Introduction to Julia

Csaba Hoch

csaba@cursorinsight.com

CURSOR INSIGHT

Agenda

1. The big picture
2. Types and extensibility
3. Julia goodies
4. Ecosystem
5. How do we use Julia?

Image source: https://en.wikipedia.org/wiki/Julia_(programming_language)#/media/File:Julia_prog_language.svg

Agenda

1. The big picture
2. Types
3. Julia goodies
4. Ecosystem
5. How do we use Julia?

Image source: https://www.jpl.nasa.gov/news/news.php?release=2013-299

Task: Implement a high performance computation

1. Statically typed languages

- C, C++, Java, etc.
- Static type system
- No math syntax,
  no extensible syntax and semantics
- Memory management (C),
  boilerplate code (Java)

Image source: https://www.pexels.com/photo/nature-animal-white-mane-45164/

2. Dynamically typed languages

- Python+NumPy, Matlab, R, etc.
- Productive development
- Slow execution → C core
- New, efficient computations → C coding

Image source: https://www.freeimages.com/photo/riders-1394941

3. Julia

- Dynamic typing
- Productive development (like Python)
- Fast execution (like C)

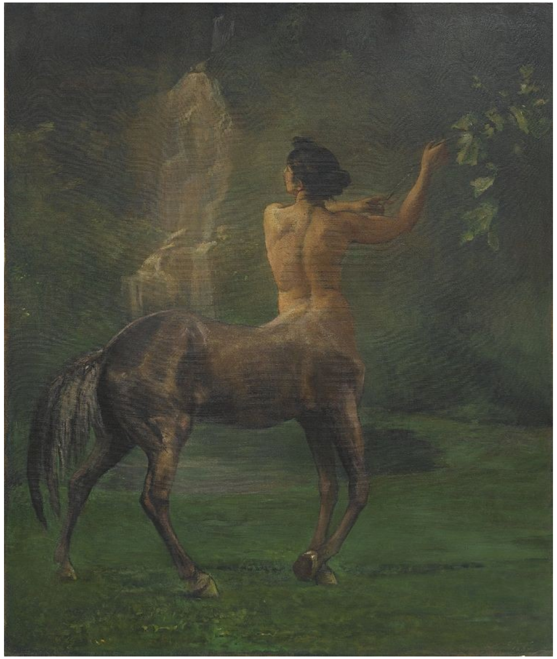


Image source: https://en.wikipedia.org/wiki/Centaur#/media/File:Brooklyn_Museum_-_Centauress_-_John_La_Farge_-_overall.jpg

What makes Julia fast and productive?

1. Just-in-time compiler (JIT)
2. Extensibility
   - Syntax: Lisp-like macros
   - Type system: Multiple dispatch
3. Designed with maths in mind

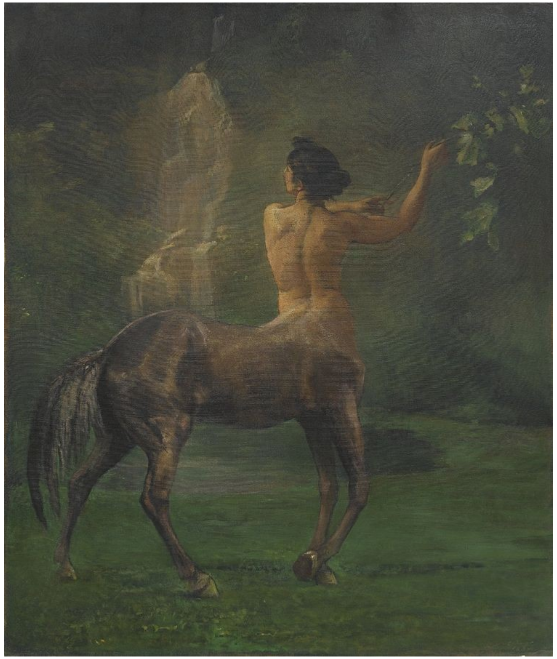


Image source: https://en.wikipedia.org/wiki/Centaur#/media/File:Brooklyn_Museum_-_Centauress_-_John_La_Farge_-_overall.jpg

Agenda

1. The big picture
2. Types and extensibility
3. Julia goodies
4. Ecosystem
5. How do we use Julia?

Image source: https://www.pexels.com/photo/alphabet-board-game-bundle-close-up-278888/

Hello World

- hello.jl:

```
println("Hello, World!")
```

- Execution:

```
$ julia hello.jl
Hello, World!
```

Type declarations

```
function hello(str)
    println("Hello, $(str)!")
end

hello("World")
```

Type declarations

```julia
function hello(str::String)
    println("Hello, $(str)!")
end
```

```julia
hello(1)
```

MethodError

```
MethodError: no method matching hello(::Int64)
Closest candidates are:
  hello(!Matched::String) at x.jl:2
```

Multiple dispatch

```
function hello(str::String)
    println("Hello string, $(str)!")
end

function hello(int::Int64)
    println("Hello int, $(int)!")
end

hello("World")
hello(42)
```

← Dispatch at runtime!

Multiple dispatch

```
julia> methods(+)
# 163 methods for generic function "+":
[1] +(x::Bool, z::Complex{Bool})
    in Base at complex.jl:277
[2] +(x::Bool, y::Bool)
    in Base at bool.jl:104
[3] +(x::Bool)
    in Base at bool.jl:101
[4] +(x::Bool, y::T) where T<:AbstractFloat
    in Base at bool.jl:112
...
```

Different files

Multiple dispatch

```
struct my
    x::Number
end
import Base.+
function +(a::my, b::my)
    return my(a.x + b.x)
end

my(1) + my(2)   →   my(3)
```

Multiple dispatch + parametric polymorphism

```
function +(a::my, b::T) where T<:Number
    return my(a.x + b)
end

function +(a::T, b::my) where T<:Number
    return my(a + b.x)
end

1 + my(2)   →   my(3)
```

Agenda

1. The big picture
2. Types and extensibility
3. Julia goodies
4. Ecosystem
5. How do we use Julia?

Image source: https://www.pexels.com/photo/plastic-animal-toy-lot-1331975/

Modern design

- Garbage collector
- Optional parameters
- Keyword parameters
- Anonymous functions ("lambdas")
- Unicode strings, multiline strings
- Dictionaries, byte arrays, etc.
- Parallel computing
- Logging, unit tests, etc.
- Docstrings
- Julia shell (docs inside)
- Package manager

Arrays

```
julia> [1, 2, 3]
3-element Array{Int64,1}:
 1
 2
 3
```

```
julia> [1 2 3; 4 5 6]
2×3 Array{Int64,2}:
 1  2  3
 4  5  6
```

Dot syntax

```
julia> [1,2,3] .+ [1,2,3]
3-element Array{Int64,1}:
 2
 4
 6
```

```
julia> max.([1 2; 3 4], [4 3; 2 1])
2×2 Array{Int64,2}:
 4  3
 3  4
```

Other goodies

```
3x
```

```
v[10:12]
```

```
v[end - 1]
```

```
i = 1_000_000
```

```
1 / 0   == Inf
1 / Inf == 0
```

```
a < b < c
```

```
rationalize(Int16, pi) == 355//113
```

```
x |> g |> f == f(g(x))
```

```
(f ∘ g)(x) == f(g(x))
```

Agenda

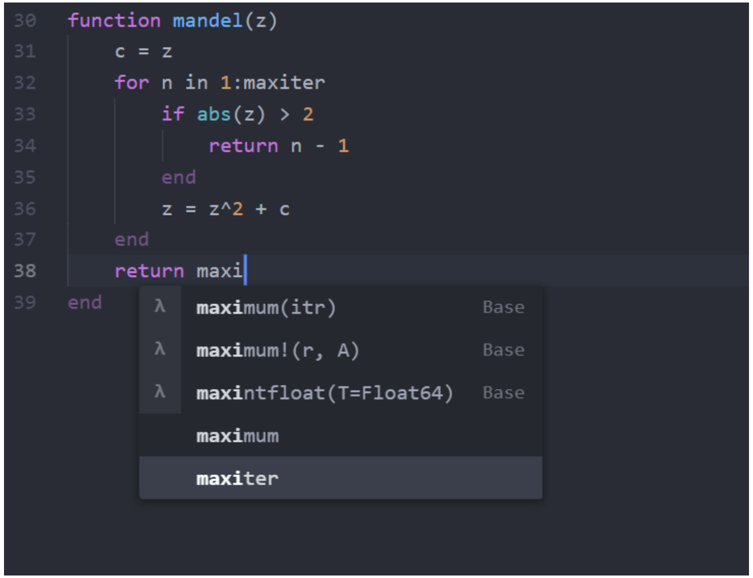
1. The big picture
2. Types and extensibility
3. Julia goodies
4. Ecosystem
5. How do we use Julia?

Julia 1.0

- Julia v1.0.0: 2018-08-08
- Packages moving to Julia 1.0

Tools for using Julia

- Juno IDE
- Jupyter notebook



```
function mandel(z)
    c = z
    for n in 1:maxiter
        if abs(z) > 2
            return n - 1
        end
        z = z^2 + c
    end
    return maxi
end
```

Image source: http://junolab.org/

Packages

- Visualization: Plots, PyPlot, Plot.ly
- Machine learning: Flux, Knet, OpenAIGym.jl
- Open dataset: MarketData.jl

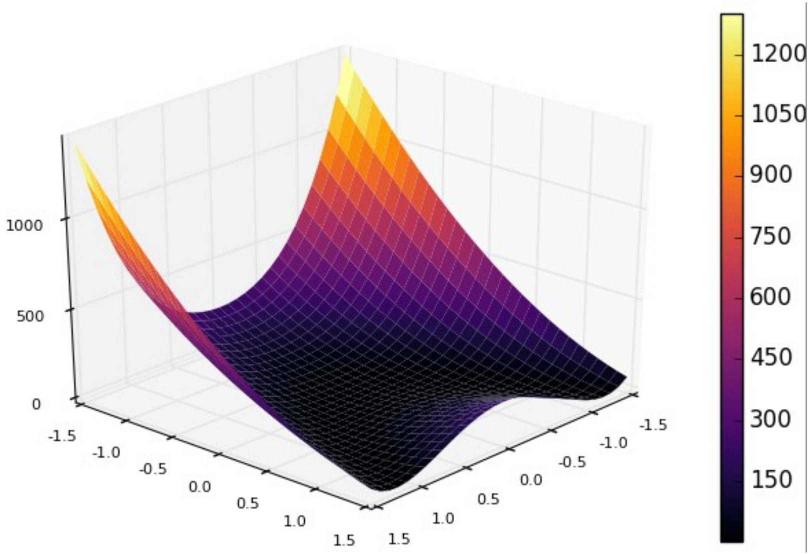


Image source: https://stackoverflow.com/questions/40650340/plotting-a-2d-function-as-surface-in-3d-space-with-plots-jl

Agenda

1. The big picture
2. Types and extensibility
3. Julia goodies
4. Ecosystem
5. How do we use Julia?

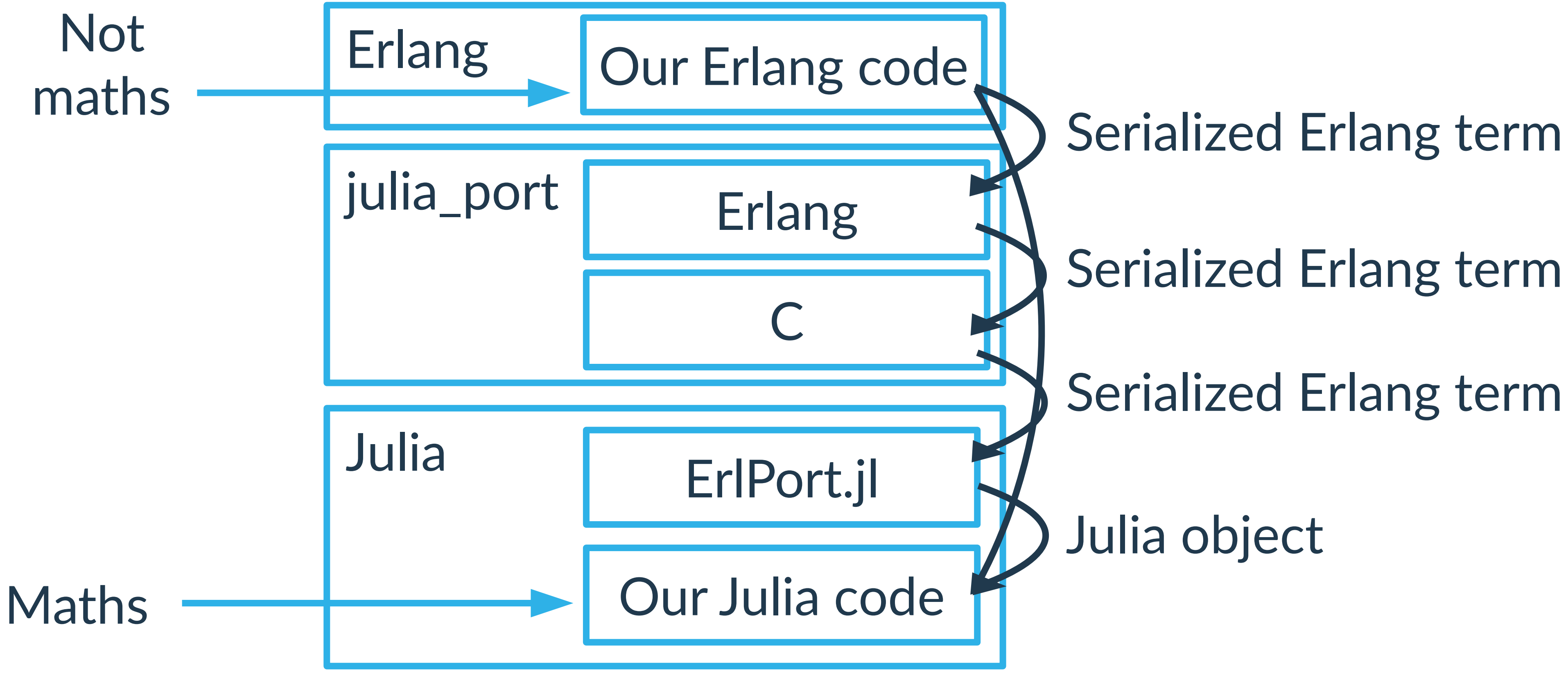
CURSOR INSIGHT

Who are we?

- Cursor Insight Ltd. (Budapest, London, Cardiff)
- Motion analysis
- Signature verification

Image source: http://cursorinsight.com/

How do we use Julia?



Not maths

Erlang

Our Erlang code

julia_port

Erlang

C

Julia

ErlPort.jl

Our Julia code

Maths

Serialized Erlang term

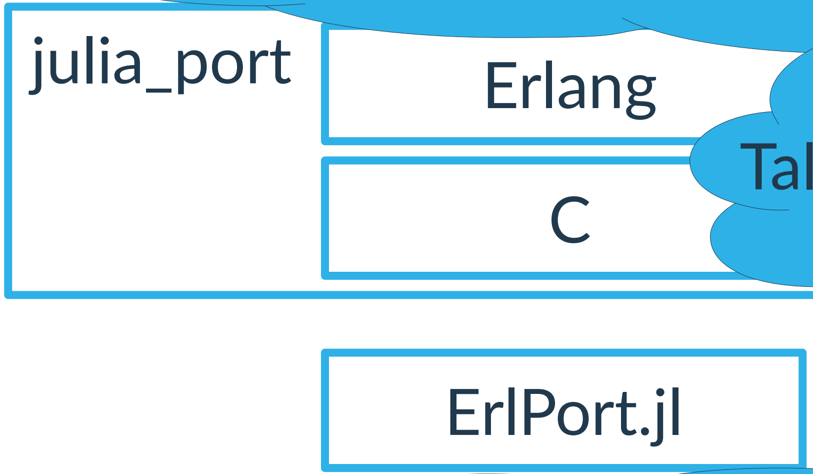
Serialized Erlang term

Serialized Erlang term

Julia object

How do we use Julia?

https://github.com/cursorinsight/julia_port
(coming soon)



julia_port

Erlang

C

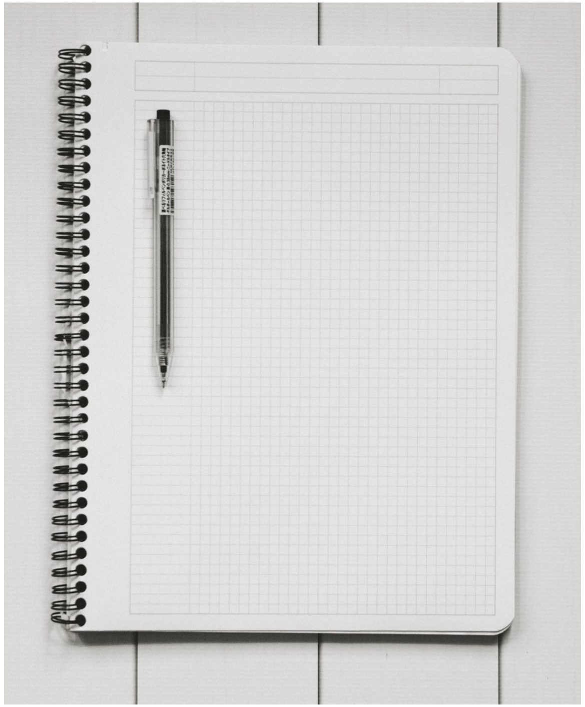
ErlPort.jl

2018-11-26 Monday:
Talk: "How we made Erlang
talk to Julia via C"

https://github.com/thorgisl/ErlPort.jl
https://github.com/cursorinsight/ErlPort.jl

Summary

1. The big picture
2. Types and extensibility
3. Julia goodies
4. Ecosystem
5. How do we use Julia?



2018-11-26 Monday:
Talk: "How we made Erlang talk to Julia via C"
(https://www.meetup.com/Budapest-Erlang-User-Group/)

Thank you!