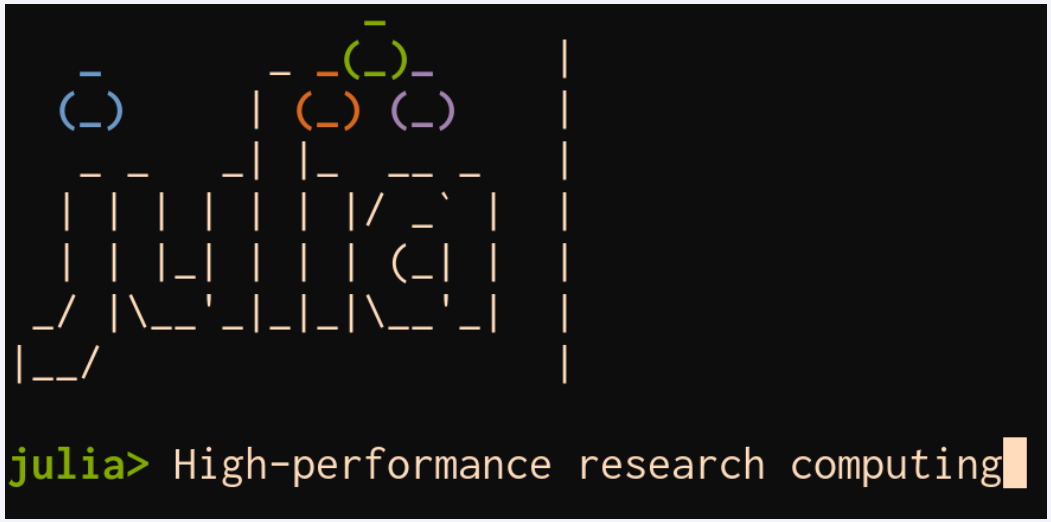


Marie-Hélène Burle

training@westgrid.ca

March 04, 2020

Brief history

Started in 2009 by Jeff Bezanson, Stefan Karpinski , Viral B. Shah , and Alan Edelman

Launched in 2012 as free and open source software

Version 1.0 released in 2018

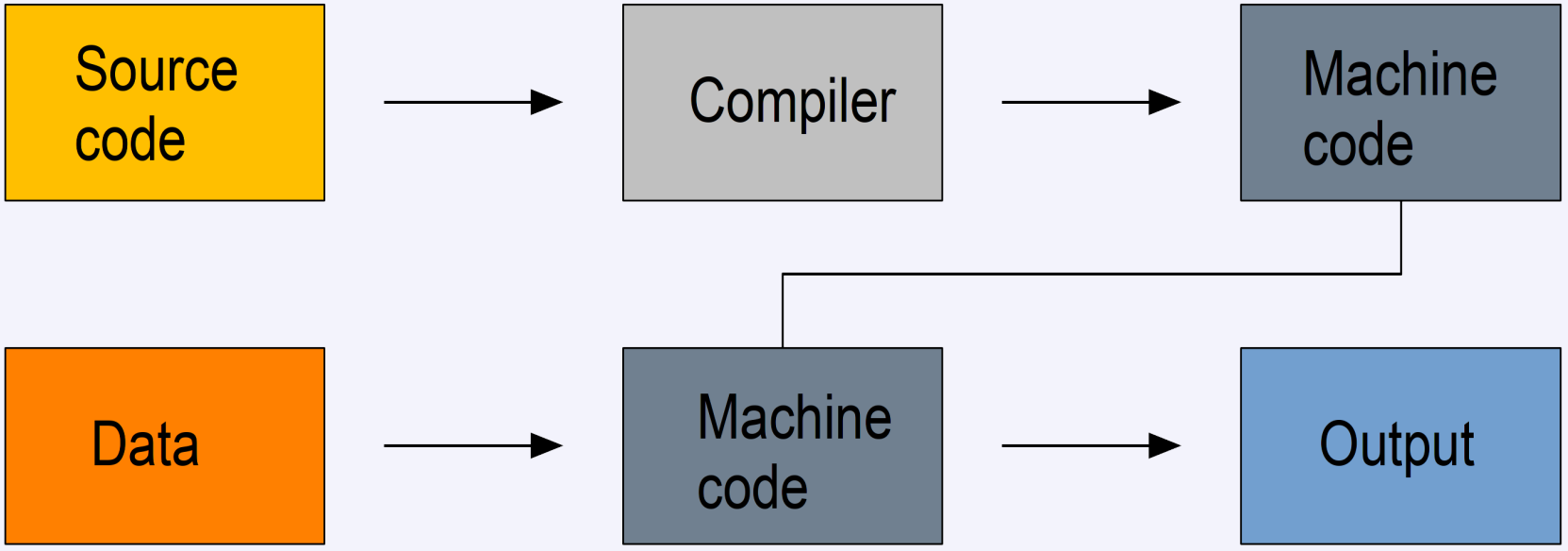
Why another language?

Computer languages mostly fall into two categories:

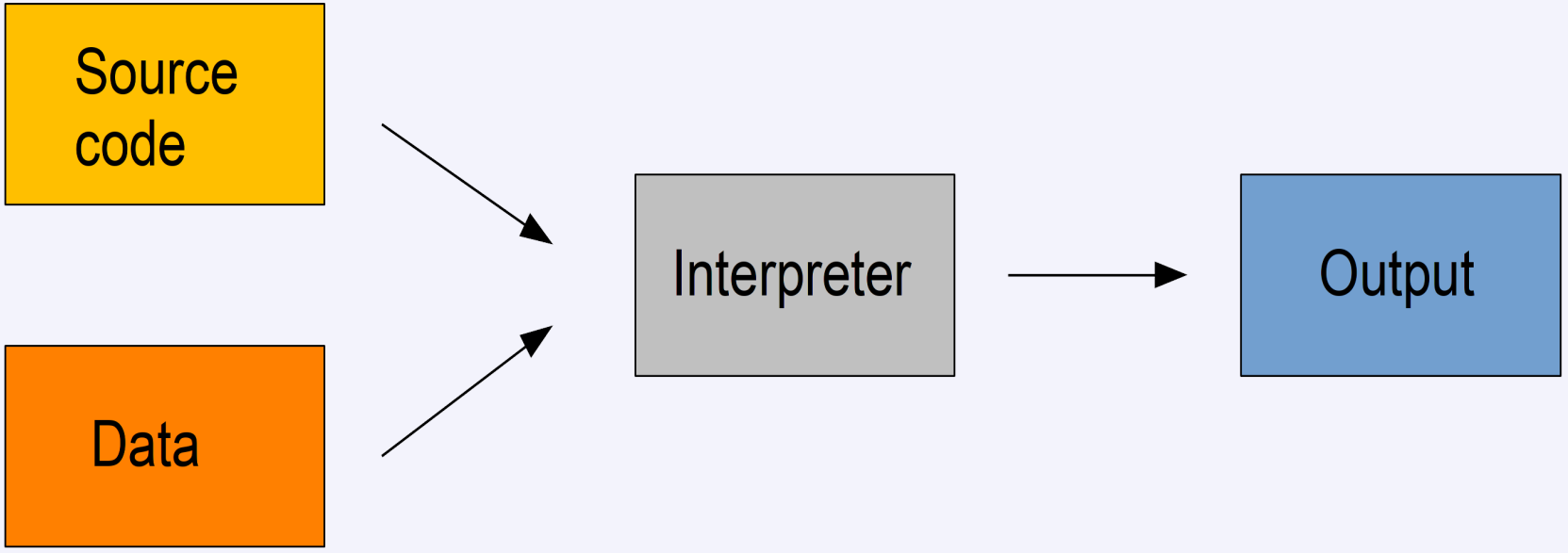
Compiled languages

Interpreted languages

Compiled languages



Interpreted languages



JIT compilation

Just-in-time compilation (JIT) based on LLVM

Source code compiled at run time

Multiple dispatch

Built-in multiple dispatch : functions apply different methods at run time based on the type of the operands

Optional type declaration

Documentation

Julia website

The official Julia manual

Online training material

The Julia YouTube channel

The Julia Wikibook

A blog aggregator for Julia

Getting help

Discourse forum

[julia] tag on Stack Overflow

Slack

#julialang hashtag on Twitter

Subreddit

Gitter channel

#julia IRC channel on Freenode

Nice ways to run Julia

Emacs

julia-emacs with julia-repl
ESS
EIN for Jupyter notebooks

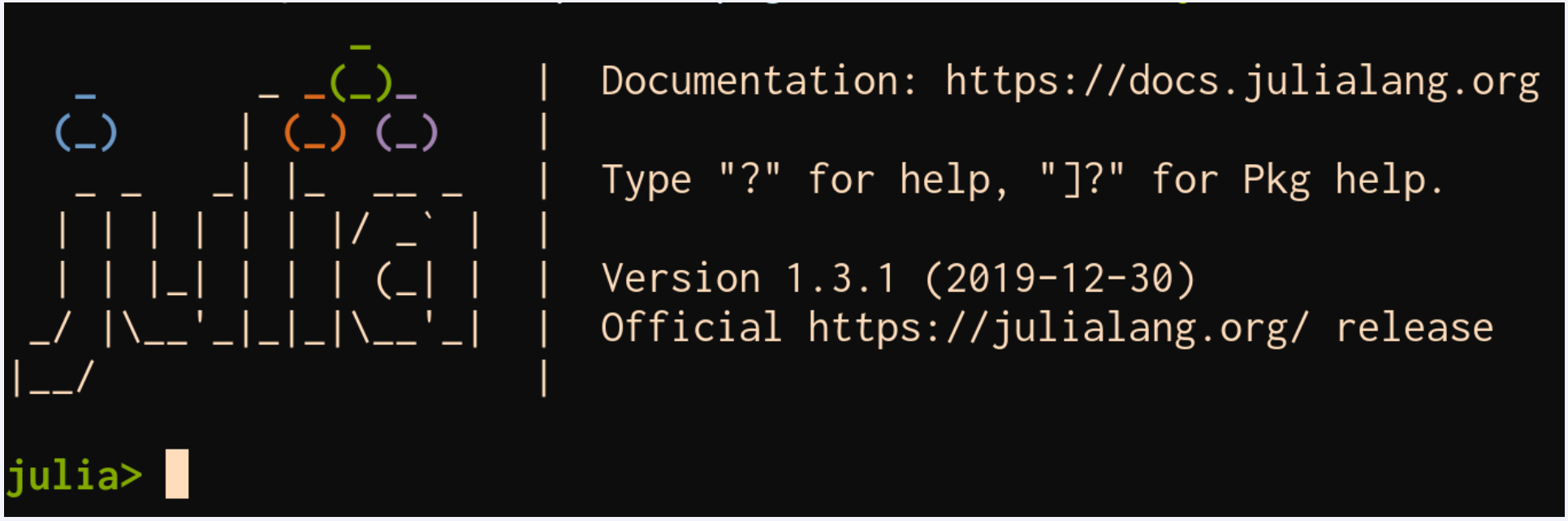
Juno

A Julia IDE built on Atom

Jupyter

Project Jupyter has a Julia kernel

REPL


Documentation: https://docs.julialang.org
Type "?" for help, "]?" for Pkg help.
Version 1.3.1 (2019-12-30)
Official https://julialang.org/ release
julia>

REPL keybindings

```
C-c             cancel
C-d             quit
C-l             clear console

C-u             kill from the start of line
C-k             kill until the end of line

C-a             go to start of line
```

Where to find packages?

Easy search engine for registered packages (all on GitHub)

Managing packages in `Pkg` mode

```
(env) pkg> add <package>          # install <package>
(env) pkg> rm <package>           # uninstall <package>
(env) pkg> up <package>           # upgrade <package>

(env) pkg> st                     # check which packages are installed
(env) pkg> up                     # upgrade all packages
```

By default, installed in `~/.julia`

Loading a package

```
> using <package>
```

Data types

```
> typeof(2)

> typeof(2.0)

> typeof("hello")

> typeof(true)
```

Indexing

Indexing starts at `1` , not `0`

```
> a = [1 2; 3 4]

> a[1, 1]

> a[1, :]
```

For loops

```
> for i in 1:10
    println(i)
end



> for i in 1:3, j = 1:2
    println(i * j)
end
```

Conditionals

```
> a = 2
> b = 2.0


> if a == b
    println("It's true")
else
    println("It's false")
end
```

Functions

```
> function addTwo(a)
    a + 2
end


> addTwo(3)


# Terse format
> addtwo = a -> a + 2
```

Plotting

Fun: plots in the command line!

```
> using UnicodePlots


> UnicodePlots.histogram(randn(1000), nbins=40)
```

This can be useful in remote sessions

Plotting

Nicer looking plots

```
> using Plots, Distributions, StatsPlots
> gr() # Using the GR framework as backend


> x = 1:10; y = rand(10, 2);
> p1 = Plots.histogram(randn(1000), nbins=40)
> p2 = plot(Normal(0, 1))
> p3 = scatter(x, y)
> p4 = plot(x, y)
```

The `Plots` site has demos

Parallel programming

Launching Julia on multiple threads

Set the environment variable:

```
$ export JULIA_NUM_THREADS=n
```

Or launch a julia session with:

```
$ JULIA_NUM_THREADS=n julia
```

See how many threads are used in a julia session:

```
> Threads.nthreads()
```

When is parallelism happening?

Non parallel code

```
> for i = 1:10
    println("Iteration $i ran on thread $(Threads.threadid())")
end
```

Parallel code

```
> Threads.@threads for i = 1:10
    println("Iteration $i ran on thread $(Threads.threadid())")
end
```

Effect on timing

Let's do a simple loop with 10,000,000 iterations

Non parallel code

```
> @time for i = 1:10000000
    i ^ i
end
```

Parallel code

```
> @time Threads.@threads for i = 1:10000000
    i ^ i
end
```

Let’s move on to the cluster

Loading the Julia module

```
# Look for available julia modules
$ module spider julia


# See modules required to load julia 1.3
$ module spider julia/1.3.0


# Load required gcc module and julia module
$ module load gcc/7.3.0 julia/1.3.0
```

Job script

```
#!/bin/bash
#SBATCH --job-name=julialoop        # job name
#SBATCH --time=00:00:30             # max walltime 30s
#SBATCH --cpus-per-task=32          # number of cores
#SBATCH --mem=100                   # max memory (in MB)
#SBATCH --output=julialoop%j.out    # output file name
#SBATCH --error=julialoop%j.err     # errors file name
```

Submit job

```
$ sbatch job_julialoop.sh
```

Check its status

```
$ sq
```

`PD` : pending

`R` : running

Results

```
Running non parallel loop on 32 cores
0.810377 seconds
Running parallel loop on 32 cores
0.093013 seconds (31.92 k allocations: 1.785 MiB)
```

89% faster


julia> Questions?