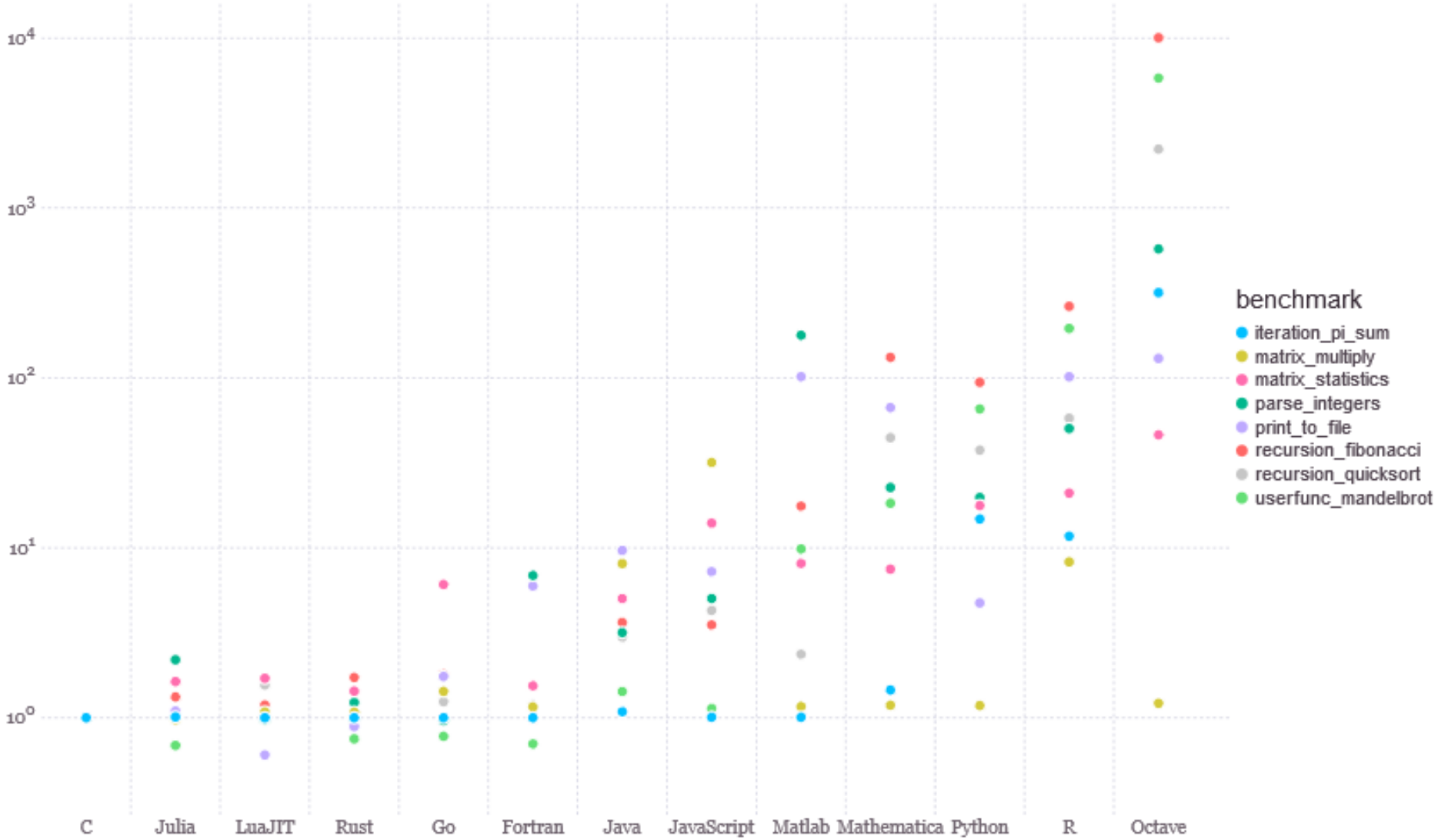
- The classical workaround is:
  - use two languages (prototype + production) → "two language problem"
- New idea is:
  - use Julia: "looks like **python**, feels like **lisp**, runs like **C**" → **productivity**, **generality** and **performance**

## "runs like **C**"-example

$$sum(a) = \sum_{i=1}^{n} a_i, \quad n = 10^7$$

C: 10ms
python: 500ms
Julia: 10ms

# Motivation




[1]

# Motivation

## "looks like **python**"-example

- python

```
def sum(a):
    s=0.0
    for x in a:
        s += x
    return s
```

- julia

```
function sum(a):
    s=0.0
    for x in a:
        s += x
    end
    return s
end
```

# Motivation

**"feels like lisp"-example**

metaprogramming!
e.g. macros, multiple dispatch, ...

This allows a program to transform and **generate** its **own code**

```
julia> @generated function foo(x)
           Core.println(x)
           return :(x * x)
       end
foo (generic function with 1 method)
```

From this **generality** → *Julia is mostly written in Julia*

# Julia in practice

## Celeste Project [2]

- In 2014 Julia was used to develop a parallel program to process the Sloan Digital Sky Survey in the Berkeley Lab
- With this program 188 million objects were cataloged in just 14.6 minutes
- Some milestones of the program:
  - ~ 178 terabytes of data were processed in this time
  - peak performance was 1.54 petaflops using 1.3 million threads on 9300 (KNL) nodes (the third language to achieve this after Fortran and C)

## Some statistics *from Aug 2018

- Over 1900 registred packages
- Over 2 million downloads
- 101% annual growth
- Over 41000 Github stars (Julia & packages)

Time to work!


[3]

# Bibliography I


[1] Julia Micro-Benchmarks, https://julialang.org/benchmarks/

[2] Parallel Supercomputing for Astronomy, https://juliacomputing.com/case-studies/celeste.html

[3] Julia Box, https://juliabox.com

[4] Programming paradigms, https://en.wikipedia.org/wiki/Programming_paradigm

# Extra Slide

## Programming paradigms [4]

- **Imperative**: the programmer instructs the machine how to change its states
- **Object-oriented**: it groups instructions together with the part of the state they operate on
- **Declarative**: the programmer merely declares properties of the desired result, but not how to compute it
- **Functional**: the desired result is declared as the value of a series of function applications