N times faster data loading with Julia

Abel & Faruk

netherlands eScience center

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

```
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

		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

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

netherlands eScience center

Pure Python

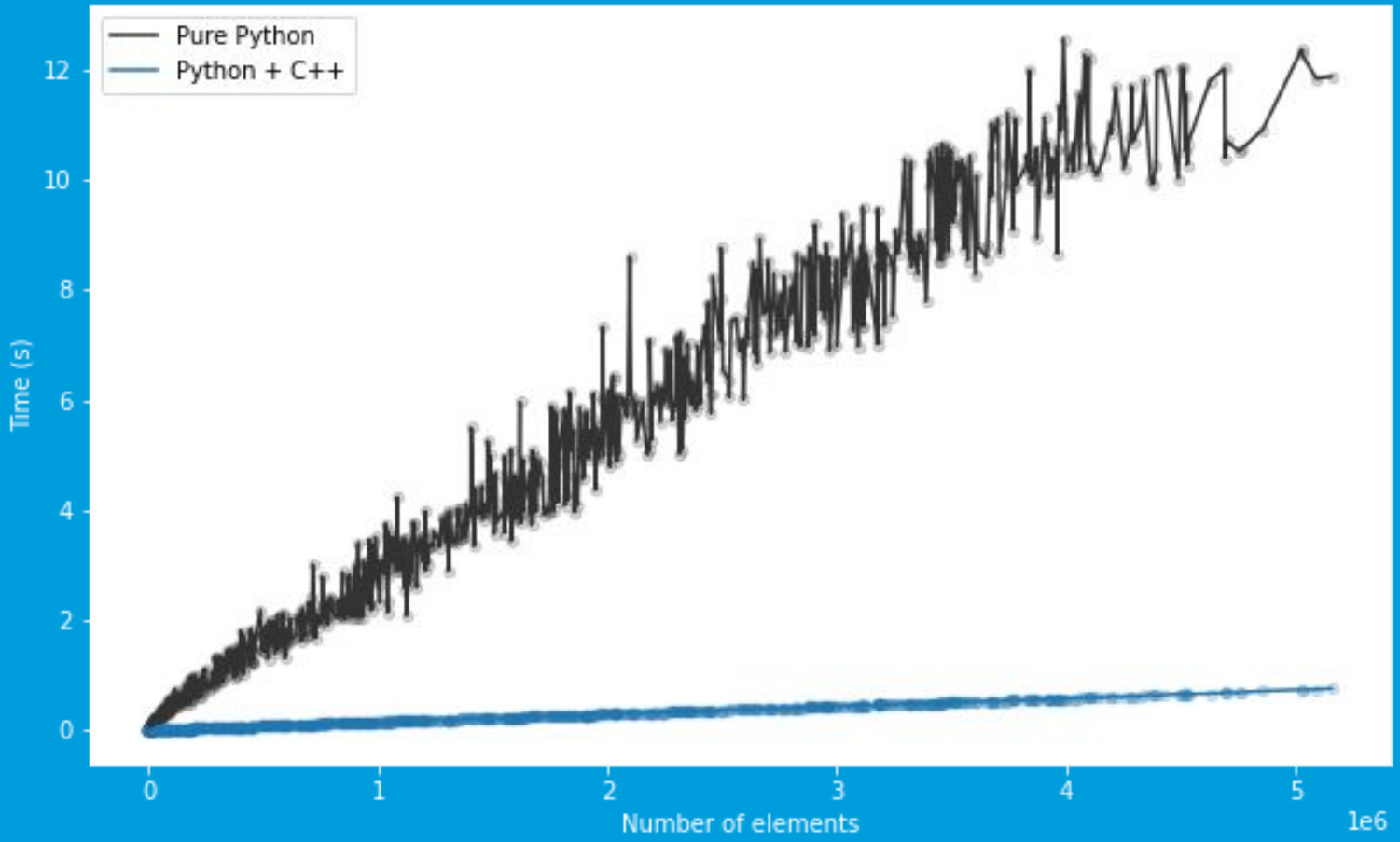
- Read + Split + Stack

Python + C++

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

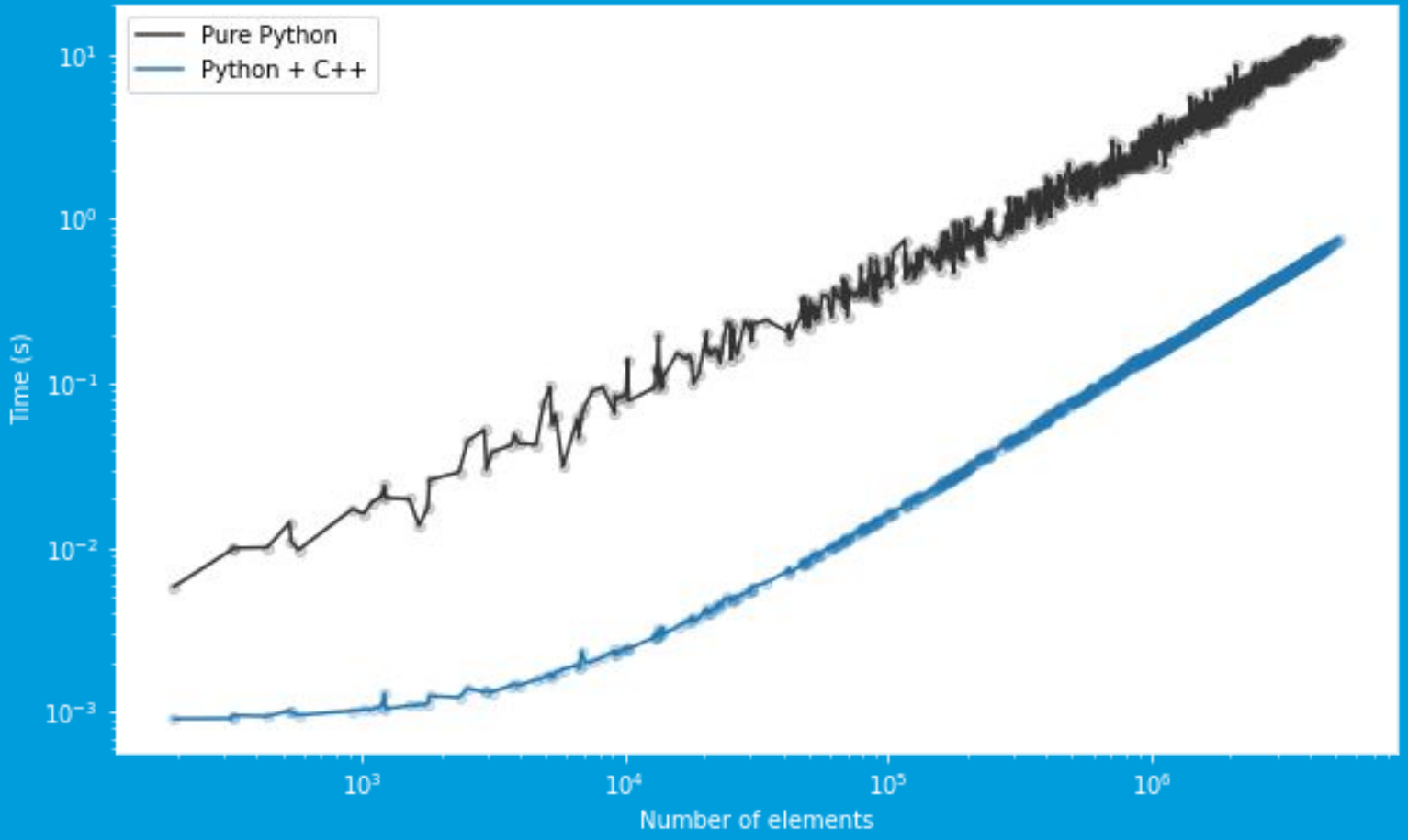
netherlands eScience center

Comparison of Pure Python vs Python + C++



netherlands eScience center

Comparison of Pure Python vs Python + C++



10 minutes of C++

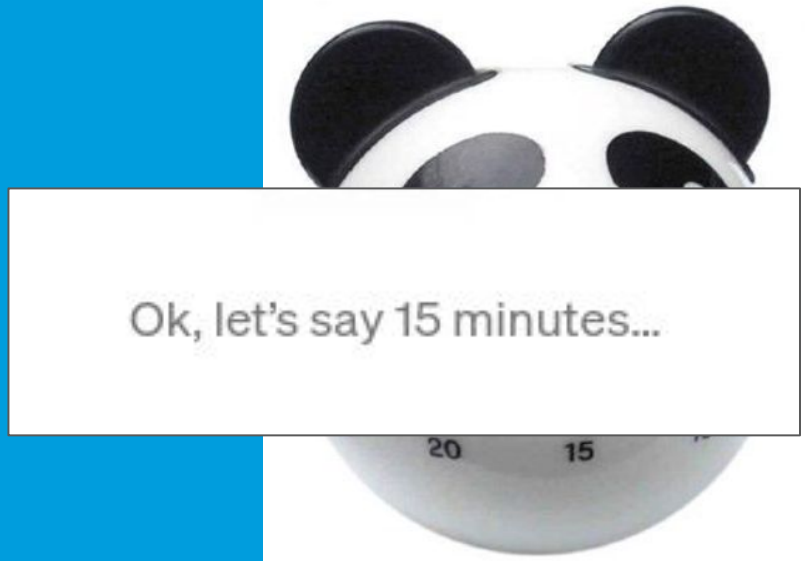
Building a Python C++ module in under 10 minutes

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

10 minutes of C++

Building a Python C++ module in under 10 minutes



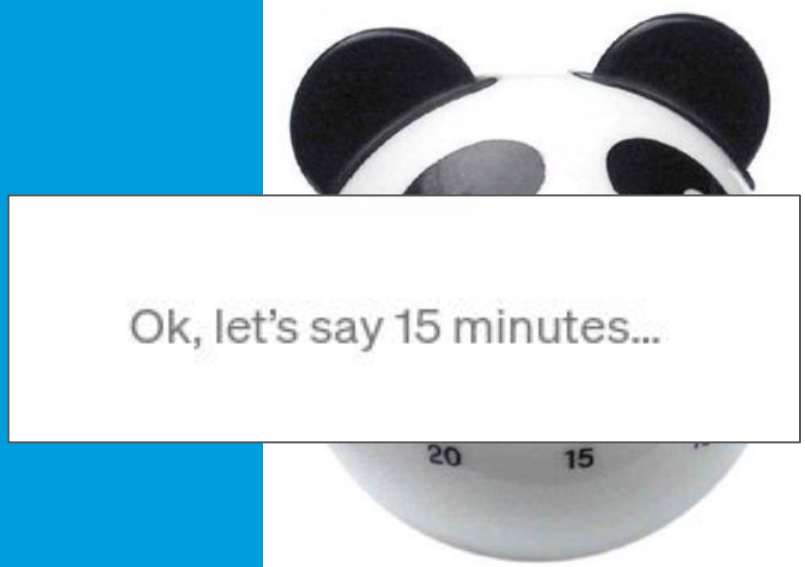
Ok, let's say 15 minutes...

… 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

10 minutes of C++

Building a Python C++ module in under 10 minutes

… I guess if you don't know any

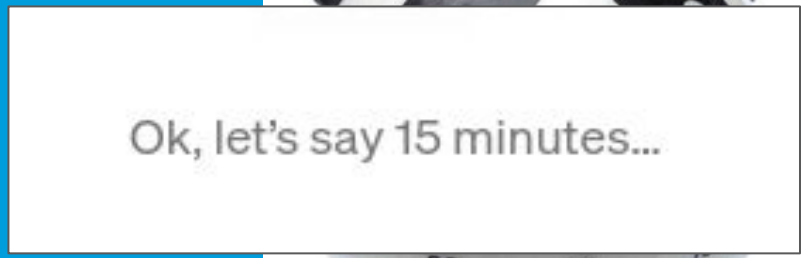


Ok, let's say 15 minutes...

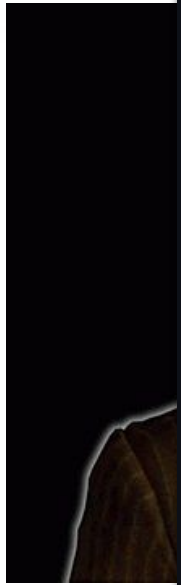
get you about 80% of the way there. Oh and avoid pointers for

netherlands eScience center

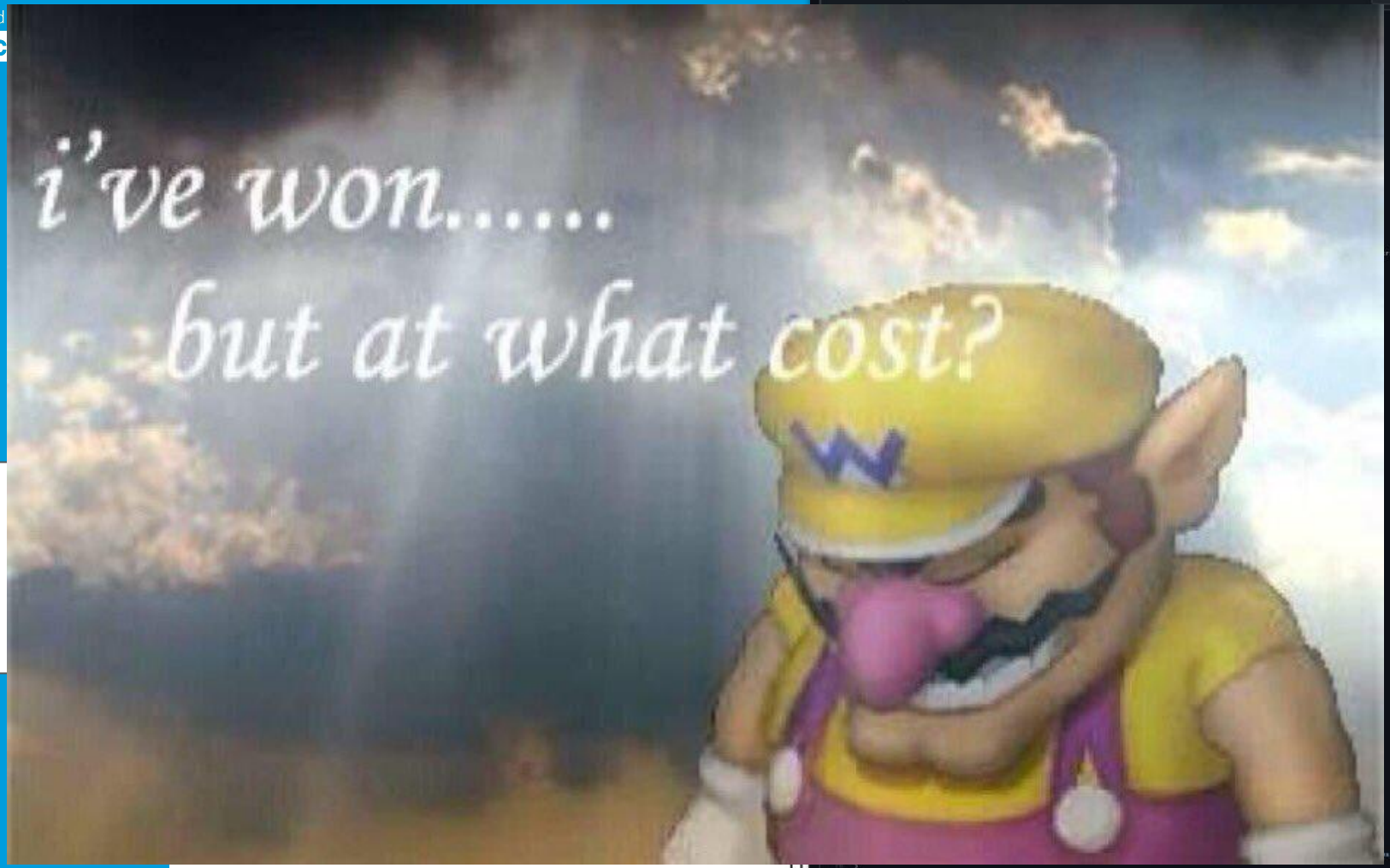
Building a Python C++ module



Ok, let's say 15 minutes...



```
76 lines (66 sloc) | 2.27 KB                                                    Copy  Raw  Blame

#include "pybind11/pybind11.h"

#define FORCE_IMPORT_ARRAY
#include "xtensor-python/pyarray.hpp"

#include <string>
#include <fstream>

std::size_t count_commas_and_newlines(const std::string& filename) {
  std::ifstream file(filename);
  char c;
  std::size_t count = 0;
  while(file.get(c)) {
    if (c == ',' || c == '\n') {
      ++count;
    }
  }
  return count;
}


std::tuple<xt::pyarray<unsigned int>, xt::pyarray<unsigned short>, xt::pyarray<unsigned long>> load_confusion_index(const std::string &filename) {
  std::size_t array_size = count_commas_and_newlines(filename);

  auto confusion_array = xt::pyarray<unsigned int>::from_shape({array_size});
  auto confusion_word_index_array = xt::pyarray<unsigned short>::from_shape({array_size});
  auto word_anahash_array = xt::pyarray<unsigned long>::from_shape({array_size});

  std::size_t ix = 0;
  unsigned int confusion;
  unsigned short confusion_word_index = 0;
  unsigned long word_anahash;
  char delimiter;

  std::ifstream file(filename);
  while(file >> confusion >> delimiter) {
    if (delimiter == '#') {
      // streaming operator will skip new-lines by default, use noskipws
      while (file >> std::noskipws >> word_anahash >> delimiter) {
        confusion_array[ix] = confusion;
        confusion_word_index_array[ix] = confusion_word_index++;
        word_anahash_array[ix] = word_anahash;
        ++ix;
        if (delimiter == '\n') {
          confusion_word_index = 0;
          break;
        }
      }
    } else {
      throw std::runtime_error("non-# delimiter found after confusion!");
    }
  }
  file.close();

  return {confusion_array, confusion_word_index_array, word_anahash_array};
}

// Python Module and Docstrings

PYBIND11_MODULE(ticcl_output_reader, m)
{
    xt::import_numpy();

    m.doc() = R"pbdoc(
        An efficient reader of TICCL output files

        .. currentmodule:: ticcl_output_reader

        .. autosummary::
           :toctree: _generate

           load_confuslist_index
    )pbdoc";

    m.def("load_confuslist_index", load_confuslist_index, "Load a confusion list index file. Returns three NumPy arrays with dtypes uint32, uint16 and uint64 re
}
```


i've won.....
but at what cost?

netherlands eScience center

- 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

netherlands eScience center

Basic Julia (.jl)

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

```julia
using Parsers

function read_arrays_jl_dict(filename)
  lines = split.(readlines(filename), "#")
  D = Dict(
    Parsers.parse(Int, line[1]) => Parsers.parse.(Int, split(line[2], ","))
    for line in lines
  )
  n = sum(length(v) for (k, v) in D)
  keys = zeros(Int, n)
  indexes = zeros(Int, n)
  values = zeros(Int, n)

  count = 0
  for (k, v) in D
    m = length(v)
    idx = count+1:count+m
    keys[idx] .= k
    indexes[idx] .= 0:m-1
    values[idx] .= v
    count += m
  end

  return keys, indexes, values
end
```

netherlands eScience center

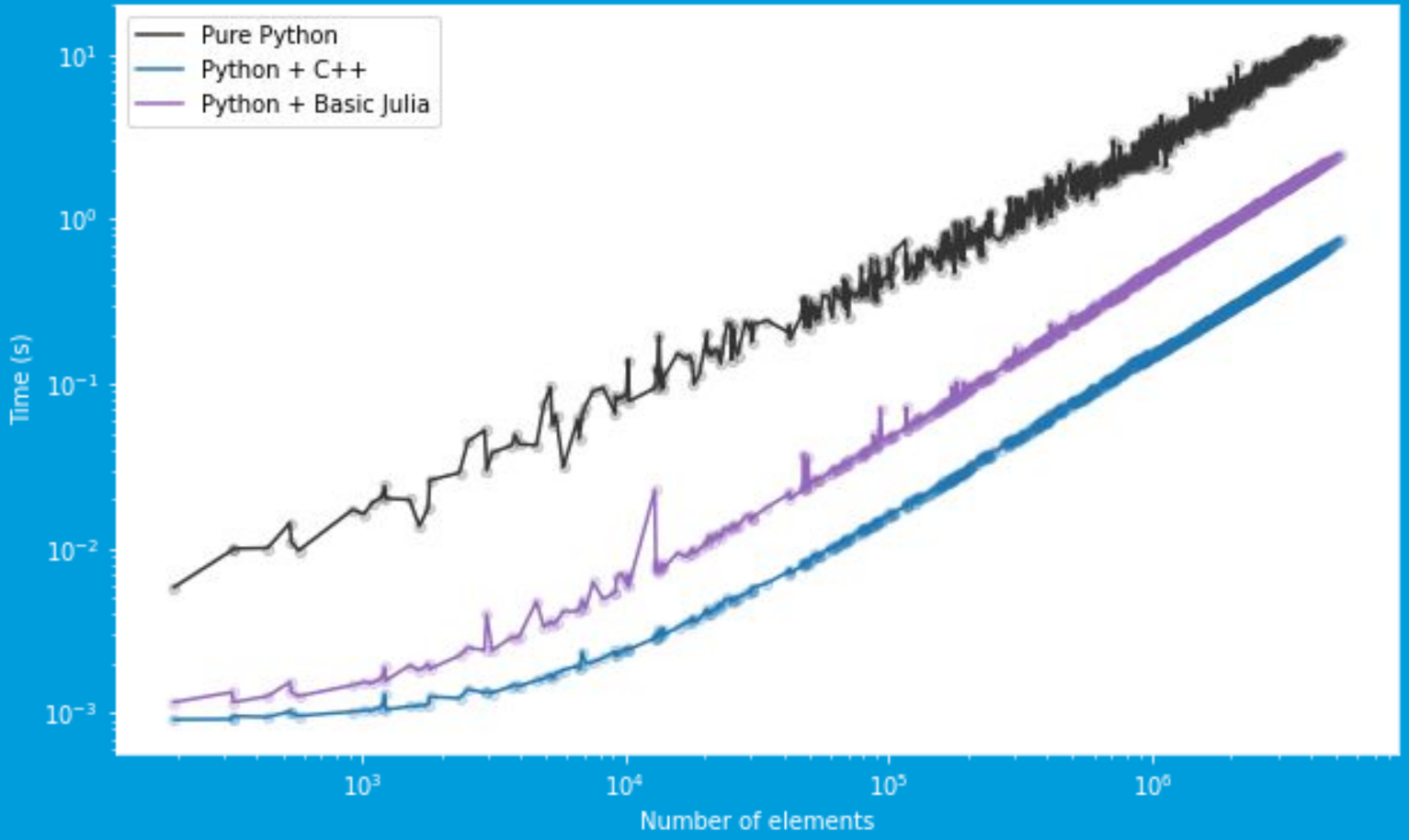
Basic Julia (.py)

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

```python
def load_external(arrays):
    df = pd.DataFrame.from_records({
            "key": arrays[0],
            "list_index": arrays[1],
            "value": arrays[2]
        }, index=["key", "list_index"])
    return df

### JULIA ###
import julia
from julia.api import Julia
jl = Julia(runtime="julia-1.6.3")
from julia import Main
jl.eval('include("jl_reader_dict.jl")')
def load_julia_dict(filename):
    arrays = jl.eval(f'read_arrays_jl_dict("{filename}")')
    return load_external(arrays)
```

netherlands eScience center

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



netherlands eScience center

But can we do better?

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

netherlands eScience center

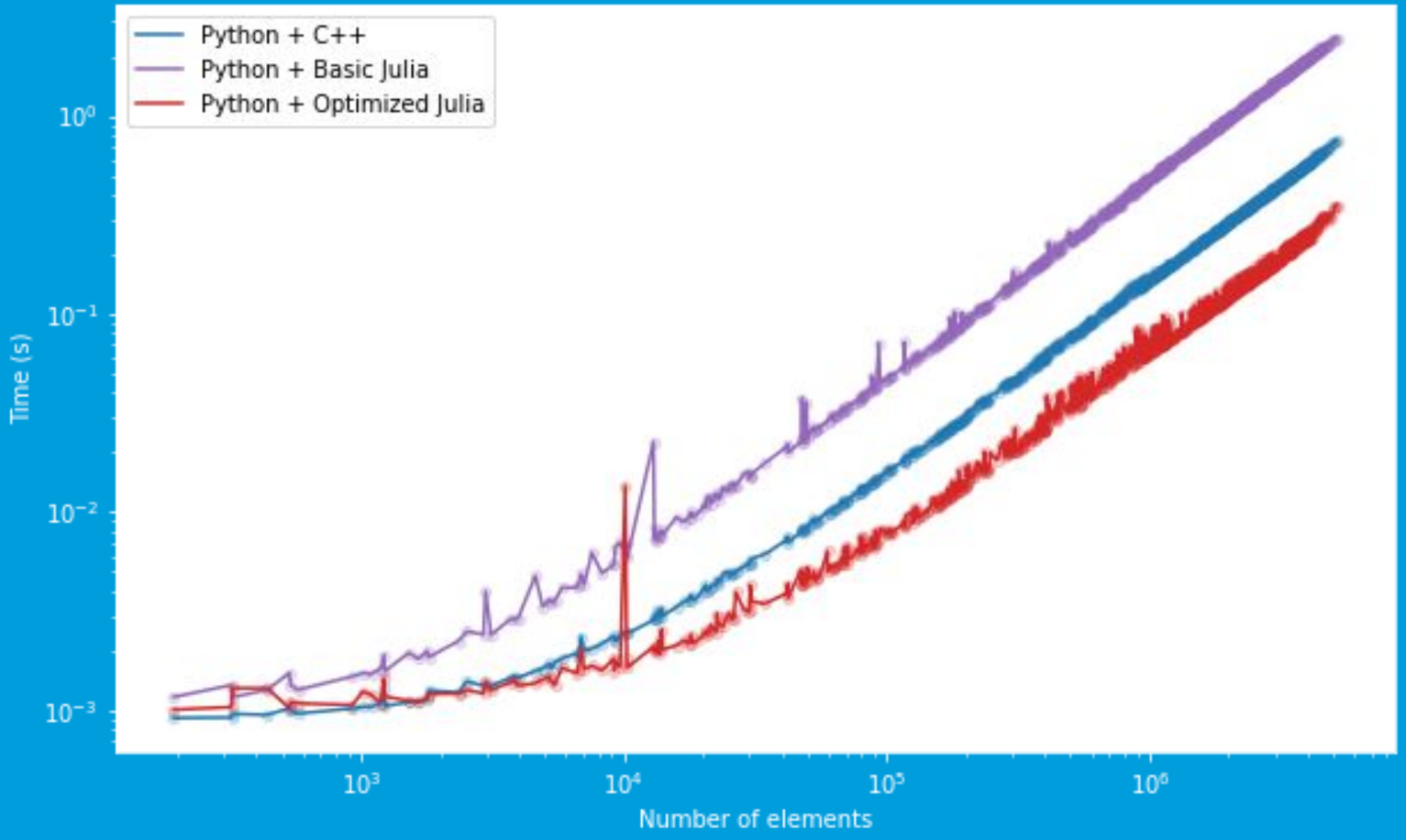
Optimized Julia

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

```
function read_arrays_jl_manual(filename)
  file = read(filename)
  n = sum(c == UInt8(',') || c == UInt8('\n') for c in file) + 1
  if file[end] == UInt8('\n') # Pesky extra blank line
    n -= 1
  end
  keys = zeros(Int, n)
  indexes = zeros(Int, n)
  values = zeros(Int, n)

  count, k, j, fi, fj, fn = -1, 0, 1, 2, length(file)
  while fi < fn && count ≤ n
    while (file[fj] ≥ 0x30) fj += 1 end
    x = Parsers.parse(Int, view(file, fi:fj-1))
    c = file[fj]
    if c == Int8('#')
      k = x
      j = 0
    else
      keys[count] = k
      indexes[count] = j
      j += 1
      values[count] = x
      count += 1
    end
    fi = fj + 1
    fj = fi + 1
  end

  return keys, indexes, values
end
```

netherlands eScience center

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



netherlands eScience center

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

netherlands eScience center

Thanks for coming. Get in touch


abelsiqueira


abel_siqueira


abelsiqueira.github.io


abel.siqueira@esciencecenter.nl

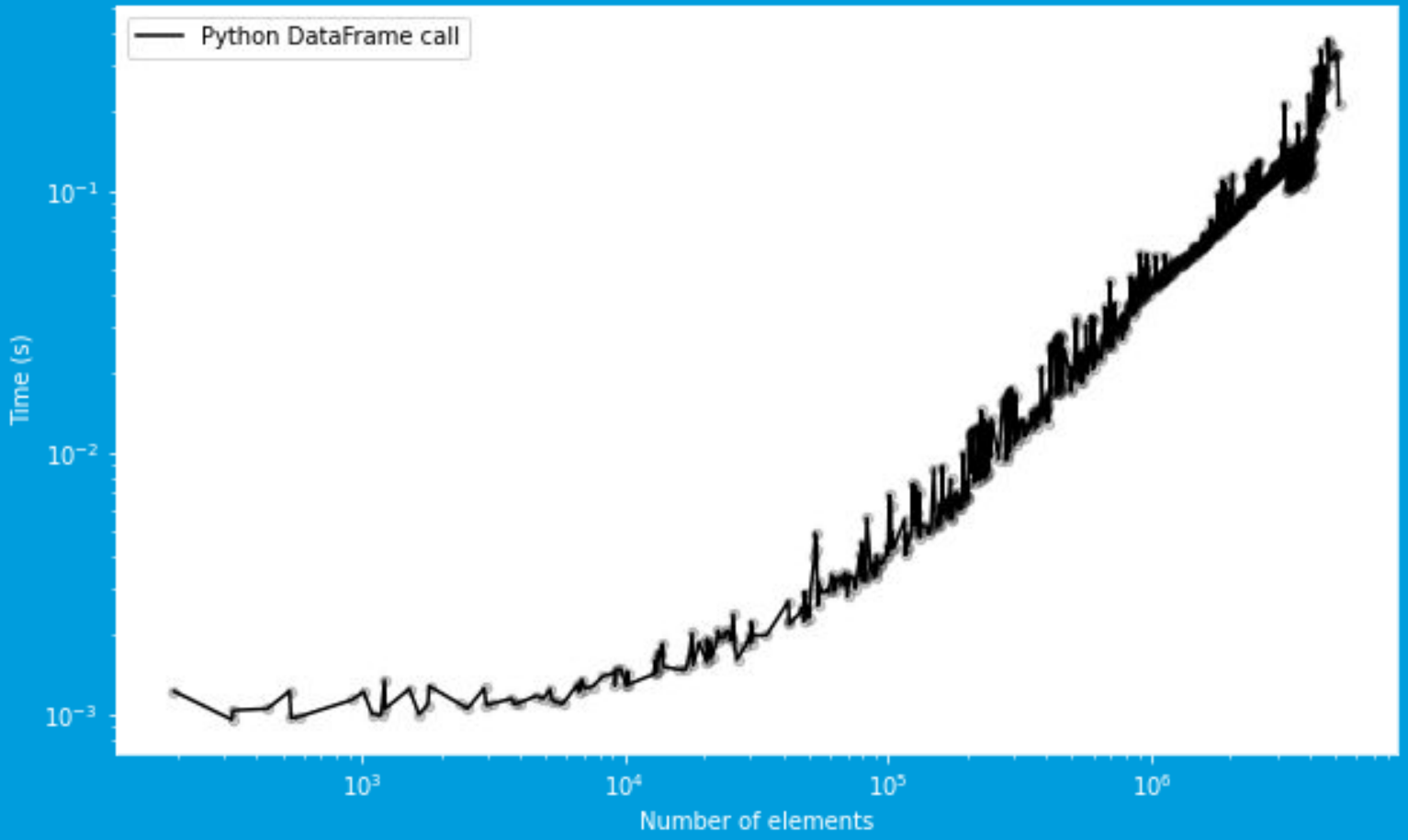

06 2648 9344


abel-siqueira


siqueiraabel

netherlands eScience center

Time to create the DataFrame in Python



netherlands eScience center

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