

N times faster data loading with Julia

Abel & Faruk

netherlands
eScience center

Discussion with Faruk

- “Why Julia?”
- Started looking for a project to test.
- We found Patrick Bos’ “50 times faster data loading with Python”.
- Proposal
 - Reproduce
 - Replace C++ and Julia
 - Optimize Julia code

50 times faster data loading for Pandas: no problem

Loading irregular data into Pandas using C++



Patrick Bos

Follow



Sep 3, 2018 · 10 min read



254219#72867967463,74038042345,75209415207
393216#132171266578,164996371986
658326#121627899375
684816#138997537243,149990243935,154141013182,174109928696,210371477340

4 rows = 4 keys

values = elements
3+2+1+5 = 11 elements

Dataset generation with

- row = 5, 10, 15, ..., 500 x 10 for each = 1000 tests
- random number of elements per key
- time.time()
- Run 5 times, compute average

		value
key	list_index	
254219	0	72867967463
	1	74038042345
	2	75209415207
393216	0	132171266578
	1	164996371986
658326	0	121627899375
684816	0	138997537243
	1	149990243935
	2	154141013182
	3	174109928696
	4	210371477340

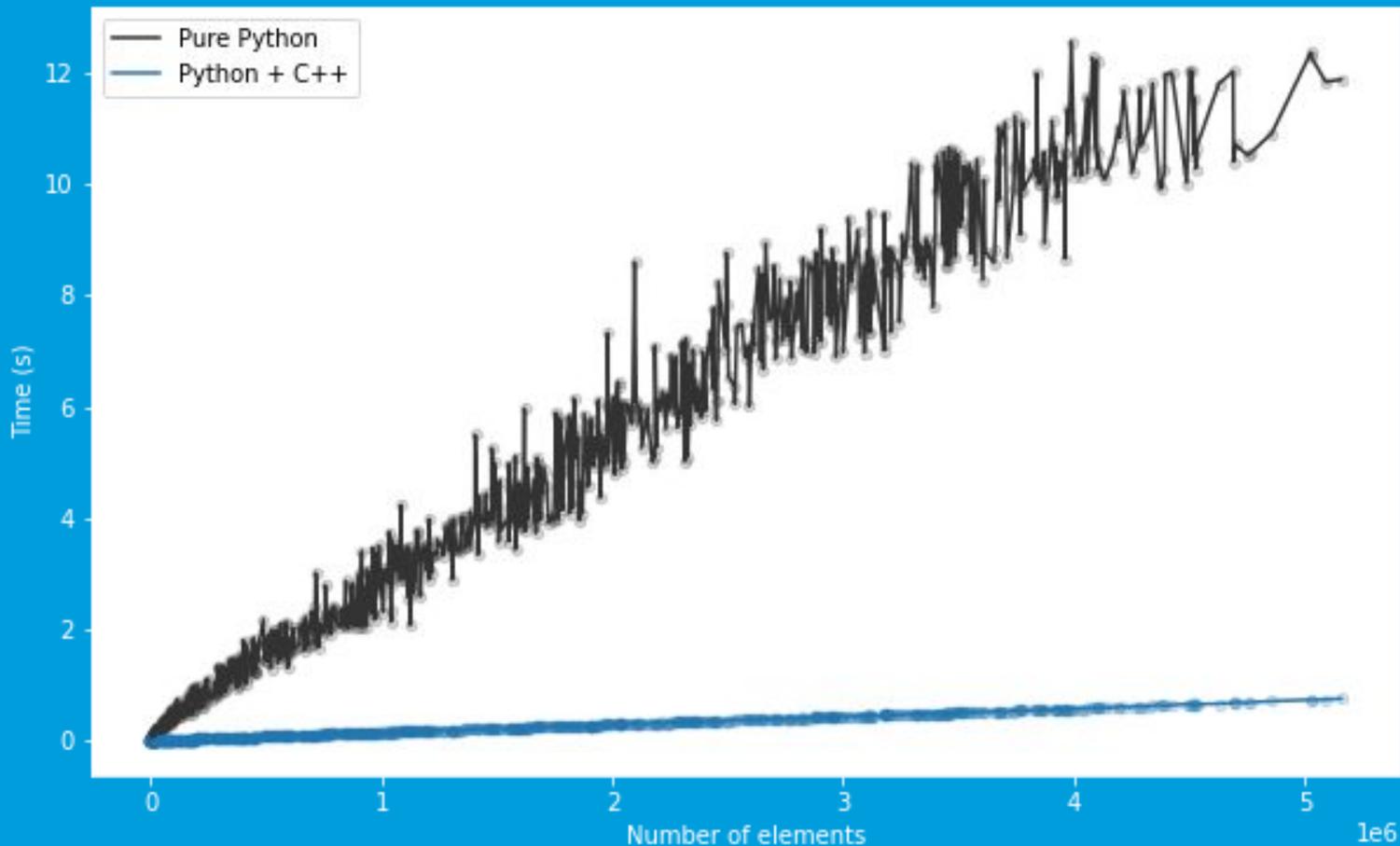
Pure Python

- Read + Split + Stack

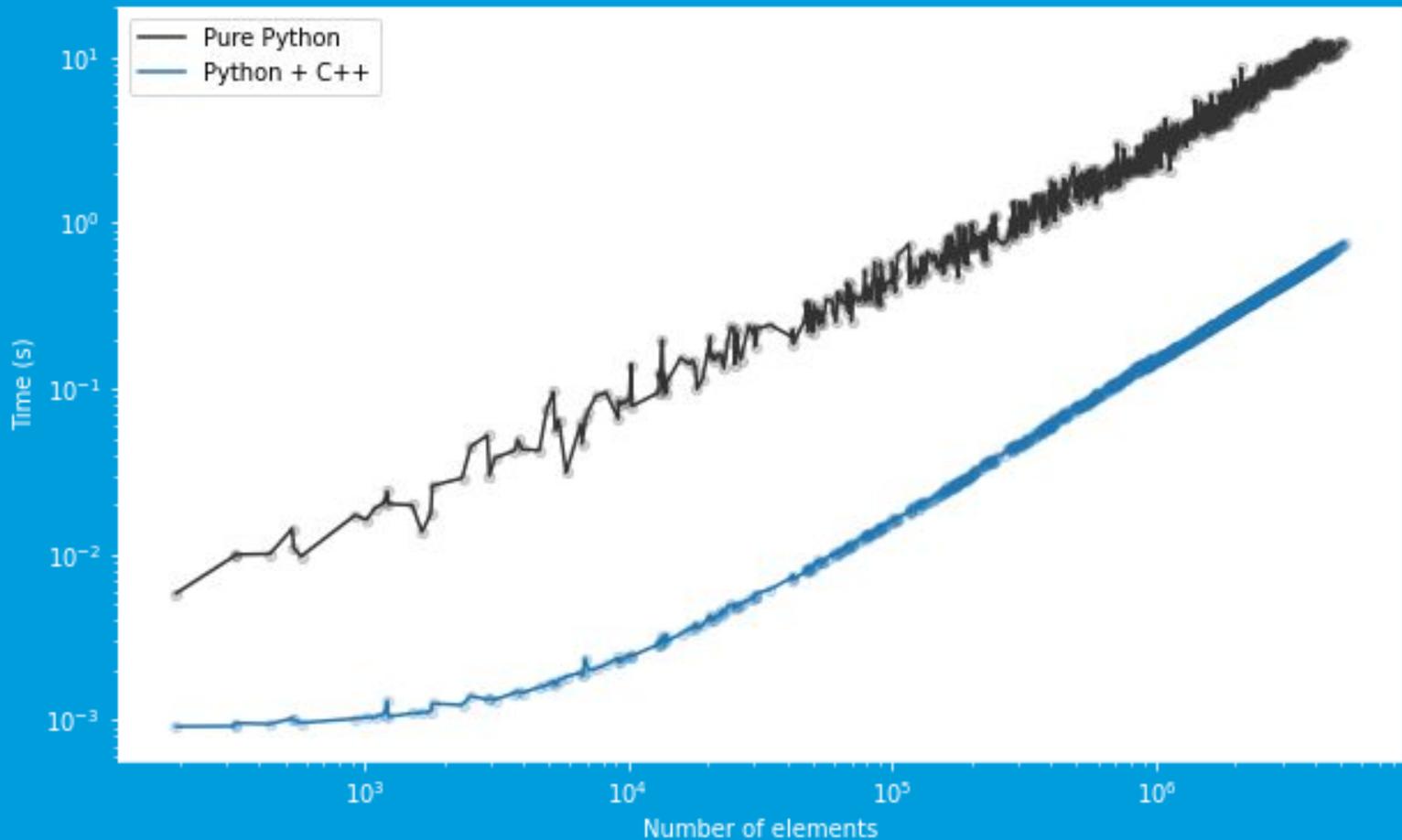
Python + C++

- Read/Parse in C++
- C++ returns columns/arrays
- Python transforms to DF

Comparison of Pure Python vs Python + C++



Comparison of Pure Python vs Python + C++



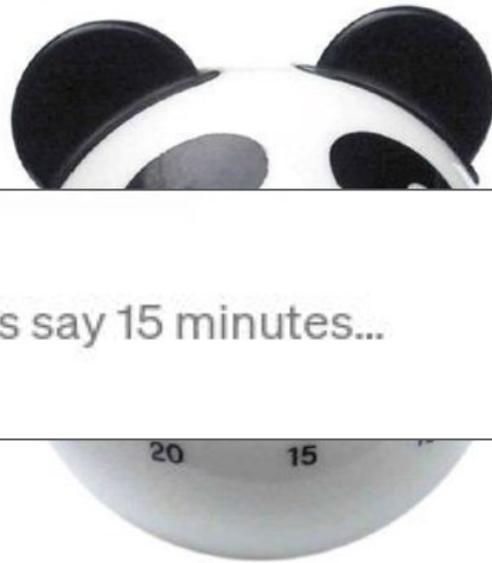
Building a Python C++ module in under 10 minutes



Ok, let's say 15 minutes...

... I guess if you don't know any C++ it may take a bit longer, but not that much. Just start out with programming like you would in Python, but declare variables with types, put semicolons at the end of lines, put loop and branching conditions in parentheses, put curly braces around indented blocks and forget about the colons that start Python indented blocks... that should get you about 80% of the way there. Oh and avoid pointers for

Building a Python C++ module in under 10 minutes



Ok, let's say 15 minutes...

Ok, let's say 15 minutes...

... I guess if you don't know any C++ it may take a bit longer, but not that much. Just start out with programming like you would in Python, but declare variables with types, put semicolons at the end of lines, put loop and branching conditions in parentheses, put curly braces around indented blocks and forget about the colons that start Python indented blocks... that should get you about 80% of the way there. Oh and avoid pointers for

Building a Python C++ module in under 10 minutes

... I guess if you don't know any



Ok, let's say 15 minutes...

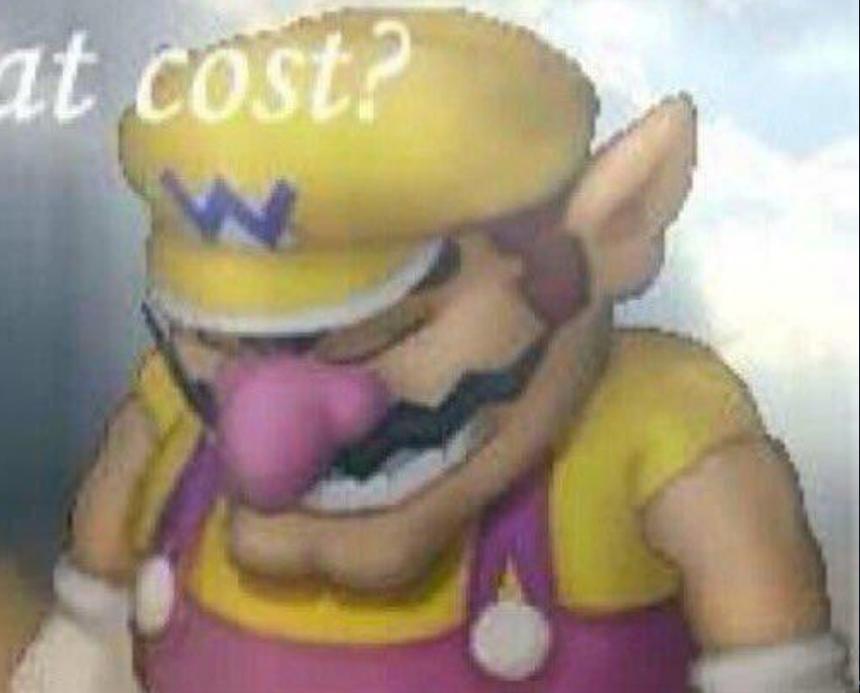
Ok, let's say 15 minutes...

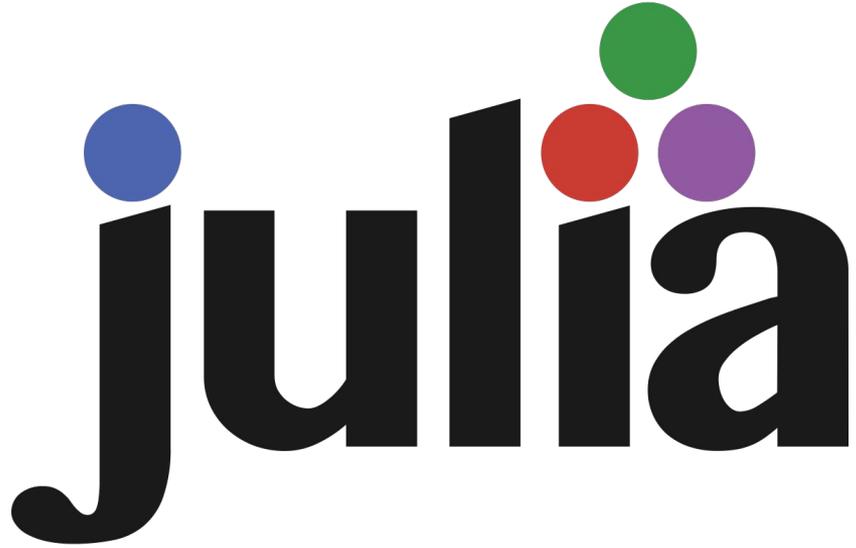


get you about 80% of the way
there. Oh and avoid pointers for

i've won.....

but at what cost?



The logo for the Julia programming language. It features the word "julia" in a bold, black, sans-serif font. The letter "j" has a blue circle above it. The letter "i" has a red circle above it. The letter "l" has a green circle above it. The letter "i" has a purple circle above it. The letter "a" has a red circle above it. The letter "i" has a purple circle above it. The letter "a" has a green circle above it.

julia

- Math friendly
 - $A * x$ works
- C/Fortran native integration
 - Access “classic” libraries
 - Maintain legacy code
- Fast
 - Compiles to C speed
- Maintainer friendly
 - Many good practices by default

Basic Julia (.jl)

- Read all lines
- Split at #
- Generate dictionary
 - key => vector
- Split at ,
- Parse individually
- Count and allocate
- Fill arrays and return

```
1 using Parsers
2
3 function read_arrays_jl_dict(filename)
4     lines = split.(readlines(filename), "#")
5     D = Dict(
6         Parsers.parse{Int, line[1]} => Parsers.parse{Int, split(line[2], ",")}
7         for line in lines
8     )
9     n = sum(length(v) for (k, v) in D)
10    keys = zeros{Int, n}
11    indexes = zeros{Int, n}
12    values = zeros{Int, n}
13
14    count = 0
15    for (k, v) in D
16        m = length(v)
17        idx = count+1:count+m
18        keys[idx] .= k
19        indexes[idx] .= 0:m-1
20        values[idx] .= v
21        count += m
22    end
23
24    return keys, indexes, values
25 end
```



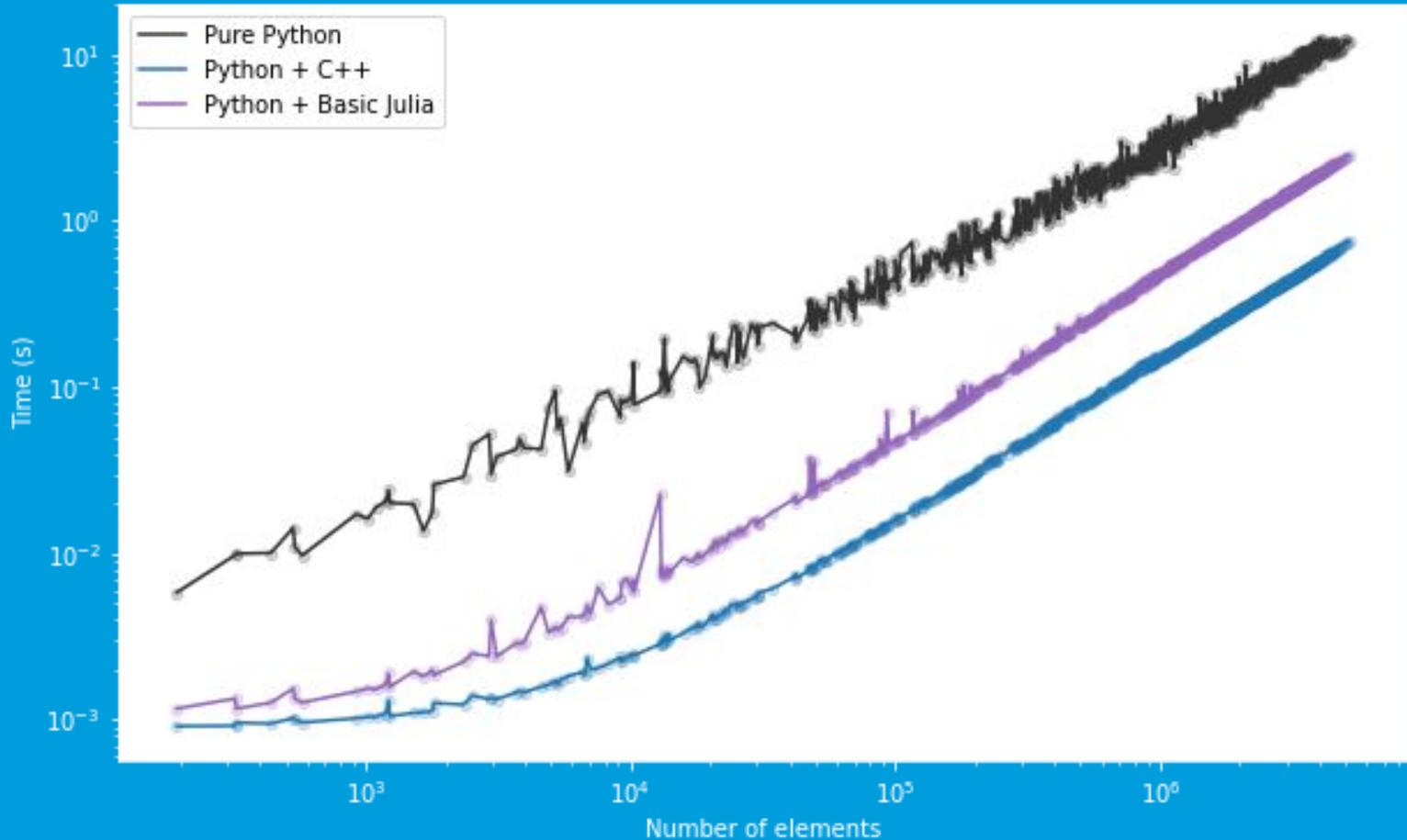
Basic Julia (.py)

- Python Code
- Include julia code
- Call function
- Get arrays
- Create DataFrame in Python

```
1 def load_external(arrays):
2     df = pd.DataFrame.from_records({
3         "key": arrays[0],
4         "list_index": arrays[1],
5         "value": arrays[2]
6     }, index=["key", "list_index"])
7     return df
8
9     ### JULIA ###
10    import julia
11    from julia.api import Julia
12    jl = Julia(runtime="julia-1.6.3")
13    from julia import Main
14    jl.eval('include("jl_reader_dict.jl)")')
15    def load_julia_dict(filename):
16        arrays = jl.eval(f'read_arrays_jl_dict("{filename}")')
17        return load_external(arrays)
```



Comparison of Pure Python vs Python + C++ vs Python + Basic Julia



But can we do better?

- Parsing is slow
- C/C++ uses scanf/cin
- Can use scanf, but what about pure Julia?
- “Low level” reading

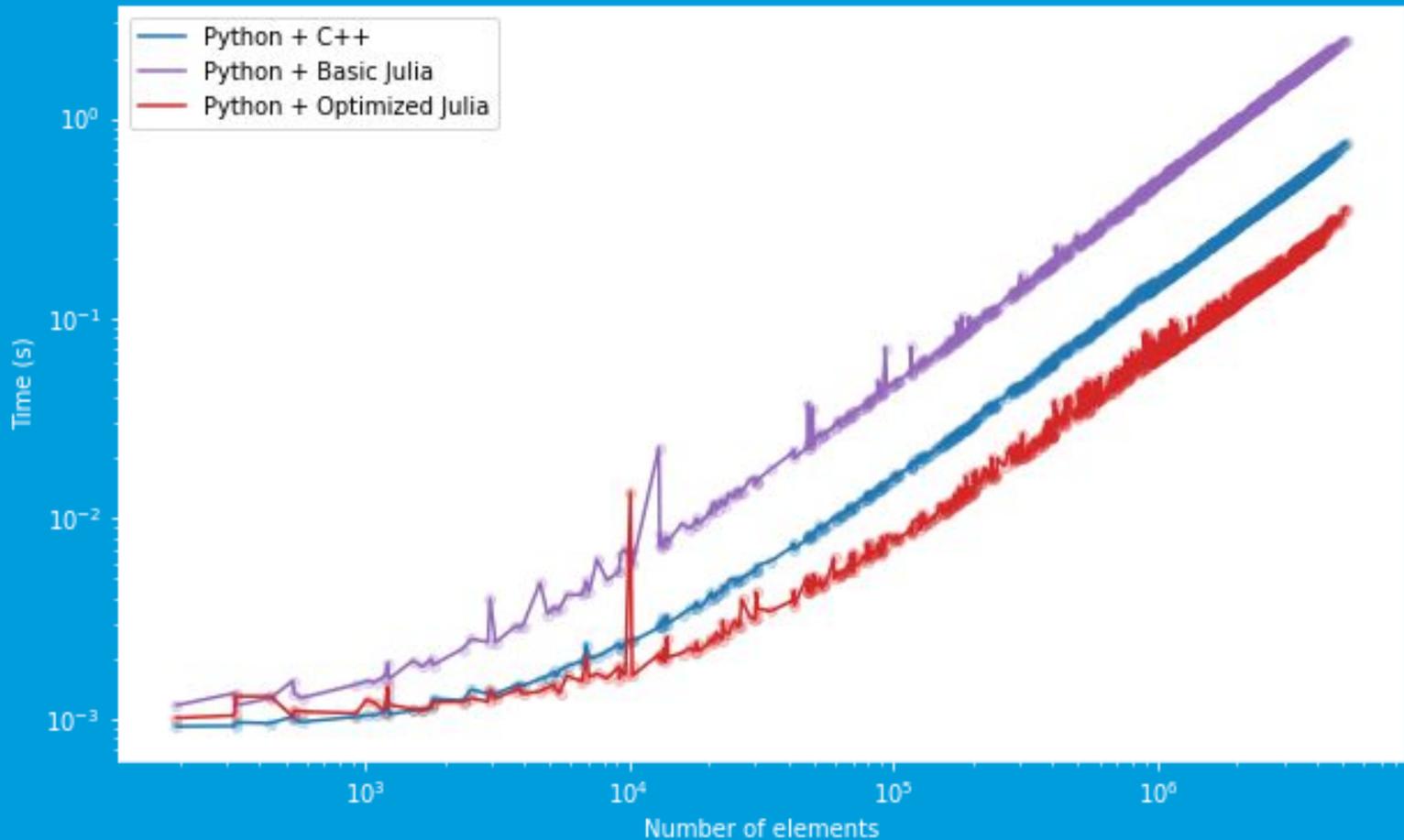
Optimized Julia

- Read bytes
- Check each byte to compute size
- Check each byte to find string that corresponds to Int
- Very C-like

```
1 function read_arrays_jl_manual(filename)
2     file = read(filename)
3     n = sum(c == UInt8(',') || c == UInt8('\n') for c in file) + 1
4     if file[end] == UInt8('\n') # Pesky extra blank line
5         n -= 1
6     end
7     keys = zeros{Int, n}
8     indexes = zeros{Int, n}
9     values = zeros{Int, n}
10
11     count, k, j, fi, fj, fn = -1, 0, 1, 2, length(file)
12     while fi < fn && count ≤ n
13         while (file[fj] ≥ 0x30) fj += 1 end
14         x = Parsers.parse{Int, view(file, fi:fj-1)}
15         c = file[fj]
16         if c == Int8('#')
17             k = x
18             j = 0
19         else
20             keys[count] = k
21             indexes[count] = j
22             j += 1
23             values[count] = x
24             count += 1
25         end
26         fi = fj + 1
27         fj = fi + 1
28     end
29
30     return keys, indexes, values
31 end
```



Comparison of Python + C++ vs Python + Basic Julia vs Python + Optimized Julia



Final remarks

- New title: “N times faster in Julia in 30 seconds”
- Takeaway: Write basic Julia and optimize as needed
- More Julia experiments
- GT is looking for good cases



Thanks for
coming.
Get in touch



abelsiqueira



abel_siqueira



abelsiqueira.github.io



abel.siqueira@esciencecenter.nl



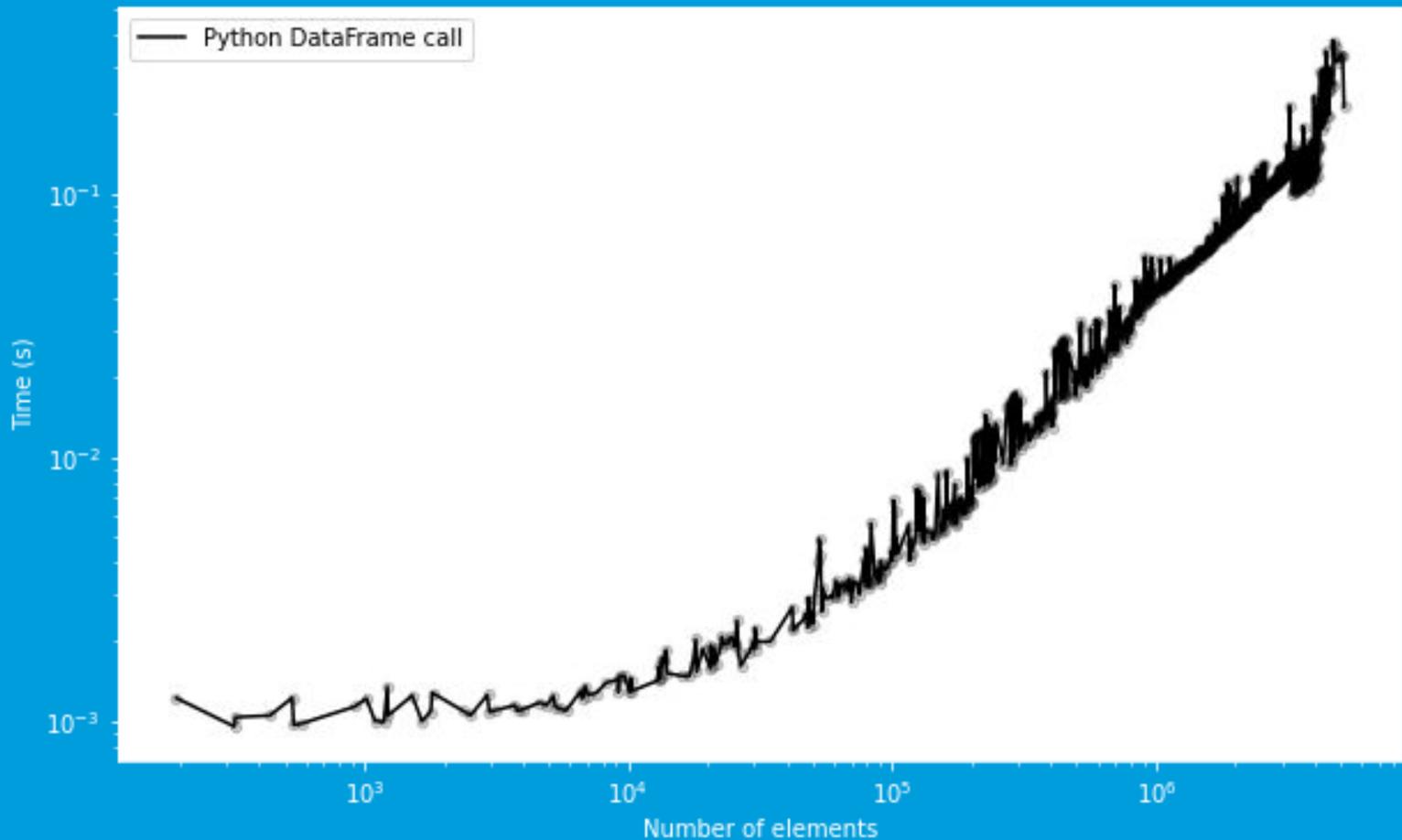
06 2648 9344



abel-siqueira



siqueiraabel



Comparison of Pure Python vs Python + C++ vs Python + Basic Julia

