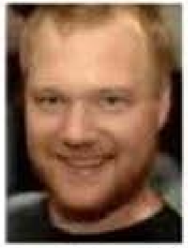
Technical computing in julia

Jiahao Chen
MIT CSAIL


Jeff Bezanson


Alan Edelman


Stefan Karpinski

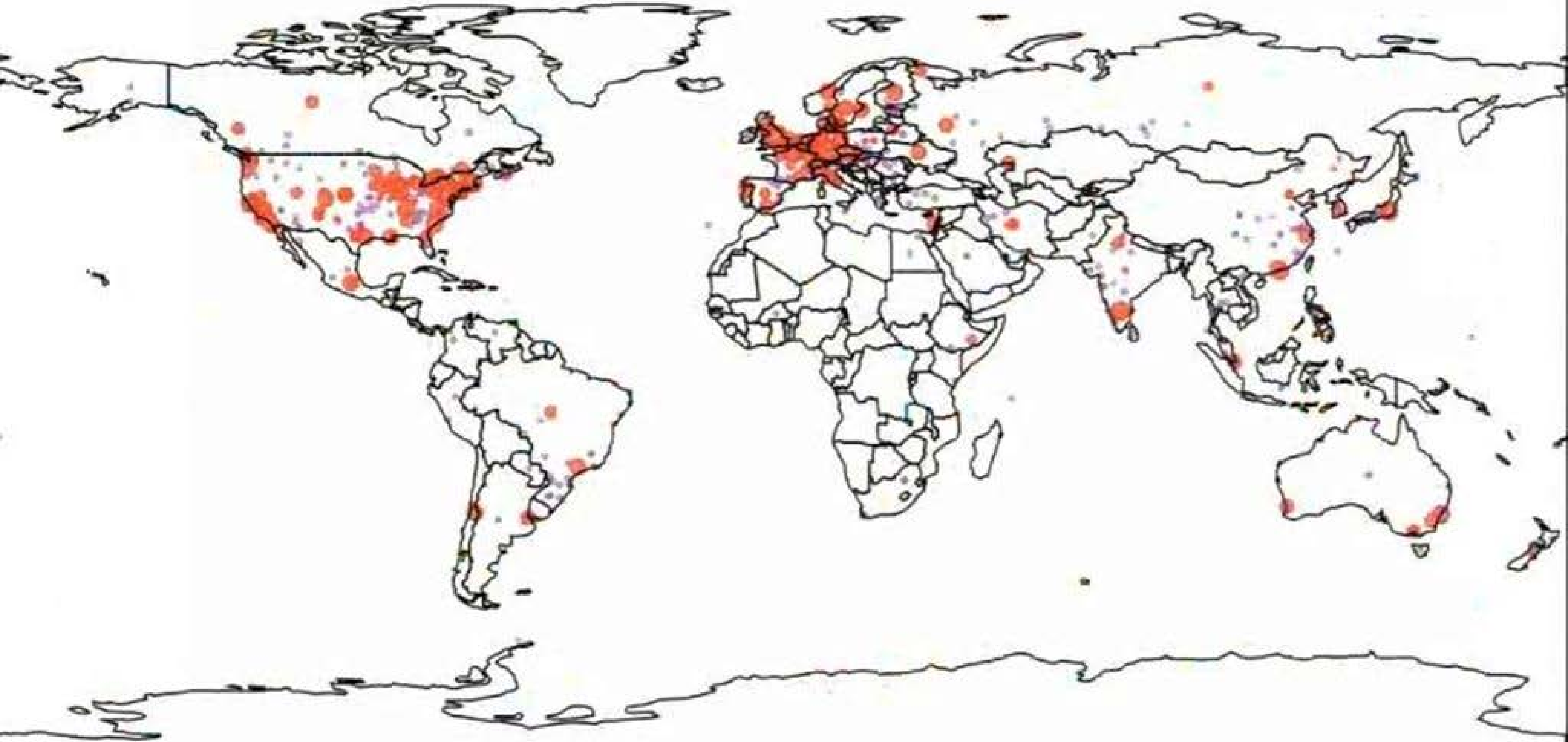

Viral B. Shah


Andreas Noack


Jake Bolewski

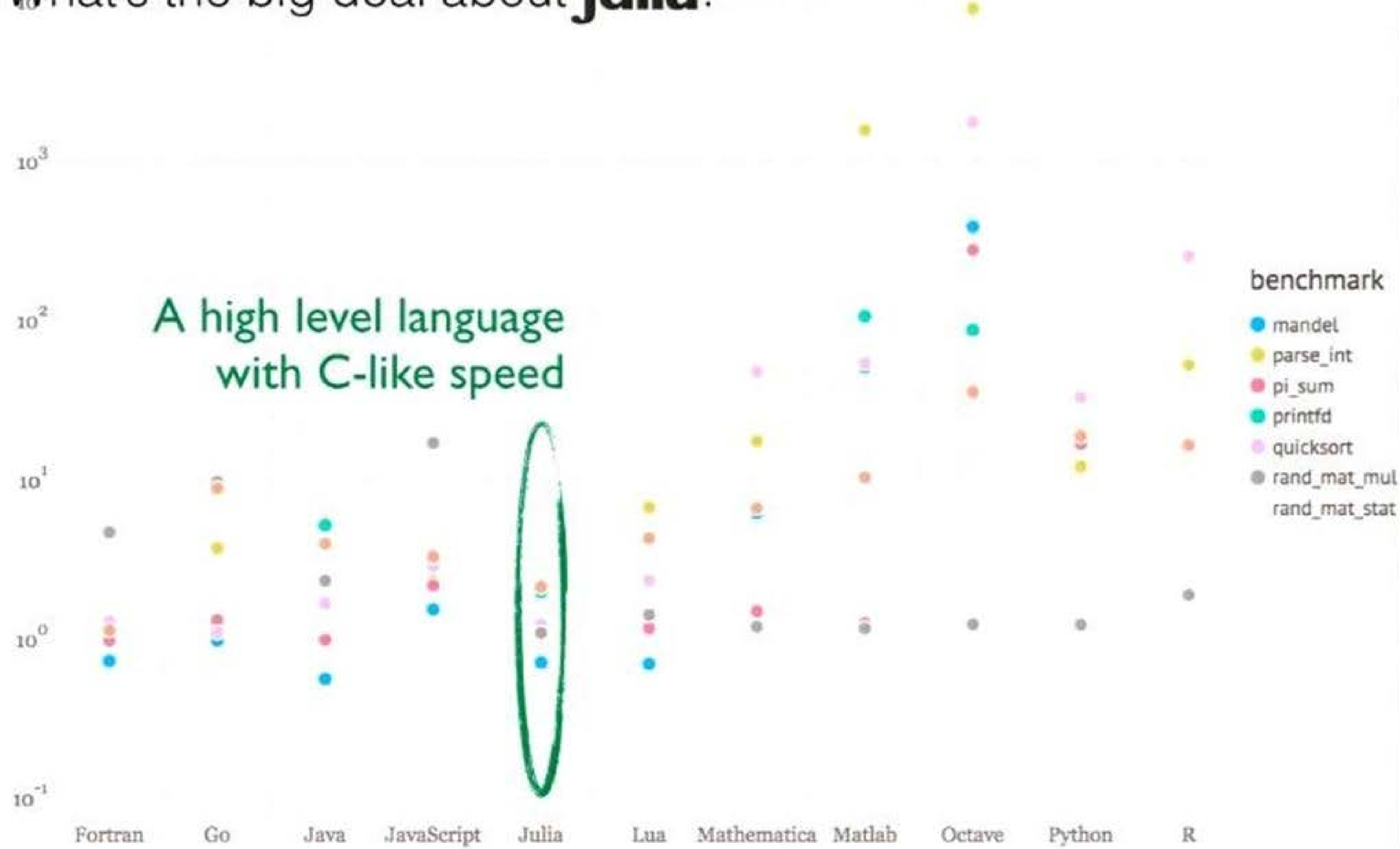
720 contributors + package writers

The world of julia



https://github.com/jiahao/julia-notebooks

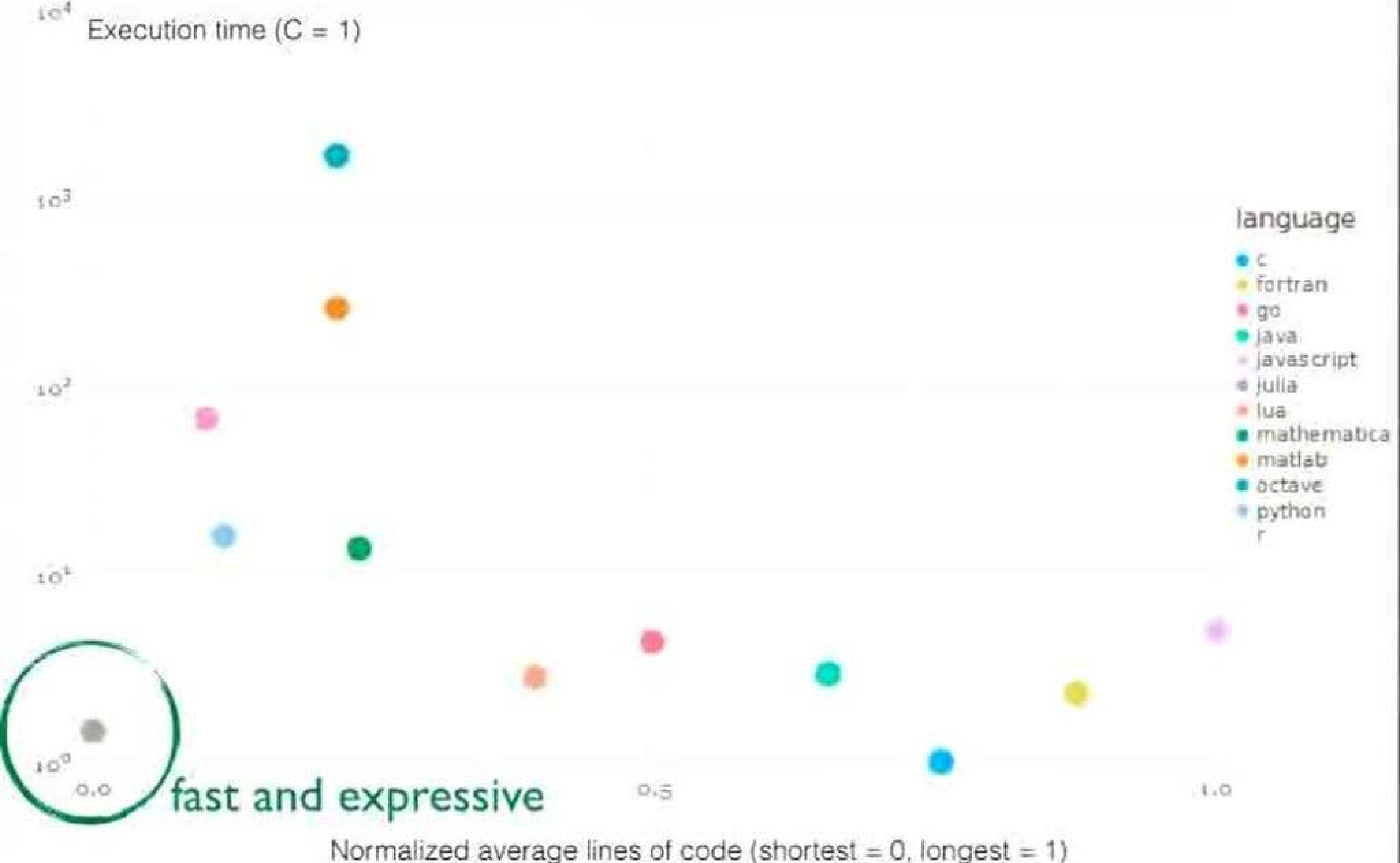
What's the big deal about julia?



julialang.org/benchmarks

Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

What's the big deal about julia?



https://groups.google.com/d/msg/julia-users/BYRAeQJuvTw/O7VK7-vp1EEJ

A simple example

```
function rrange(A, l::Integer; p::Integer=0, basis=_->qr(_)[1])
    p≥0 || error()
    n = size(A, 2)
    Ω = randn(n,l+p)
    Y = A*Ω
    Q = basis(Y)
    p==0 ? Q : Q[:,1:l]
end
```

Algorithm 4.2 computes an orthonormal matrix Q such that

$$\|(I - QQ') * A\| \leq \epsilon \quad (4.2)$$

holds with probability $\geq (1 - \min(m, n)10^{-r})$

```
ϵ=√(eps())
Q=rrange(A,k,ϵ)
E=norm((I-Q*Q')*A)
println(E, E≤ϵ?" ≤ ":"> ", ϵ) #(4.2)
```

What's the big deal about julia?

It bridges the divide between **computer science**
and **computational science**

Take home message

Types helps users express scientific computations
and helps the compiler specialize code for performance

OOP with ~~classes~~ **multi-methods**

What can I __**do with/to**__ a __**thing**__?

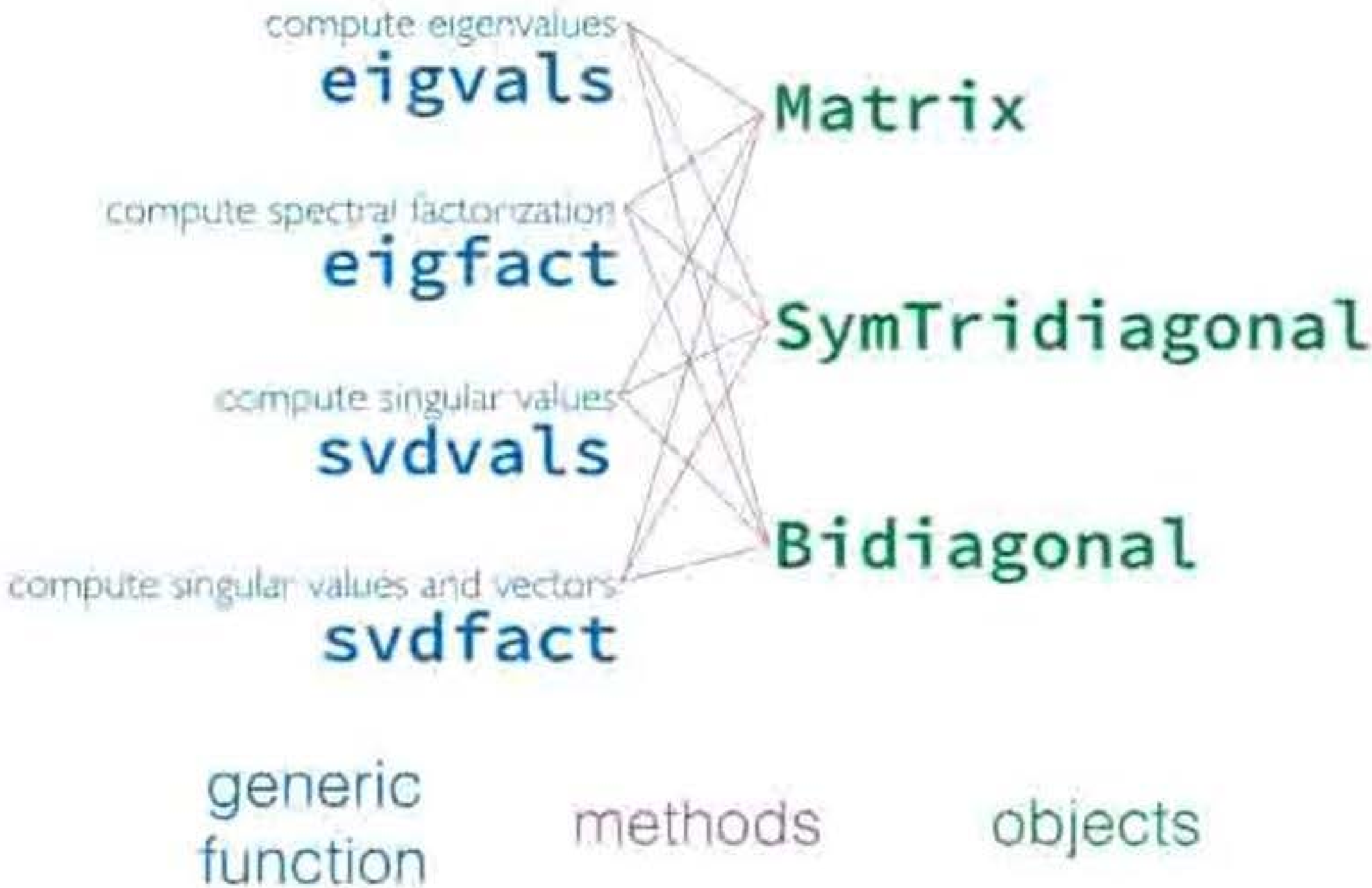
multimethods

relationships between objects and functions

generic function methods objects

Multi-methods for linear algebra

What can I **do with/to** a **thing**?



Methods can take advantage of special matrix structures

```
@which eigvals(rand(5,5))

eigvals{T}(A::Union(SubArray{T,2,A<:DenseArray{T,N},I<:
(Union(Range{Int64},Int64)...,)},DenseArray{T,2})) at linalg/factorization.jl:603

@which eigvals(UpperTriangular(rand(5,5)))

eigvals(A::Base.LinAlg.AbstractTriangular{T,S<:AbstractArray{T,2}}) at linalg/triangular.jl:882

@which eigvals(SymTridiagonal(rand(5), rand(4)))

eigvals{T}(A::Base.LinAlg.SymTridiagonal{T}) at linalg/tridiag.jl:108
```

```
eigvals!{T<:BlasReal}(A::SymTridiagonal{T}) = LAPACK.stev!('N', A.dv, A.ev)[1]
eigvals{T}(A::SymTridiagonal{T}) = (S = promote_type(Float32, typeof(zero(T)/norm(one(T)))); eigvals!(S != T ? convert(SymTridiagon
```

```
# [illegible]
# [illegible]
function stev!(job::Char, dv::Vector{$elty}, ev::Vector{$elty})
    n = length(dv)
    if length(ev) != (n-1) throw(DimensionMismatch("stev!")) end
    Zmat = similar(dv, $elty, (n, job != 'N' ? n : 0))
    work = Array($elty, max(1, 2n-2))
    info = Array(BlasInt, 1)
    ccall(($(blasfunc(stev)), liblapack), Void,
          (Ptr{UInt8}, Ptr{BlasInt}, Ptr{$elty}, Ptr{$elty}, Ptr{$elty},
           Ptr{BlasInt}, Ptr{$elty}, Ptr{BlasInt}),
          &job, &n, dv, ev, Zmat, &n, work, info)
    @lapackerror
    dv, Zmat
end
```

easy to call external C functions, e.g. CLAPACK sstev, dstev…

So how does this help us with linear algebra?

Multi-method dispatch with generic fallbacks

Matrix operations on general rings

textbook algorithm

```
In [32]: N = 8
         H = Rational{BigInt}[1//(i+j-1) for i=1:N, j=1:N]

Out[32]: 8x8 Array{Rational{BigInt},2}:
          1//1  1//2  1//3   1//4   1//5   1//6   1//7   1//8
          1//2  1//3  1//4   1//5   1//6   1//7   1//8   1//9
          1//3  1//4  1//5   1//6   1//7   1//8   1//9   1//10
          1//4  1//5  1//6   1//7   1//8   1//9   1//10  1//11
          1//5  1//6  1//7   1//8   1//9   1//10  1//11  1//12
          1//6  1//7  1//8   1//9   1//10  1//11  1//12  1//13
          1//7  1//8  1//9   1//10  1//11  1//12  1//13  1//14
          1//8  1//9  1//10  1//11  1//12  1//13  1//14  1//15

In [38]: Hf = Float64[1//(i+j-1) for i=1:N, j=1:N];

In [48]: norm(inv(Hf) - float64(float(inv(H))))

Out[48]: 117.6643671248577

In [49]: @which inv(H)

Out[49]: inv(A::Array{T,2}) at linalg/dense.jl:294

In [51]: @which lufact(H)

Out[51]: lufact{T}(A::AbstractArray{T,2}) at linalg/lu.jl:65
```

So how does this help us with linear algebra?

Multi-method dispatch with generic fallbacks

Matrix operations on general rings

In [1]:
```
using Quaternions
```

In [5]:
```
N=5
A=fill(Quaternion([rand(1:10)//rand(1:100) for i=1:4]...), 4, 4)
```

Out[5]:
```
4x4 Array{Quaternion{Rational{Int64}},2}:
 2//33 + 8//29im + 7//80jm + 4//39km  …  2//33 + 8//29im + 7//80jm + 4//39km
 2//33 + 8//29im + 7//80jm + 4//39km     2//33 + 8//29im + 7//80jm + 4//39km
 2//33 + 8//29im + 7//80jm + 4//39km     2//33 + 8//29im + 7//80jm + 4//39km
 2//33 + 8//29im + 7//80jm + 4//39km     2//33 + 8//29im + 7//80jm + 4//39km
```

In [6]:
```
lufact(A)
```

Out[6]:
```
LU{Quaternion{Rational{Int64}},Array{Quaternion{Rational{Int64}},2}}(4x4 Array{Quate
rnion{Rational{Int64}},2}:
 2//33 + 8//29im + 7//80jm + 4//39km  …  2//33 + 8//29im + 7//80jm + 4//39km
     1//1 + 0//1im + 0//1jm + 0//1km         0//1 + 0//1im + 0//1jm + 0//1km
     1//1 + 0//1im + 0//1jm + 0//1km         0//1 + 0//1im + 0//1jm + 0//1km
     1//1 + 0//1im + 0//1jm + 0//1km         0//1 + 0//1im + 0//1jm + 0//1km,
[1,2,3,4],2)
```

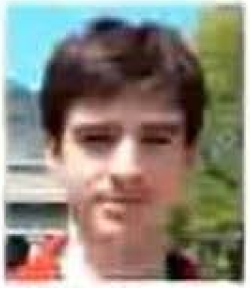
Native parallelism constructs

```
import Base: fetch, length
fetch(t::Vector) = map(fetch, t) #Vectorize fetch

#Define elementary operations on remote data
length(r1::RemoteRef)=length(fetch(r1))
+(r1::RemoteRef,r2::RemoteRef)=@spawnat r2.where fetch(r1)+fetch(r2)
*(r1::RemoteRef,r2::RemoteRef)=@spawnat r2.where fetch(r1)*fetch(r2)
function prefix!(y,.*)
    l=length(y)
    k=int(ceil(log2(l)))
    @inbounds for j=1:k, i=2^j:2^j:min(l, 2^k)              #"reduce"
        y[i]=y[i-2^(j-1)].*y[i]
    end
    @inbounds for j=(k-1):-1:1, i=3*2^(j-1):2^j:min(l, 2^k) #"broadcast"
        y[i]=y[i-2^(j-1)].*y[i]
    end
    y
end
```

```
Out[2]: prefix! (generic function with 1 method)
```

JuMP: a domain specific language



Iain Dunning Miles Lubin
MIT
Operations Research


```
#Solve a simple knapsack problem:
# max sum(p_j x_j) s.t. sum(w_j x_j) ≤ c
profit = [5, 3, 2, 7, 4]
weight = [2, 8, 4, 2, 5]
capacity = 10

using JuMP
m = Model()
@defVar(m, x[1:5], Bin)
@setObjective(m, Max, profit ⋅ x) #Maximize profit
@addConstraint(m, weight ⋅ x ≤ capacity) #Carry all
status = solve(m) #use MIP solver

println("Total profit is: ", getObjectiveValue(m))
println("Solution is:")
for i = 1:5
    print("x[$i] = ", getValue(x[i]))
    println(", p[$i]/w[$i] = ", profit[i]/weight[i])
end
```

```
Total profit is: 16.0
Solution is:
x[1] = 1.0, p[1]/w[1] = 2.5
x[2] = 0.0, p[2]/w[2] = 0.375
x[3] = 0.0, p[3]/w[3] = 0.5
x[4] = 1.0, p[4]/w[4] = 3.5
x[5] = 1.0, p[5]/w[5] = 0.8
```

Jupyter Simple Parallel Demo

File Edit View Insert Cell Kernel Help

Julia 0.3.7-pre

Code Cell Toolbar: None

IP[y]: IPython Interactive Computing

The IPython Notebook

jupyter

Estimating π using Monte Carlo

A circle of radius 1 and area π can be inscribed in a square with sides of length 2. If we pick points at random inside the square, the probability that any given point lies in the circle is

$$\frac{\int_{-1}^{1} 2\sqrt{1-x^2}\,dx}{2^2} = \frac{\pi}{4}$$

In [9]:

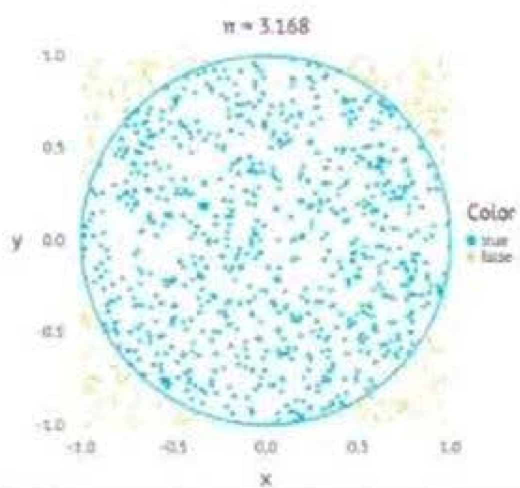
```
n = 1000
xs= convert(Array, @parallel [1-2rand() for i=1:n])
ys= convert(Array, @parallel [1-2rand() for i=1:n])
incircle = convert(Vector{Bool}, @parallel [xs[i]^2 + ys[i]^2 < 1 for i=1:n])
π = 4sum(incircle)/n
```

Out[9]: 3.168

In [22]:

```
using Gadfly
plot(layer(x=xs, y=ys, color=incircle, Geom.point,
    Theme(default_point_size =0.4mm, highlight_width=0px)),
layer(x=map(cos, 0:0.1:2π), y=map(sin, 0:0.1:2π), Geom.line(preserve_order=true)),
Guide.title("π = $π"), Coord.Cartesian(aspect_ratio=1))
```

Out[22]:



http://github.com/JuliaLang/IJulia.jl

JuliaBox beta

Run Julia from the Browser. No setup.

The Julia community is doing amazing things.

We want you in on it!

Sign in via Google

IJulia

Create IJulia Notebooks and share them.

Console

Use in-browser terminal emulator to fully control your Docker instance.

Google Drive

Collaborate with others. Sync notebooks and data via Google Drive.

Sync & Share

Setup folders to sync with remote git repositories.

In summary,

Types helps users express scientific computations and helps the compiler specialize code for performance

Other advanced features for performance: code generation, native parallel computing, ...

try it today! juliabox.org

JuliaCon - June 24-28 at MIT
juliacon.org

MS246: High-level Technical Computing with Julia
4:25 PM - 6:05 PM today in Room 254 B