

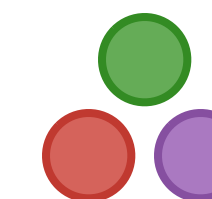


A Fresh Approach to Technical Computing

Stefan Karpinski



Numerical languages



What's the deal with these?

- ▶ specialized for **numerical** work

Matlab

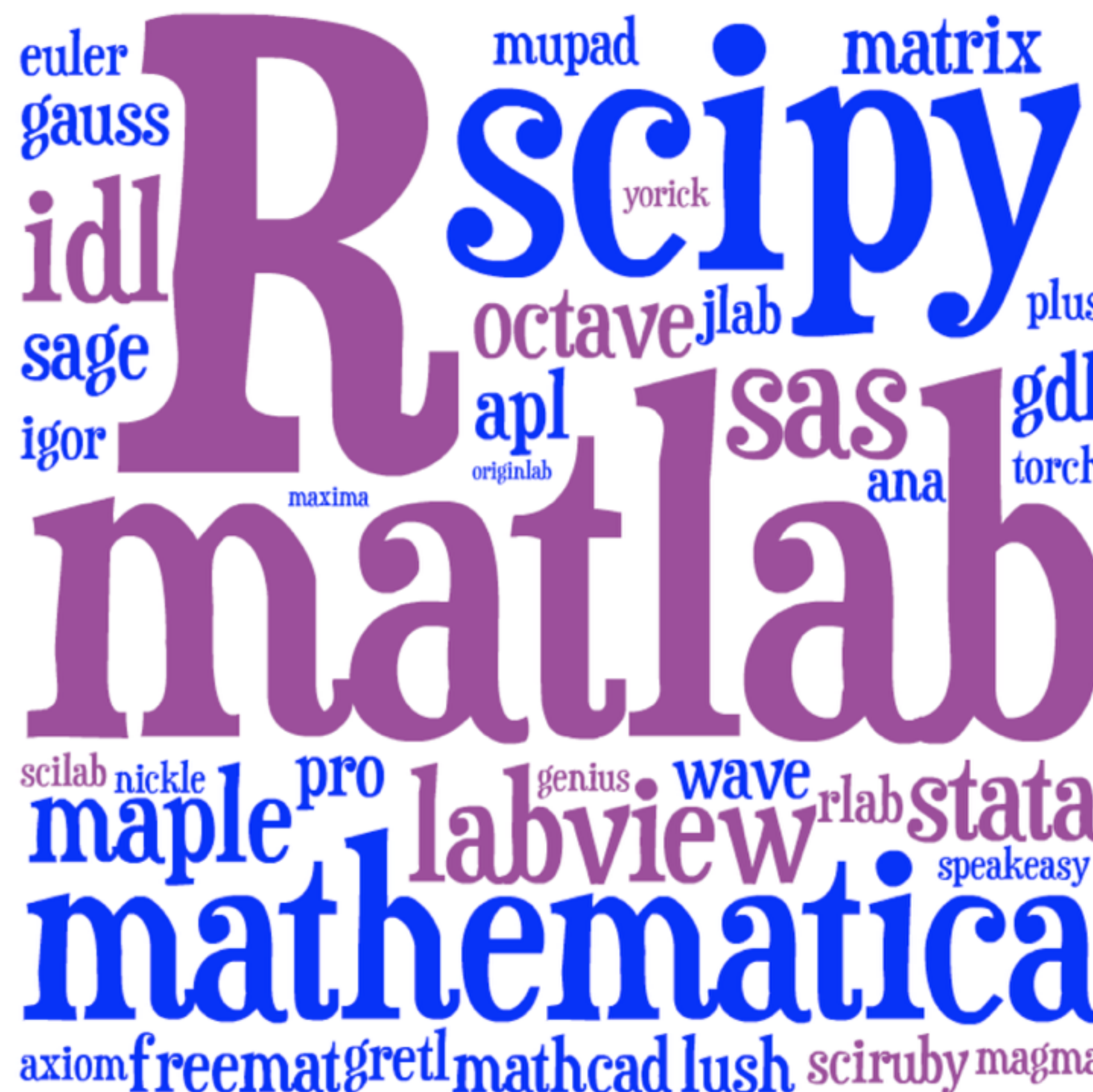
- ▶ everything is a complex matrix

R (and S before it)

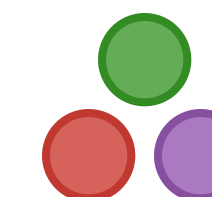
- ▶ allow “NA” values everywhere
- ▶ data frame as basic data type

Mathematica

- ▶ symbolic rewriting everywhere



So... is Scheme numerical?

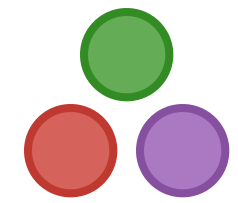


R⁶RS spec:

- ▶ **20% numerical**
- ▶ (C99 is similar)

CONTENTS	
Introduction	3
Description of the language	
1 Overview of Scheme	5
1.1 Basic types	5
1.2 Expressions	6
1.3 Variables and binding	6
1.4 Definitions	6
1.5 Forms	7
1.6 Procedures	7
1.7 Procedure calls and syntactic keywords	7
1.8 Assignment	7
1.9 Derived forms and macros	8
1.10 Syntactic data and datum values	8
1.11 Continuations	8
1.12 Libraries	9
1.13 Top-level programs	9
2 Requirement levels	9
3 Numbers	10
3.1 Numerical tower	10
3.2 Exactness	10
3.3 Fixnums and flonums	10
3.4 Implementation requirements	10
3.5 Infinities and NaNs	11
3.6 Distinguished -0.0	11
4 Lexical syntax and datum syntax	11
4.1 Notation	12
4.2 Lexical syntax	12
4.3 Datum syntax	16
5 Semantic concepts	17
5.1 Programs and libraries	17
5.2 Variables, keywords, and regions	17
5.3 Exceptional situations	18
7.1 Library form	23
7.2 Import and export levels	25
7.3 Examples	26
8 Top-level programs	27
8.1 Top-level program syntax	27
8.2 Top-level program semantics	28
9 Primitive syntax	28
9.1 Primitive expression types	28
9.2 Macros	29
10 Expansion process	29
11 Base library	31
11.1 Base types	31
11.2 Definitions	31
11.3 Bodies	32
11.4 Expressions	32
11.5 Equivalence predicates	37
11.6 Procedure predicate	39
11.7 Arithmetic	39
11.8 Booleans	47
11.9 Pairs and lists	47
11.10 Symbols	49
11.11 Characters	50
11.12 Strings	50
11.13 Vectors	51
11.14 Errors and violations	52
11.15 Control features	53
11.16 Iteration	55
11.17 Quasiquote	55
11.18 Binding constructs for syntactic keywords	56
11.19 Macro transformers	57
11.20 Tail calls and tail contexts	59
Appendices	
A Formal semantics	61
A.1 Background	61

Are we doing it wrong?

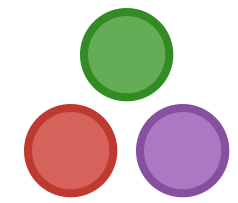


Numerical languages are strangely diverse

General languages are strangely numerical

This doesn't seem quite right.

Julia: a new approach

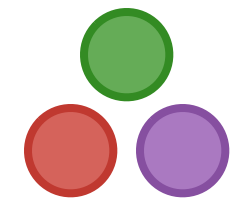


Stop making numbers **special**

All numeric types are **user-defined**

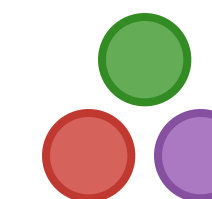
(Some of them are just defined for you)

The challenge



What does it take
to make numbers **just another type**
and still do real **numerical work?**

Numeric operations



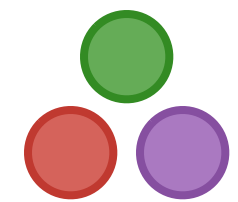
Operations like `+`, `*`, `[]`, `\`

- ▶ **unusually polymorphic** – often not normal functions
- ▶ behavior depends on **all arguments** not just first
- ▶ **extensibility** to new numeric types \iff *the expression problem*

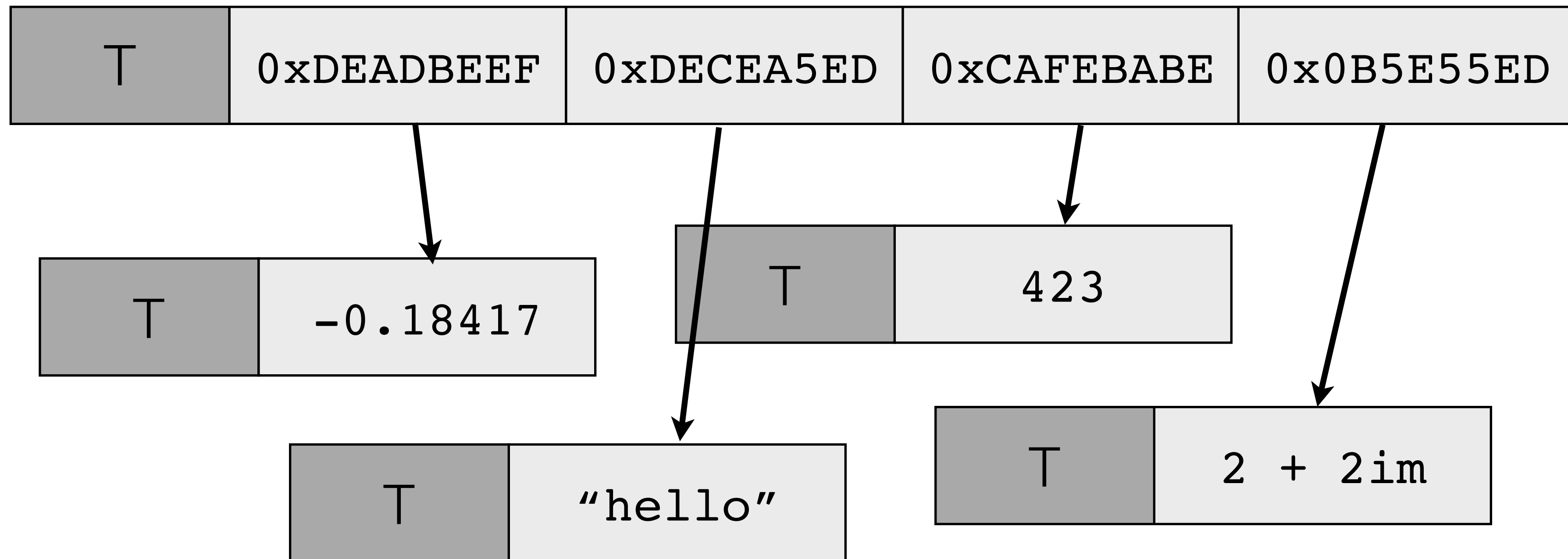
We need **multiple dispatch**

- ▶ but if ops like `Int+Int` and `Float64*Float64` are generic
- ▶ generic functions had better be **blazingly fast** (basically free)

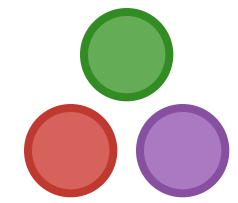
Arrays



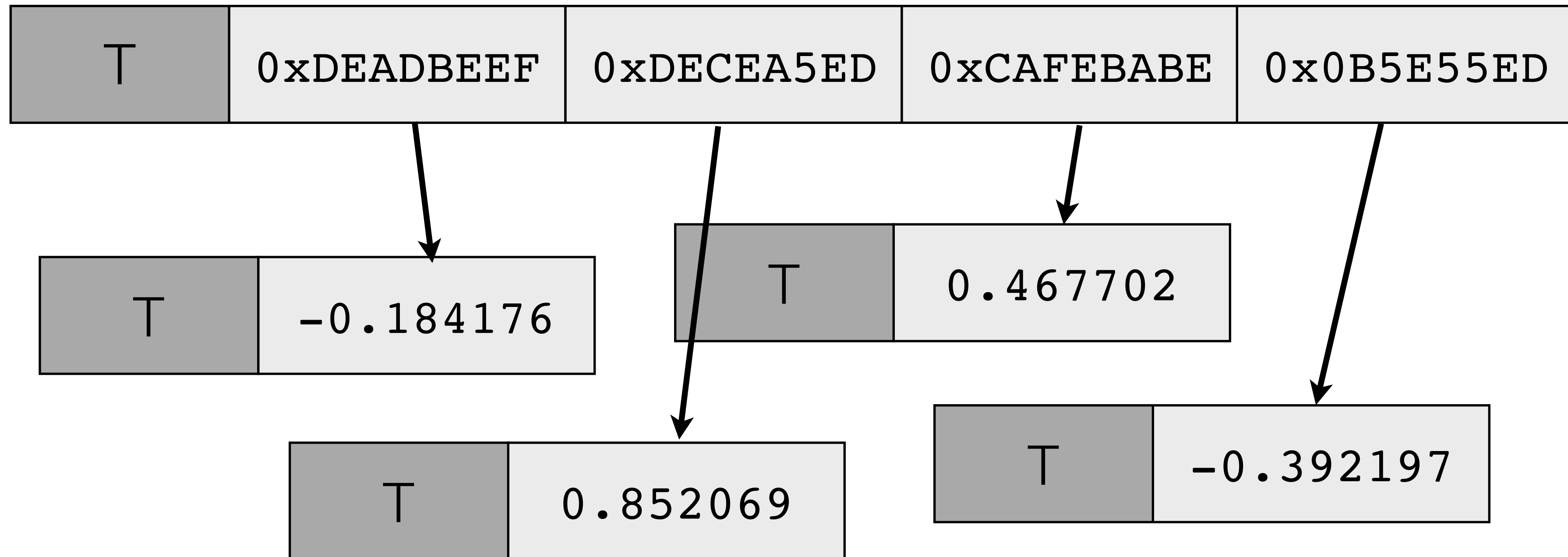
```
[-0.18417, "hello", 423, 2 + 2im]
```



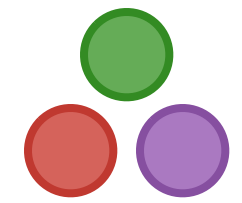
Arrays



`Any[-0.18417, 0.85206, 0.46770, -0.39219]`

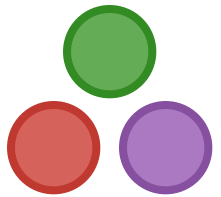


Staged programming

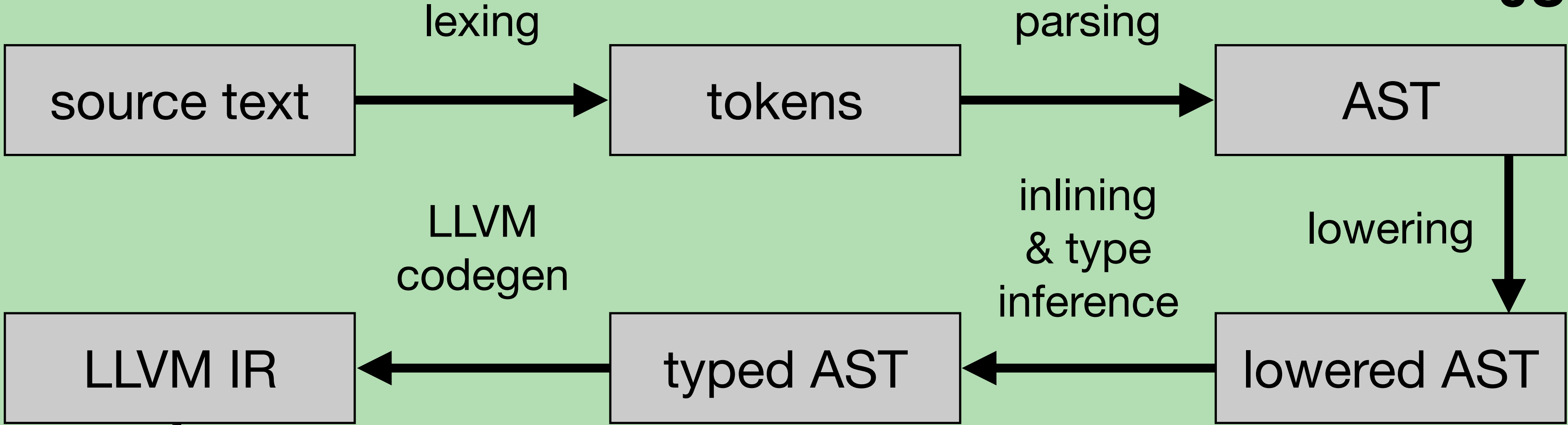


Allowing the programmer to generate code at various points in the compilation process

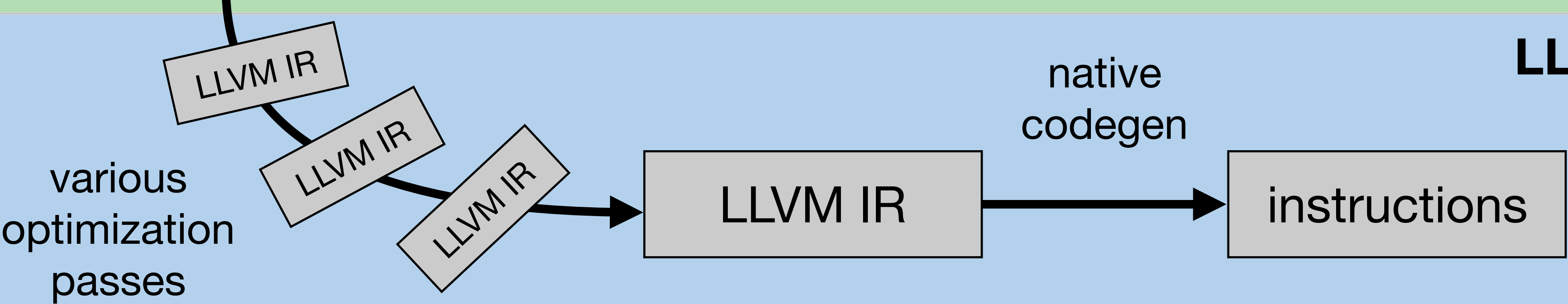
From Source to Machine Code



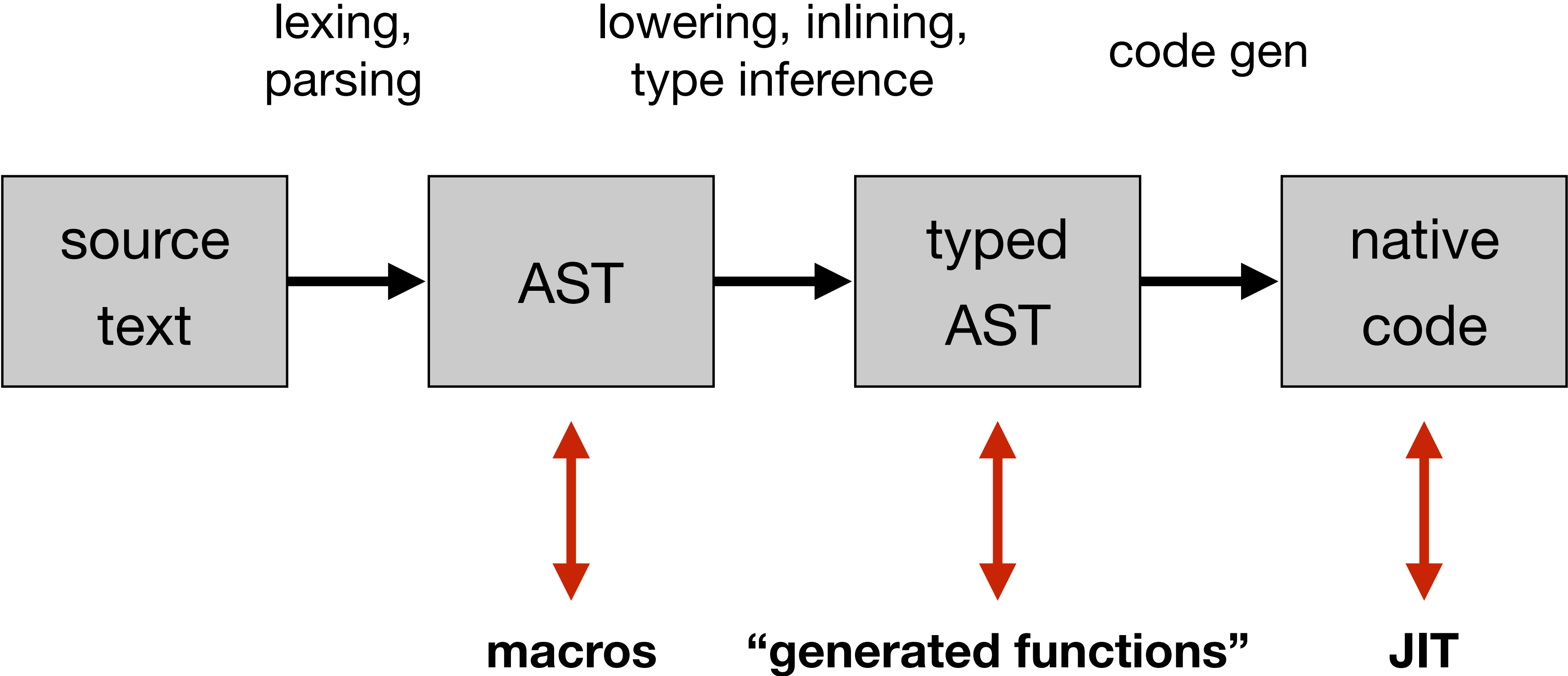
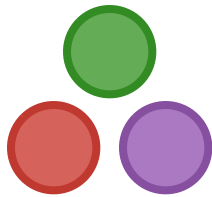
JULIA



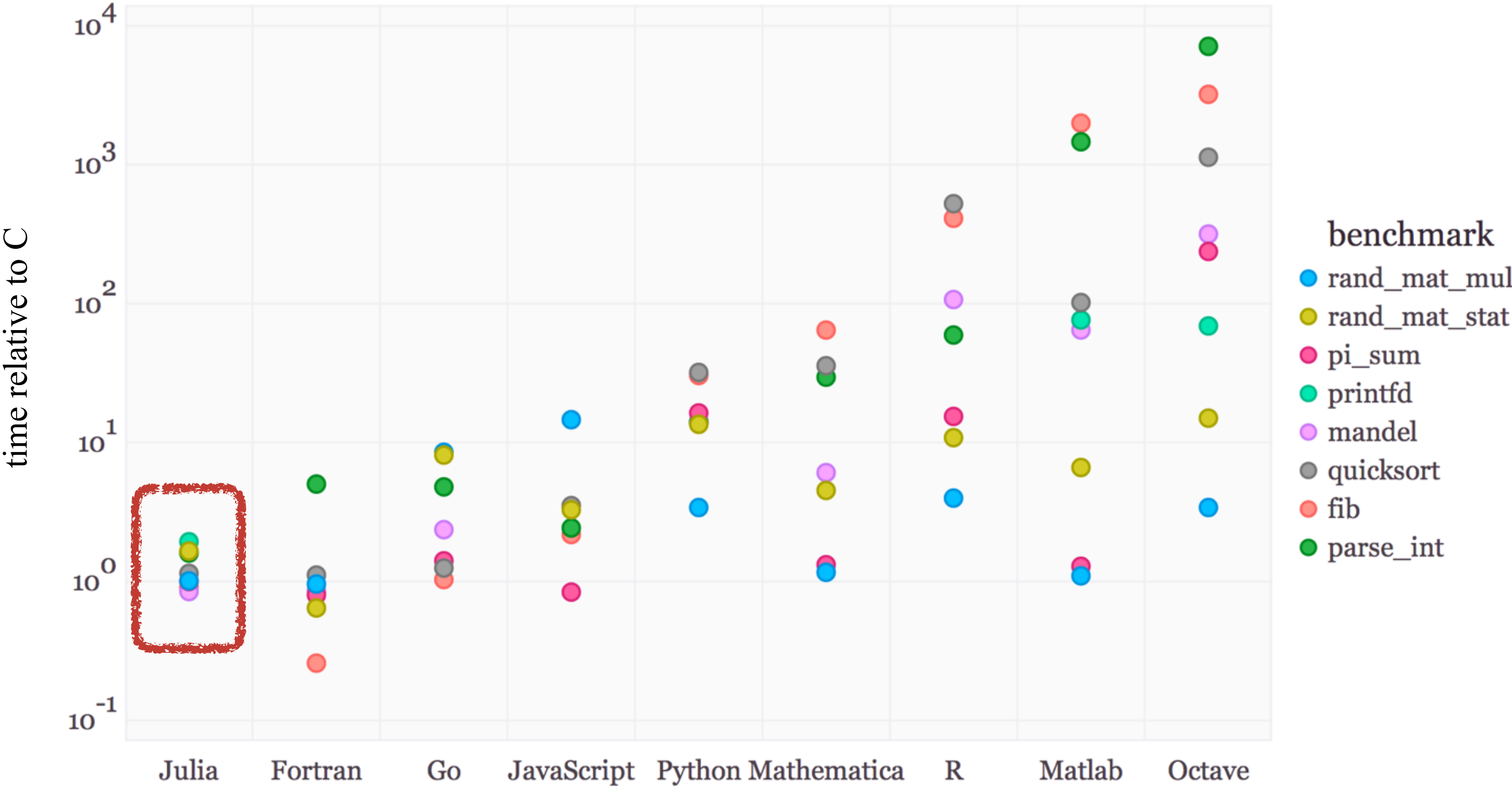
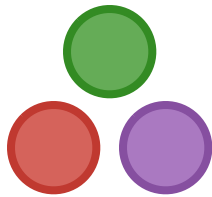
LLVM



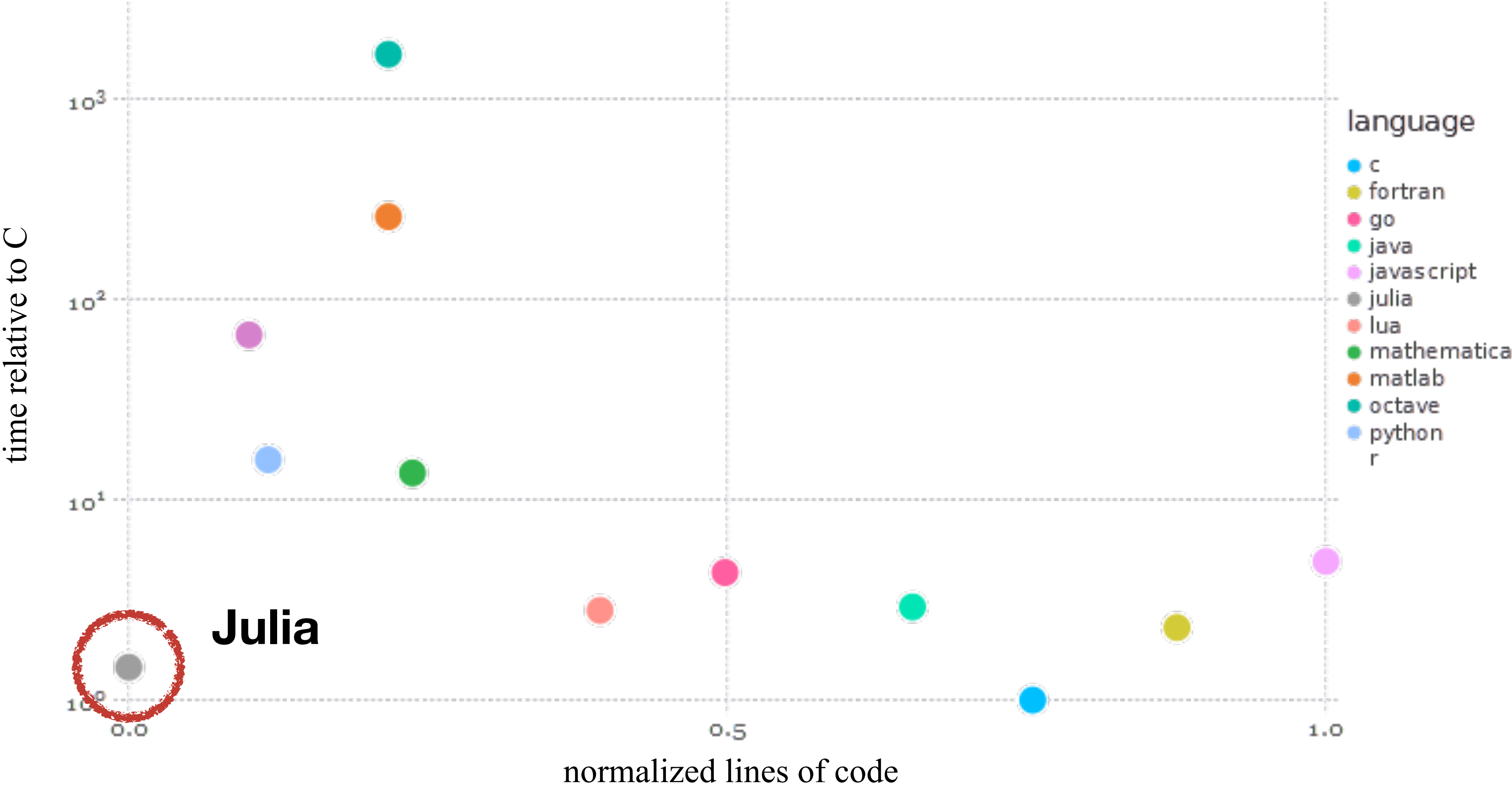
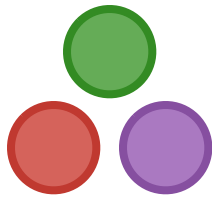
Hooking into compilation



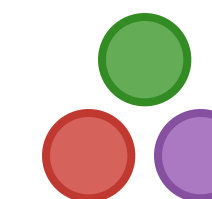
Speed



Speed vs. Productivity



Demos & Examples



- ▶ Simple generic programming: **nextfib**
- ▶ Interactive visualization: **Julia set**
- ▶ Efficient custom types: **Kakuro**
- ▶ Multiple dispatch: **notebook, promotion system**
- ▶ Macros & metaprogramming: **Horner & evalpoly**
- ▶ Generated functions: **nloops, Savitsky-Golay smoothing**