Julia Workshop

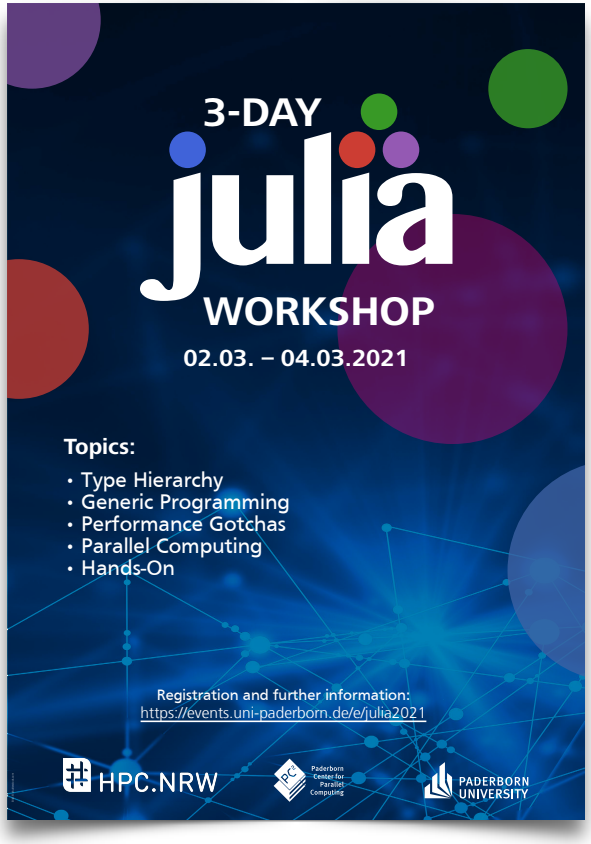
Carsten Bauer @ HPC.NRW, March 2021

thp Institute for Theoretical Physics University of Cologne

bcgs Bonn-Cologne Graduate School of Physics and Astronomy

Tentative schedule

	Tuesday	Wednesday	Thursday
9:30- 11:30 (2h)	Types + Dispatch	Performance Gotchas	Distributed-
	Short break		
11:45- 12:45 (1h)	Custom types	Linear Algebra	Computing
	Lunch break		
14:45- 15:45 (1h)	Specialization	Automatic Differentiation	Multithreading
	Short break		
16:00 – 17:00 (1h)	Generic Prog.	Environments	Q&A



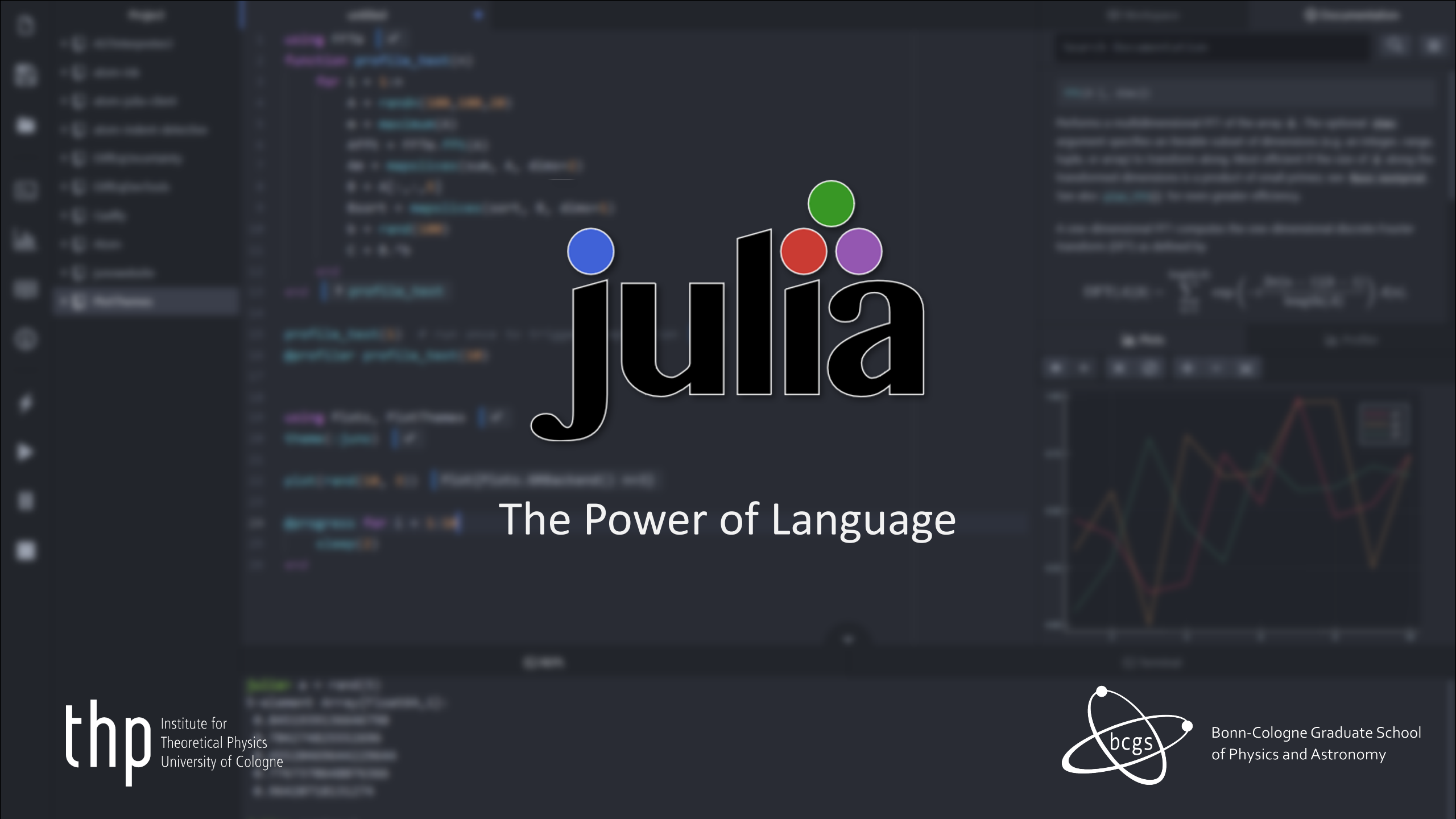
GitHub repository

github.com/crstnbr/JuliaNRW21

Zoom poll…

thp Institute for Theoretical Physics University of Cologne

bcgs Bonn-Cologne Graduate School of Physics and Astronomy



The Power of Language

thp Institute for Theoretical Physics University of Cologne

bcgs Bonn-Cologne Graduate School of Physics and Astronomy

Vandermonde matrix

$$V = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \dots & \alpha_m^{n-1} \end{bmatrix}$$

`vander(x)`

```
numpy.vander(x)
```

Python

```python
def vander(x, N=None, increasing=False):
    x = asarray(x)
    if x.ndim != 1:
        raise ValueError("x must be a one-dimensional array or sequence.")
    if N is None:
        N = len(x)

    v = empty((len(x), N), dtype=promote_types(x.dtype, int))
    tmp = v[:, ::-1] if not increasing else v

    if N > 0:
        tmp[:, 0] = 1
    if N > 1:
        tmp[:, 1:] = x[:, None]
        multiply.accumulate(tmp[:, 1:], out=tmp[:, 1:], axis=1)

    return v
```

calls

uses

C code

```c
PyUFunc_Accumulate(PyUFuncObject *ufunc, PyArrayObject *arr, PyArrayObject *out,
                   int axis, int otype)
{
    PyArrayObject *op[2];
    PyArray_Descr *op_dtypes[2] = {NULL, NULL};
    int op_axes_arrays[2][NPY_MAXDIMS];
    int *op_axes[2] = {op_axes_arrays[0], op_axes_arrays[1]};
    npy_uint32 op_flags[2];
    int idim, ndim, otype_final;
    int needs_api, need_outer_iterator;

    NpyIter *iter = NULL, *iter_inner = NULL;

    /* The selected inner loop */
    PyUFuncGenericFunction innerloop = NULL;
    void *innerloopdata = NULL;

    const char *ufunc_name = ufunc_get_name_cstr(ufunc);

    /* These parameters come from extobj= or from a TLS global */
    int buffersize = 0, errormask = 0;

    NPY_BEGIN_THREADS_DEF;

    NPY_UF_DBG_PRINT1("\nEvaluating ufunc %s.accumulate\n", ufunc_name);

#if 0
    printf("Doing %s.accumulate on array with dtype :  ", ufunc_name);
    PyObject_Print((PyObject *)PyArray_DESCR(arr), stdout, 0);
    printf("\n");
#endif

    if (_get_bufsize_errmask(NULL, "accumulate", &buffersize, &errormask) < 0) {
        return NULL;
    }

    /* Take a reference to out for later returning */
    Py_XINCREF(out);

    otype_final = otype;
    if (get_binary_op_function(ufunc, &otype_final,
                               &innerloop, &innerloopdata) < 0) {
        PyArray_Descr *dtype = PyArray_DescrFromType(otype);
        PyErr_Format(PyExc_ValueError,
                     "could not find a matching type for %s.accumulate, "
                     "requested type has type code '%c'",
```

```c
#if NPY_UF_DBG_TRACING
    printf("Found %s.accumulate inner loop with dtype :  ", ufunc_name);
    PyObject_Print((PyObject *)op_dtypes[0], stdout, 0);
    printf("\n");
#endif

    /* Set up the op_axes for the outer loop */
    for (idim = 0; idim < ndim; ++idim) {
        op_axes_arrays[0][idim] = idim;
        op_axes_arrays[1][idim] = idim;
    }

    /* The per-operand flags for the outer loop */
    op_flags[0] = NPY_ITER_READWRITE |
                  NPY_ITER_NO_BROADCAST |
                  NPY_ITER_ALLOCATE |
                  NPY_ITER_NO_SUBTYPE;
    op_flags[1] = NPY_ITER_READONLY;

    op[0] = out;
    op[1] = arr;

    need_outer_iterator = (ndim > 1);
    /* We can't buffer, so must do UPDATEIFCOPY */
    if (!PyArray_ISALIGNED(arr) || (out && !PyArray_ISALIGNED(out)) ||
            !PyArray_EquivTypes(op_dtypes[0], PyArray_DESCR(arr)) ||
            (out &&
             !PyArray_EquivTypes(op_dtypes[0], PyArray_DESCR(out)))) {
        need_outer_iterator = 1;
    }
    /* If input and output overlap in memory, use iterator to figure it out */
    else if (out != NULL && solve_may_share_memory(out, arr,
                                NPY_MAY_SHARE_BOUNDS) != 0) {
        need_outer_iterator = 1;
    }

    if (need_outer_iterator) {
        int ndim_iter = 0;
        npy_uint32 flags = NPY_ITER_ZEROSIZE_OK|
                           NPY_ITER_REFS_OK|
                           NPY_ITER_COPY_IF_OVERLAP;
        PyArray_Descr **op_dtypes_param = NULL;

        /*
         * The way accumulate is set up, we can't do buffering,
         * so make a copy instead when necessary.
         */
```

```c
    /* Get the output */
    if (out == NULL) {
        if (iter) {
            op[0] = out = NpyIter_GetOperandArray(iter)[0];
            Py_INCREF(out);
        }
        else {
            PyArray_Descr *dtype = op_dtypes[0];
            Py_INCREF(dtype);
            op[0] = out = (PyArrayObject *)PyArray_NewFromDescr(
                                    &PyArray_Type, dtype,
                                    ndim, PyArray_DIMS(op[1]), NULL,
NULL,
                                    0, NULL);
            if (out == NULL) {
                goto fail;
            }
        }
    }

    /*
     * If the reduction axis has size zero, either return the
reduction
     * unit for UFUNC_REDUCE, or return the zero-sized output array
     * for UFUNC_ACCUMULATE.
     */
    if (PyArray_DIM(op[1], axis) == 0) {
        goto finish;
    }
    else if (PyArray_SIZE(op[0]) == 0) {
        goto finish;
    }

    if (iter && NpyIter_GetIterSize(iter) != 0) {
        char **dataptr_copy[3];
        npy_intp stride_copy[3];
        npy_intp count_m1, stride0, stride1;

        NpyIter_IterNextFunc *iternext;
        char **dataptr;

        int itemsize = op_dtypes[0]->elsize;

        /* Get the variables needed for the loop */
        iternext = NpyIter_GetIterNext(iter, NULL);
        if (iternext == NULL) {
            goto fail;
```

```c
    if (count_m1 > 0) {
        /* Turn the two items into three for the
inner loop */
        dataptr_copy[1] += stride1;
        dataptr_copy[2] += stride0;
        NPY_UF_DBG_PRINT1("iterator loop count
%d\n",
(int)count_m1);
        innerloop(dataptr_copy, &count_m1,
                  stride_copy, innerloopdata);
    }
    } while (iternext(iter));

    NPY_END_THREADS;
}
else if (iter == NULL) {
    char *dataptr_copy[3];
    npy_intp stride_copy[3];

    int itemsize = op_dtypes[0]->elsize;

    /* Execute the loop with no iterators */
    npy_intp count = PyArray_DIM(op[1], axis);
    npy_intp stride0 = 0, stride1 =
PyArray_STRIDE(op[1], axis);

    NPY_UF_DBG_PRINT("UFunc: Reduce loop with no
iterators\n");

    if (PyArray_NDIM(op[0]) != PyArray_NDIM(op[1]) ||
        !PyArray_CompareLists(PyArray_DIMS(op[0]),
                              PyArray_DIMS(op[1]),
                              PyArray_NDIM(op[0]))
{
        PyErr_SetString(PyExc_ValueError,
                "provided out is the wrong size "
                "for the reduction");
        goto fail;
    }
    stride0 = PyArray_STRIDE(op[0], axis);

    stride_copy[0] = stride0;
    stride_copy[1] = stride1;
    stride_copy[2] = stride0;

    /* Turn the two items into three for the inner loop
*/
    dataptr_copy[0] = PyArray_BYTES(op[0]);
    dataptr_copy[1] = PyArray_BYTES(op[1]);
    dataptr_copy[2] = PyArray_BYTES(op[0]);

    /*
```

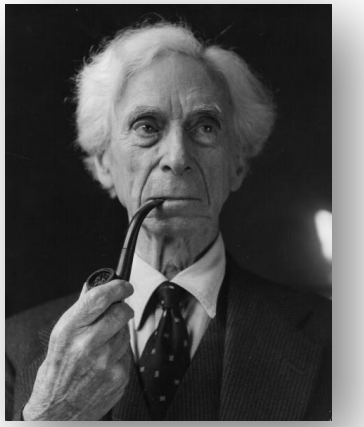
```c
@TYPE@_@kind@(char **args, npy_intp *dimensions, npy_intp *steps, void *NPY_UNUSED(func))
{
    if (IS_BINARY_REDUCE) {
#if @PW@
        @type@ * iop1 = (@type@ *)args[0];
        npy_intp n = dimensions[0];

        *iop1 @OP@= pairwise_sum_@TYPE@(args[1], n, steps[1]);
#else
        BINARY_REDUCE_LOOP(@type@) {
            io1 @OP@= *(@type@ *)ip2;
        }
        *((@type@ *)iop1) = io1;
#endif
    }
    else if (!run_binary_simd_@kind@_@TYPE@(args, dimensions, steps)) {
        BINARY_LOOP {
            const @type@ in1 = *(@type@ *)ip1;
            const @type@ in2 = *(@type@ *)ip2;
            *((@type@ *)op1) = in1 @OP@ in2;
        }
    }
}
```

C template

Julia

```
function vander(x::AbstractVector{T}) where T
    m = length(x)
    V = Matrix{T}(undef, m, m)
    for j = 1:m
        V[j,1] = one(x[j])
    end
    for i= 2:m
        for j = 1:m
            V[j,i] = x[j] * V[j,i-1]
            end
        end
    return V
end
```

The Power of Language

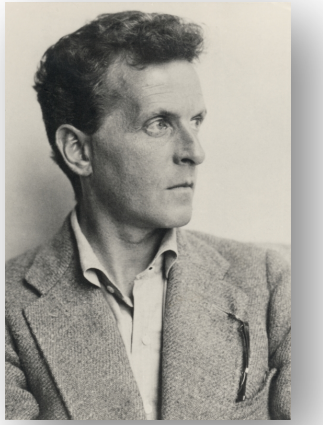


Language serves not only to express thoughts, but to make possible thoughts which could not exist without it.

Bertrand Russell

When language has been well chosen, one is astonished… (in mathematics what's possible…)

Henri Poincaré



The limits of my language mean the limits of my world.

Ludwig Wittgenstein

What does science need from a programming language?

Performance!

Workflow: roll up sleeves & performance engineer

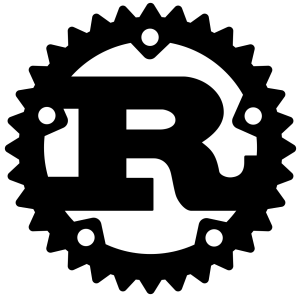
What does science need from a programming language?

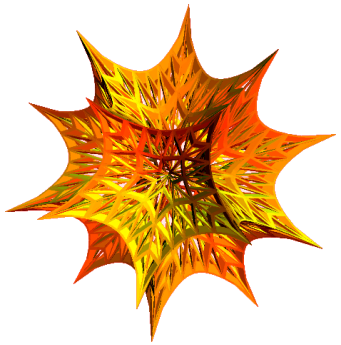
Easy to write and read !

Fast and scalable !

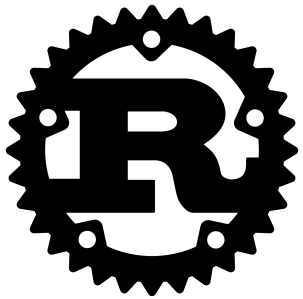
Interactive !

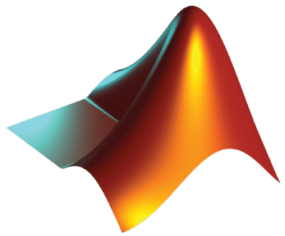
There’s a plethora of programming languages

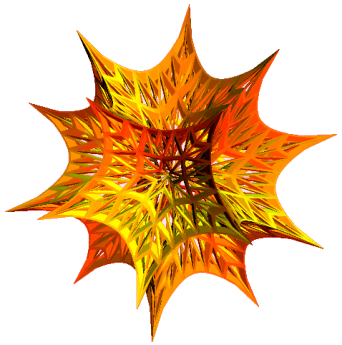




There’s a plethora of programming languages

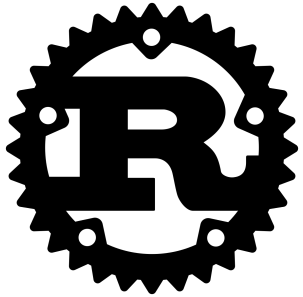






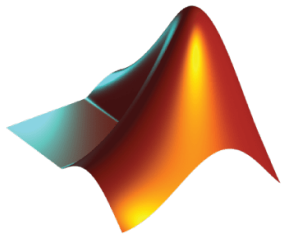
There’s a plethora of programming languages

Fortran



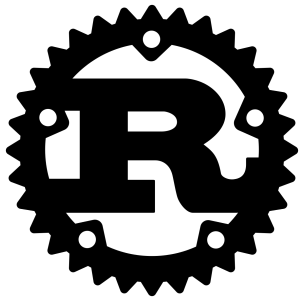
Fast

Slow(ish)



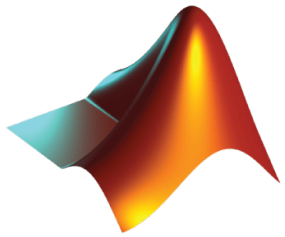
MATLAB

There’s a plethora of programming languages

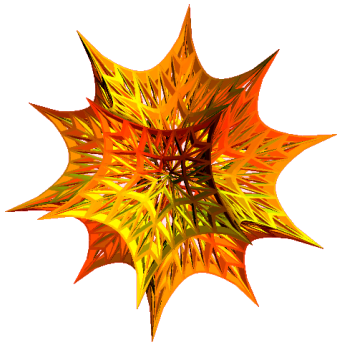


Compiled

Interpreted

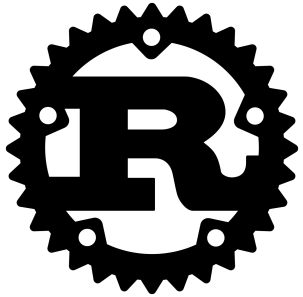


MATLAB



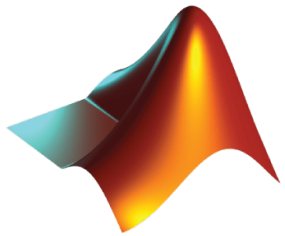
There’s a plethora of programming languages

Fortran

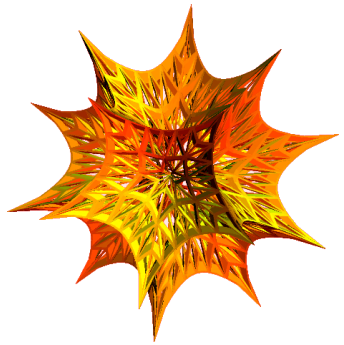


Static

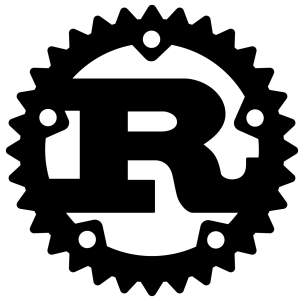
Dynamic



MATLAB

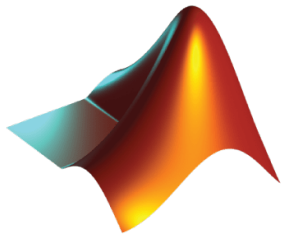


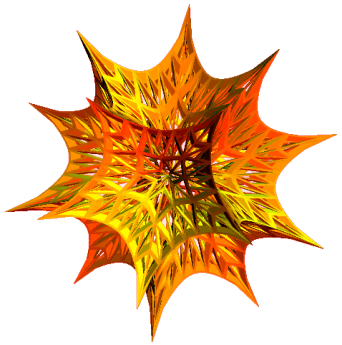
There’s a plethora of programming languages



Speed

Convenience





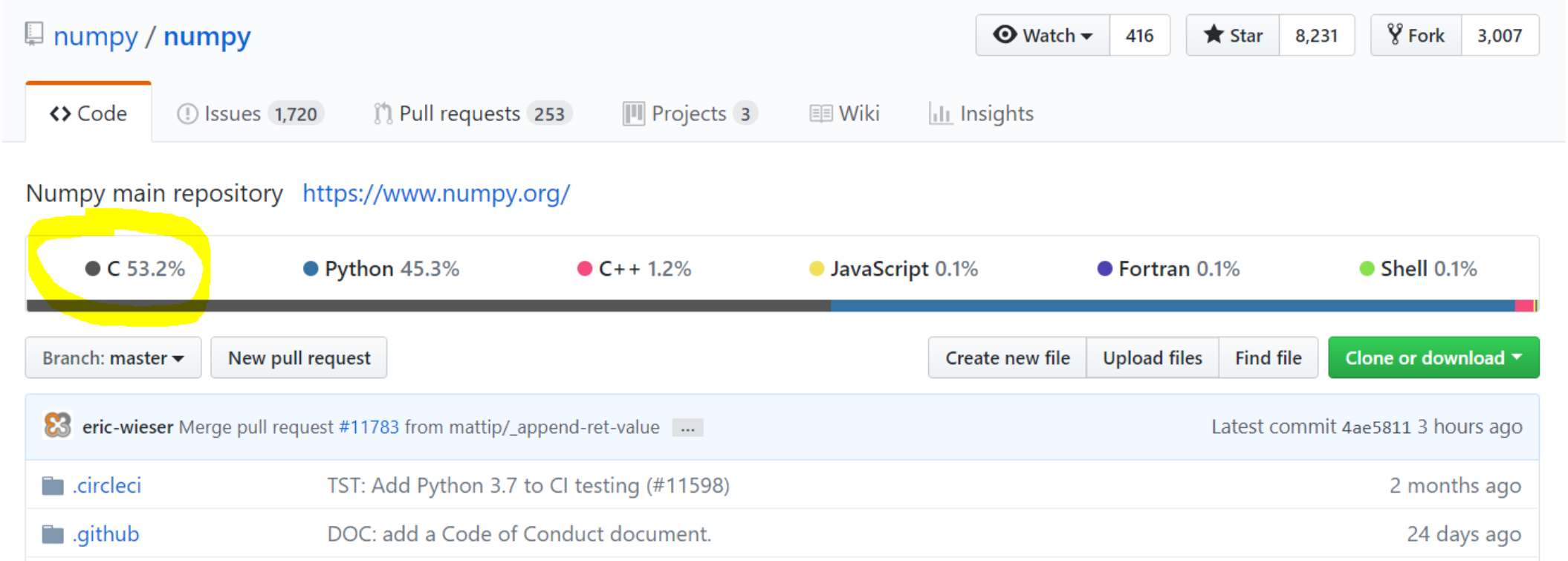
The “two language problem”

a.k.a Ousterhout's dichotomy

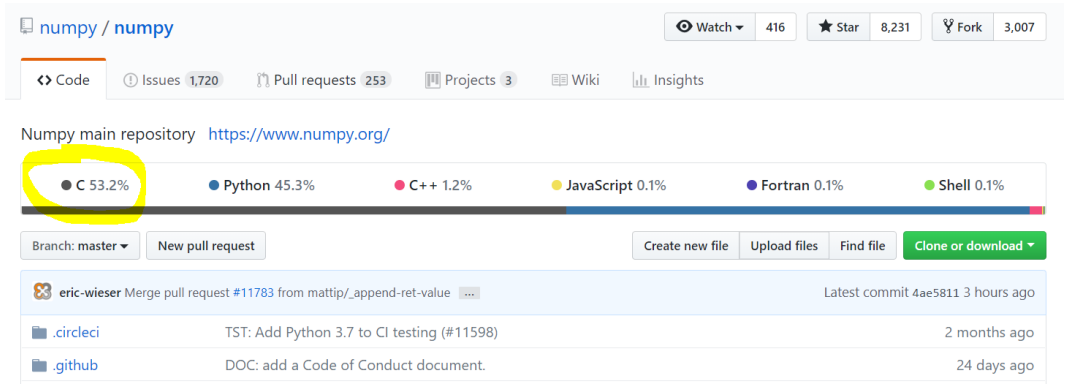
Prototype

Production

The “two language problem”

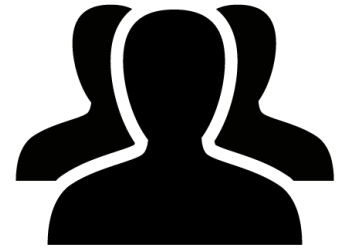


The “two language problem”



Developer

User



The “two language problem”

static	dynamic
compiled	interpreted
user types	standard types
standalone	glue

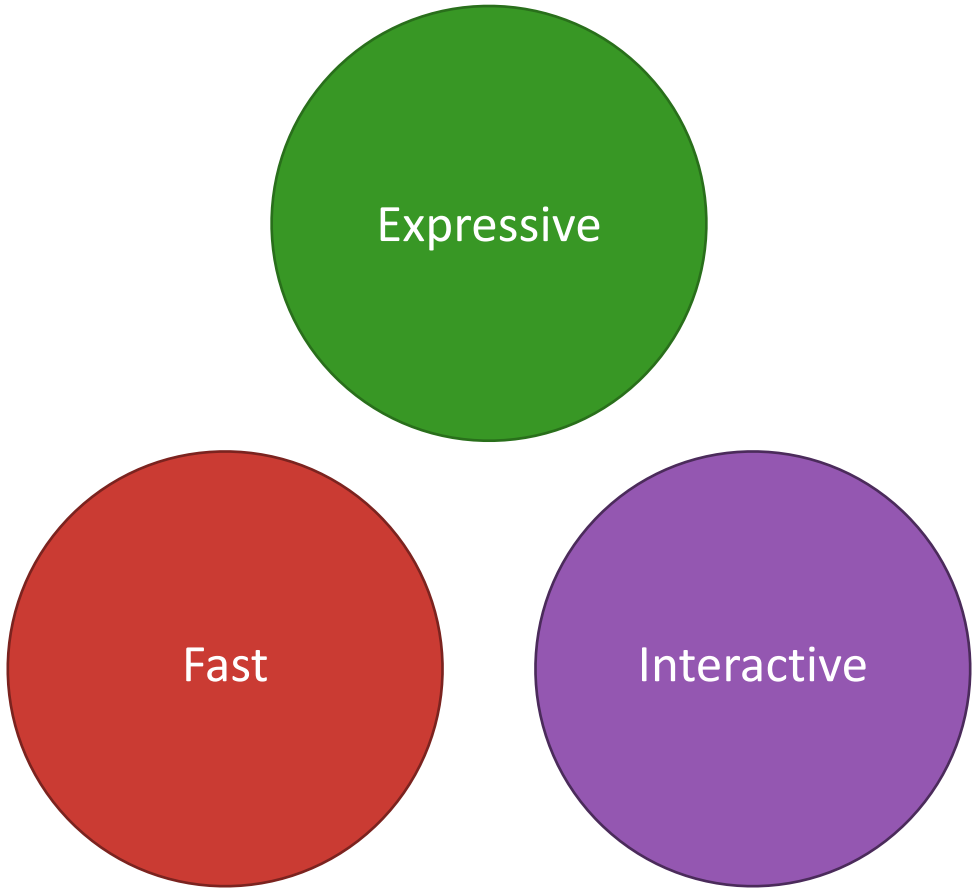
dynamic

compiled

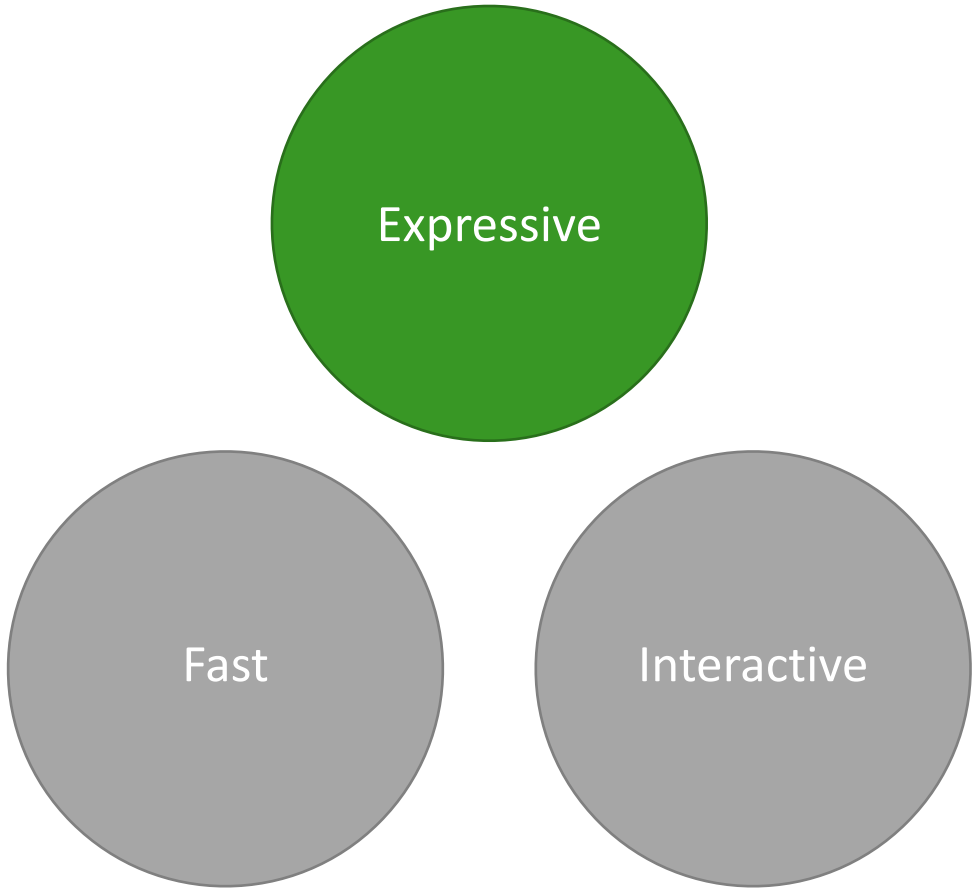
user types **and** standard types

standalone **or** glue

The julia unification



The julia unification



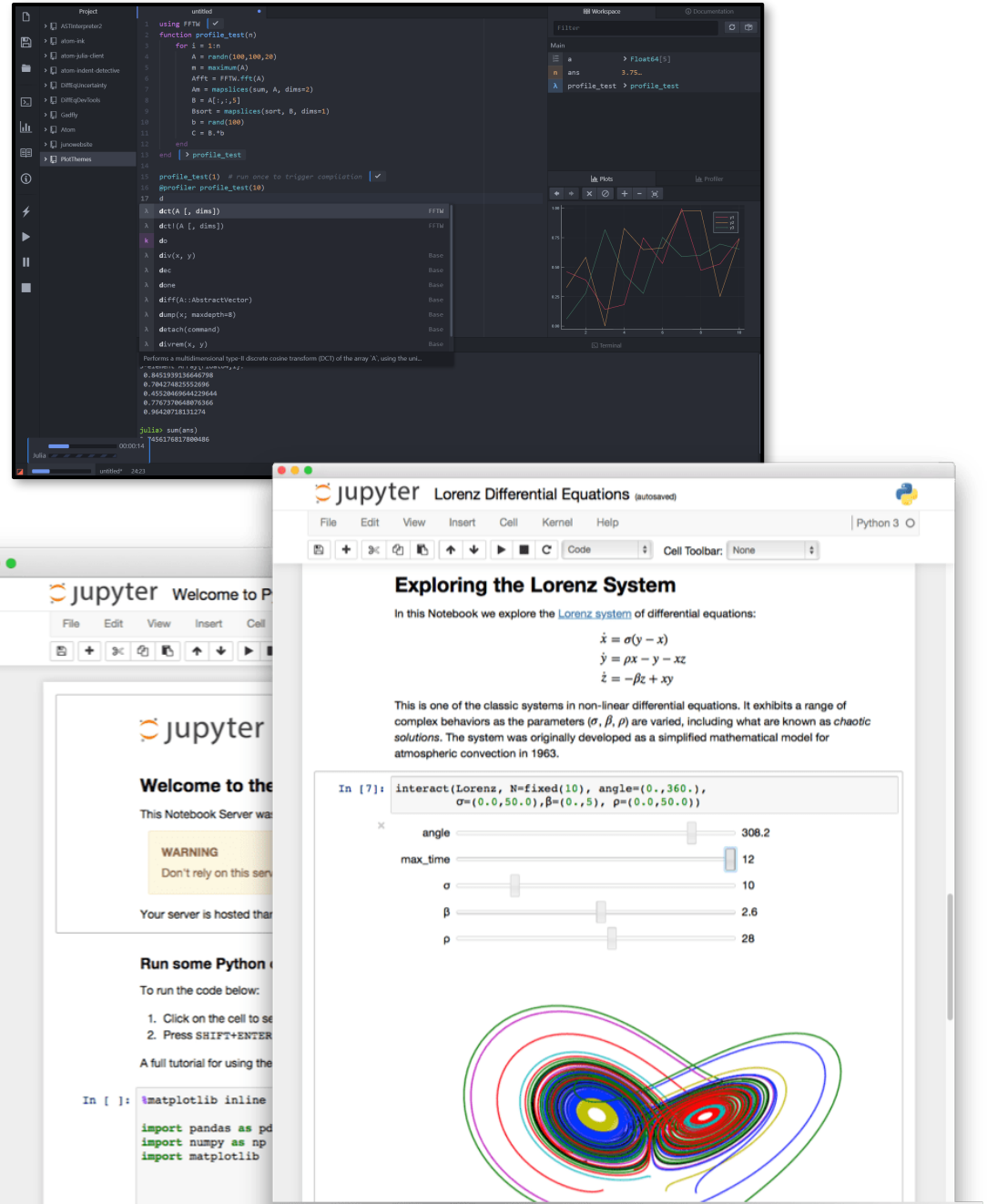

```
function babylonian(α; N = 10)
    @assert α > 0 "α must be > 0"
    t = (1+α)/2
    for i = 2:N
        t = (t + α/t)/2
    end
    t
end

babylonian(π) ≈ √π
```

The julia unification

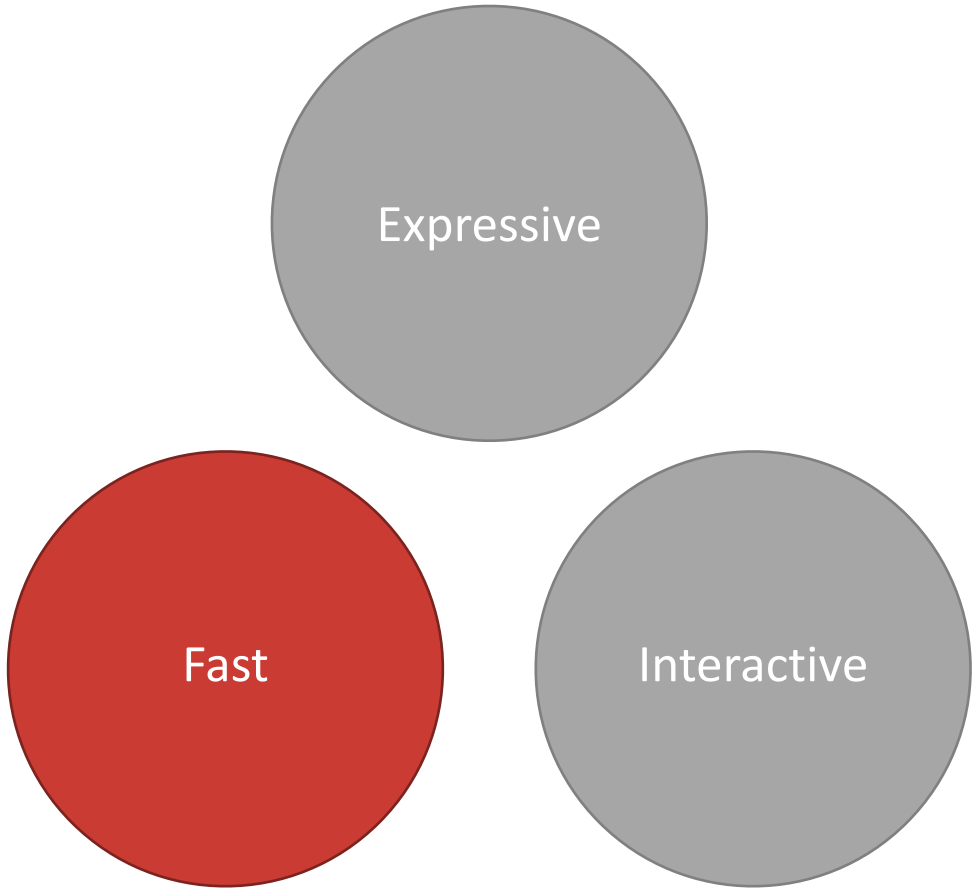
Expressive

Fast

Interactive

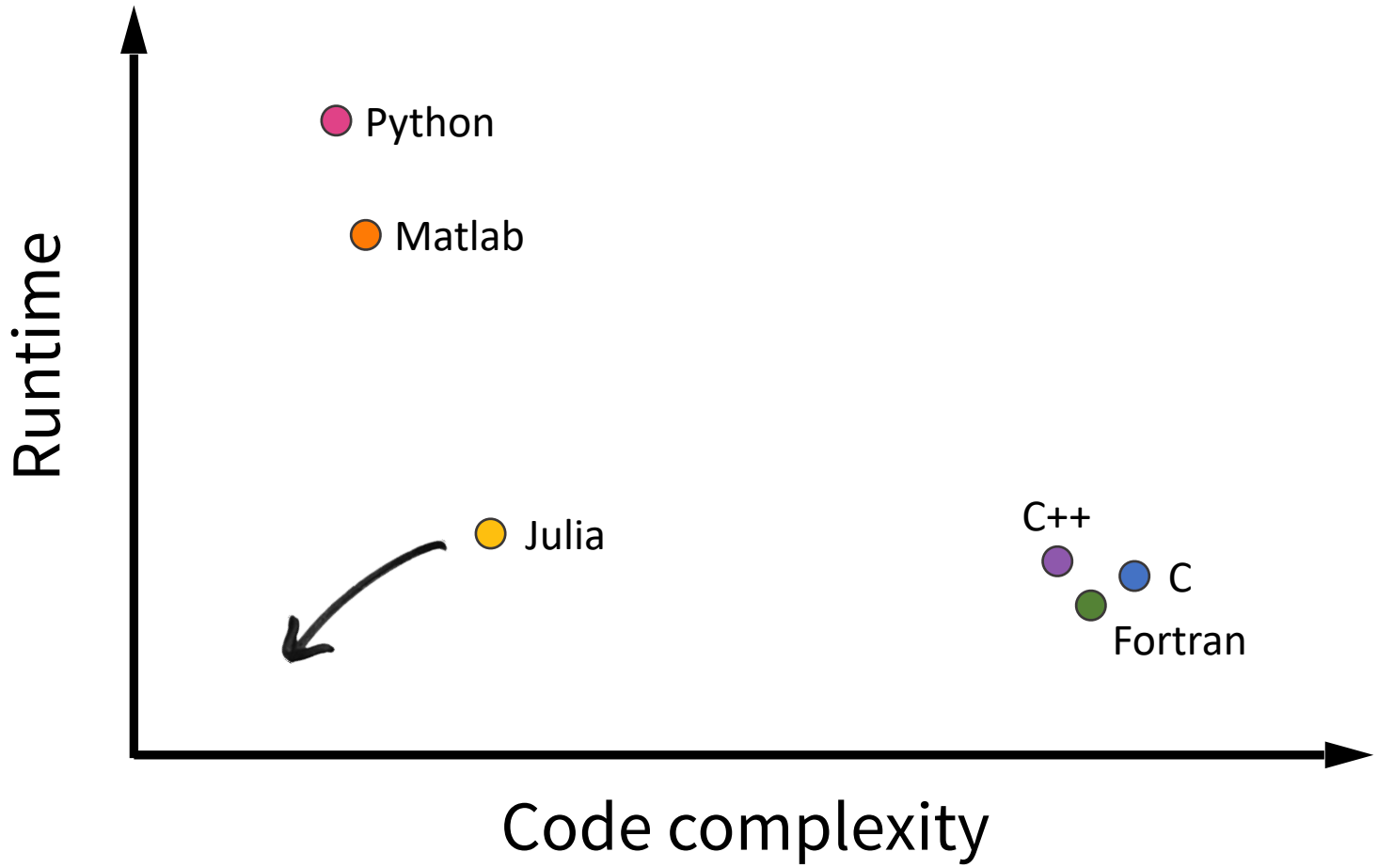


The julia unification

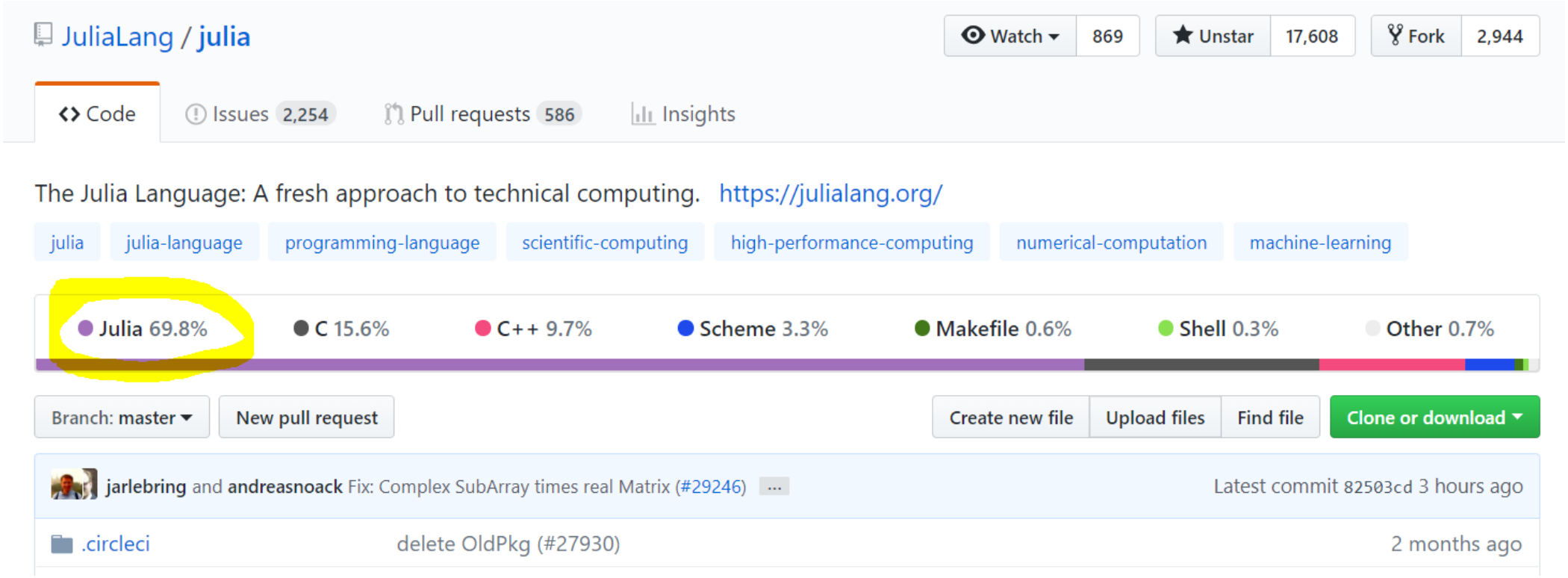



```
julia> function sumup()
           x = 0
           for i in 1:100
               x += i
           end
           x
       end
sumup (generic function with 2 methods)

julia> @code_llvm debuginfo=:none sumup()

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

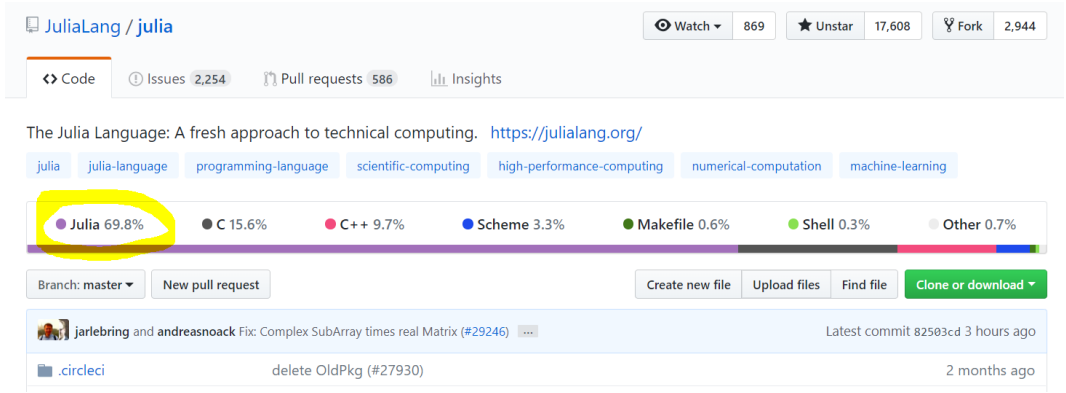
Just returns the answer!


Runtime
Python
Matlab
Julia
C++
C
Fortran
Code complexity

Free and open source

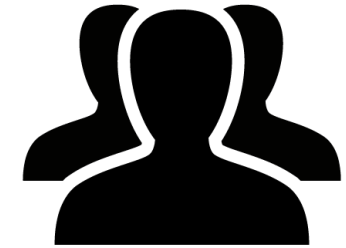


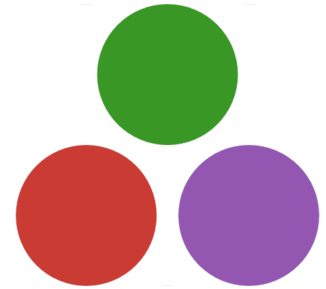
Inviting



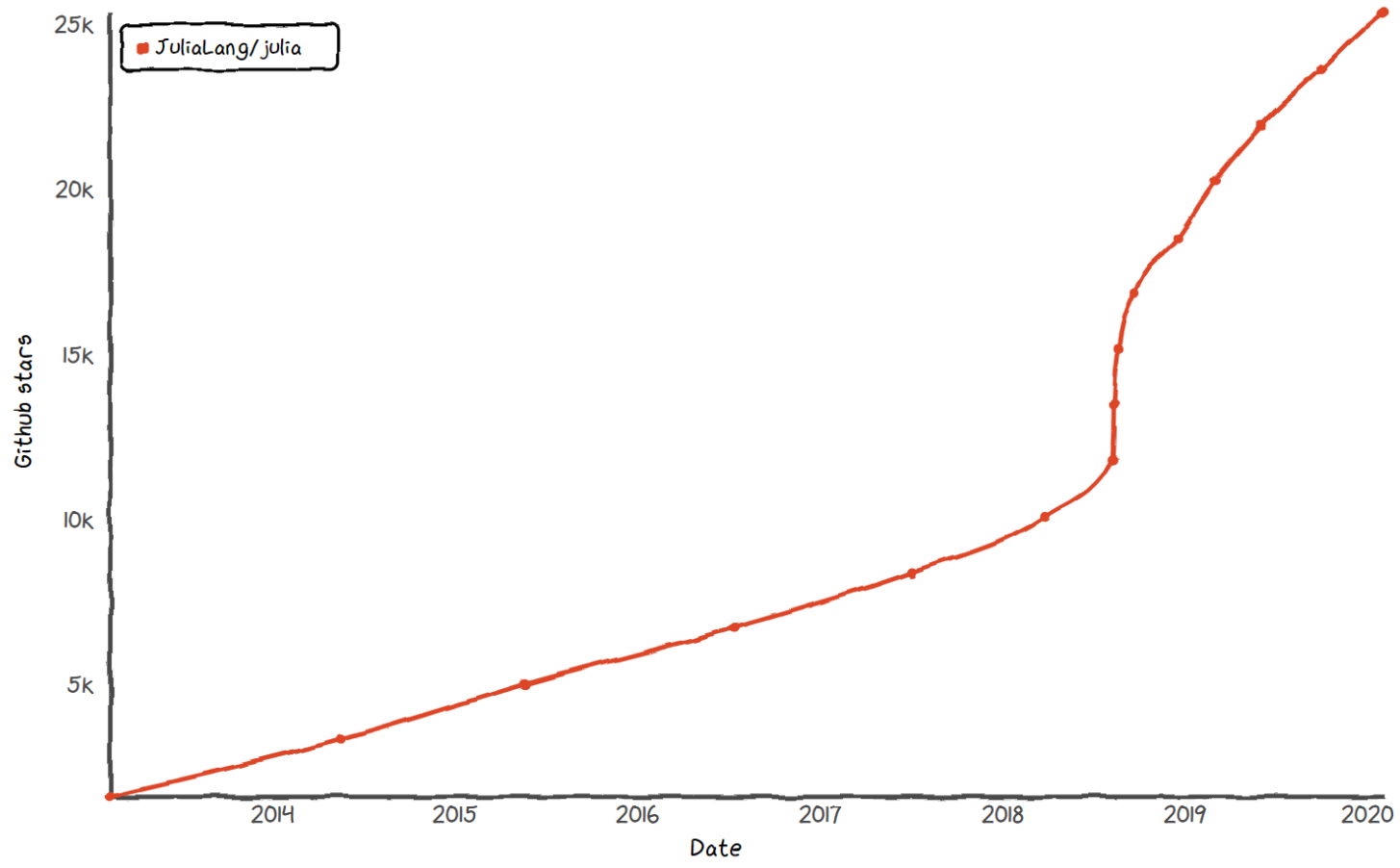
Developer

User





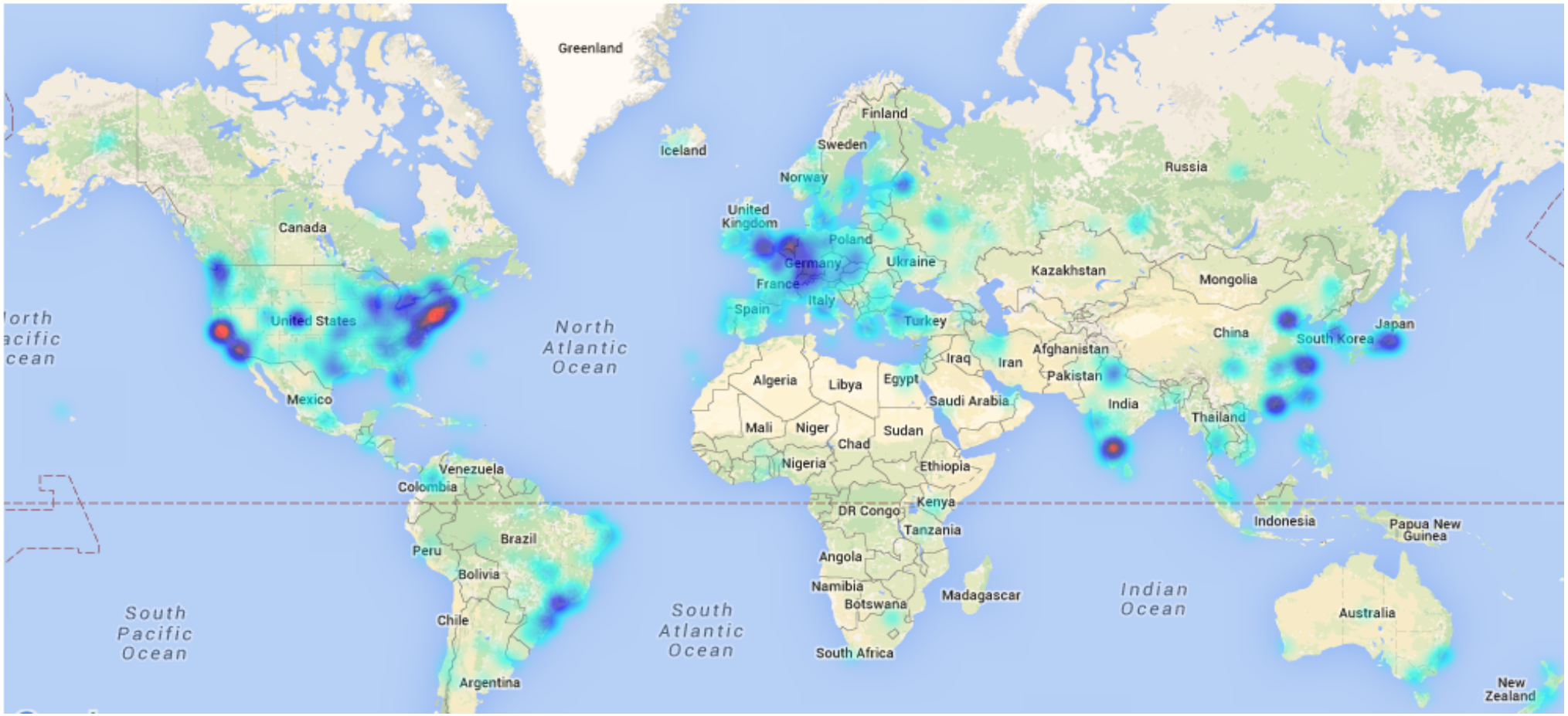
Julia GitHub stars



* Base language, does not include packages

A global community

More than 25 Million downloads, >5000 packages



James H. Wilkinson Prize For Numerical Software	Forbes 30 under 30	IEEE Babbage Prize IEEE Fellow
Stefan Karpinski Viral B. Shah Jeff Bezanson (2019)	Keno Fischer (2019)	Prof. Alan Edelman (2018)
		

BROWN
UNIVERSITY

CORNELL UNIVERSITY
FOUNDED A.D. 1865

EMORY
UNIVERSITY

NORTHEASTERN UNIVERSITY
LVX VERITAS VIRTVS
FOUNDED A.D. 1898
BOSTON, MASSACHUSETTS

PSU
PENN STATE UNIVERSITY

PURDUE

SZKOŁA GŁÓWNA HANDLOWA W WARSZAWIE

Stanford
University

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
POR MI RAZA HABLARA EL ESPIRITU

THE UNIVERSITY OF EDINBURGH

UNIVERSITY
of
GLASGOW

UNIVERSITY of SOUTH FLORIDA
TRUTH WISDOM
1956

UNIVERSITE
PAUL
SABATIER
TOULOUSE III

UNIVERSITY OF CONNECTICUT
1881

Universitas leodiensis

FIDE SPLENDET ET SCIENTIA

THE UNIVERSITY OF CALIFORNIA BERKELEY
LET THERE BE LIGHT
1868

Western
UNIVERSITY · CANADA

ARIZONA STATE UNIVERSITY
GREAT SEAL OF THE STATE OF ARIZONA
DITAT DEUS
1912
1885

دانشگاه آزاد اسلامی

MŰEGYETEM 1782

ILLINOIS STATE UNIVERSITY
18 57
Gladly we learn and teach

SIDERE MENS EADEM MUTATO

THE CITY
UNIVERSITY
OF
NEW YORK

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

MIT

TOKYO METROPOLITAN UNIVERSITY
首都大学東京

Ucla

AGH

Let’s get started!

thp Institute for Theoretical Physics University of Cologne

bcgs Bonn-Cologne Graduate School of Physics and Astronomy