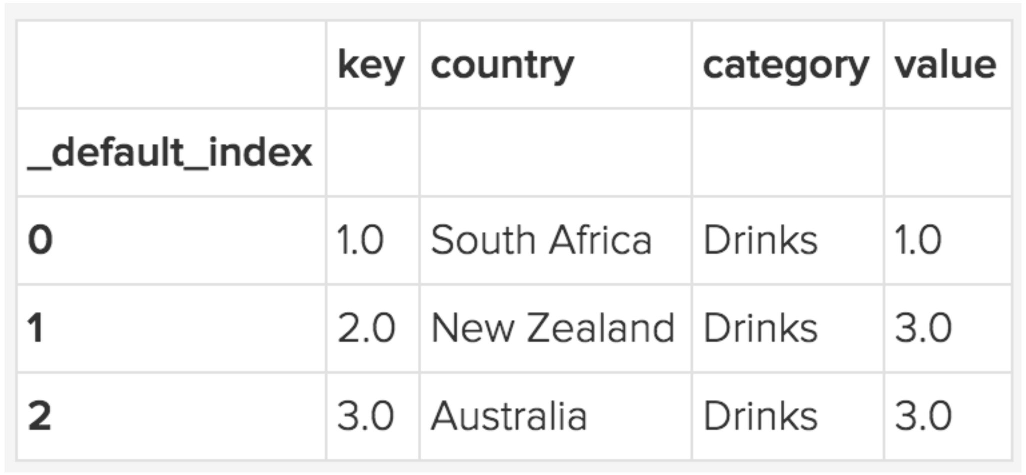
Chasing Pandas

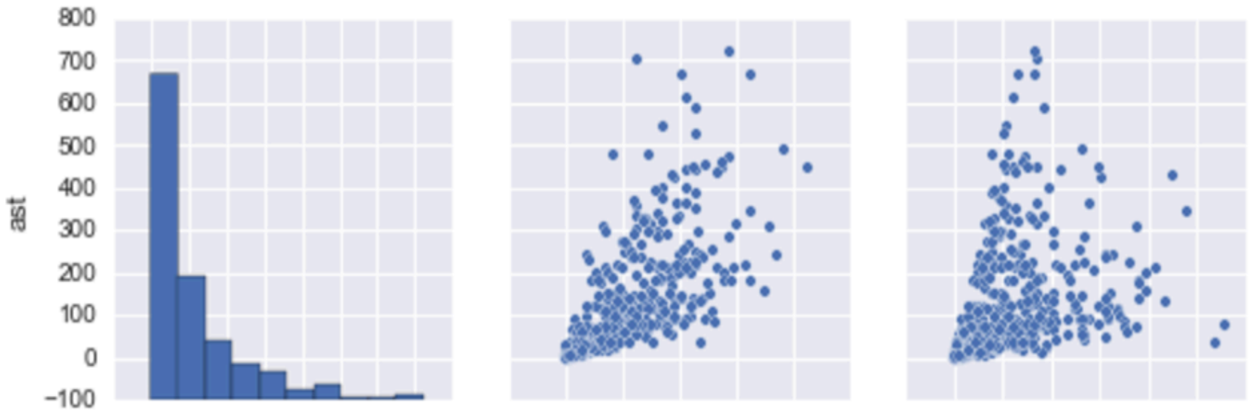
Data Analysis in Ruby

What do you need for data analysis?

- Reading from various sources
- Tabular data
- Indexes
- Mathematical Operations
- Slicing, Filtering, Grouping, Merging
- Time Series
- Visualization and plotting
- Metadata



	key	country	category	value
_default_index				
0	1.0	South Africa	Drinks	1.0
1	2.0	New Zealand	Drinks	3.0
2	3.0	Australia	Drinks	3.0



What tools are people using?
(i.e. the competition)

- Excel
- R
- Python Pandas

- Julia

So why do data analysis in Ruby?

- We ❤️ Ruby
- For the Ruby/Rails ecosystem to stay relevant
- SciRuby provides a variety of libraries for the job

Pandas

- Open Source Python library
- Fast
- Excellent at munging data
- Very active community & development
- Uses NumPy library for fast numeric operations
- Uses DataFrames and Series as it’s main data structures
- Great visualization integration through matplotlib

Pandas

- Driving Python growth!
- Fastest growing major language on stack overflow

https://stackoverflow.blog/2017/09/14/python-growing-quickly/

Quattro

Why?

- Our data is loosely structured but highly dimensional
- No comparable Ruby library at the time (2014) - only NArray, GSL
- We need real-time interrogation
- We already had a mature Rails app and Ruby developers

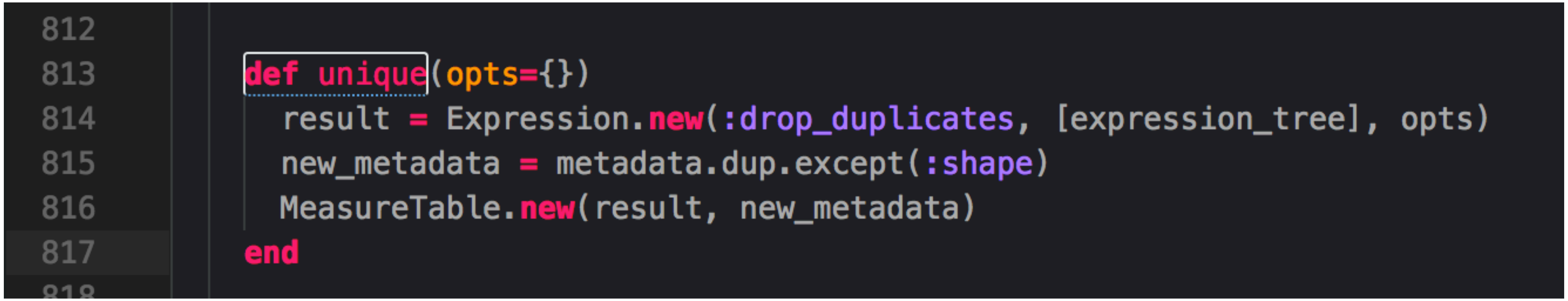
Quattro

What?

- Inspired by Active Records scopes and LISP
- Code is data
- Ruby => Data => Python (Via resque/redis)
- Uses MeasureTable and Measure as it’s main data structures
- Performance very close to that of Pandas:
  - Overhead of 1-4ms per node
  - 1ms average roundtrip
- No Visualization library integrated

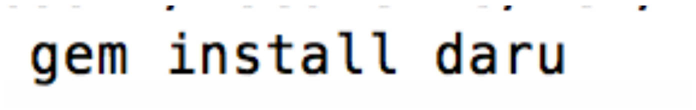
Quattro

Adding new methods



```ruby

def unique(opts={})
  result = Expression.new(:drop_duplicates, [expression_tree], opts)
  new_metadata = metadata.dup.except(:shape)
  MeasureTable.new(result, new_metadata)
end
```

Daru



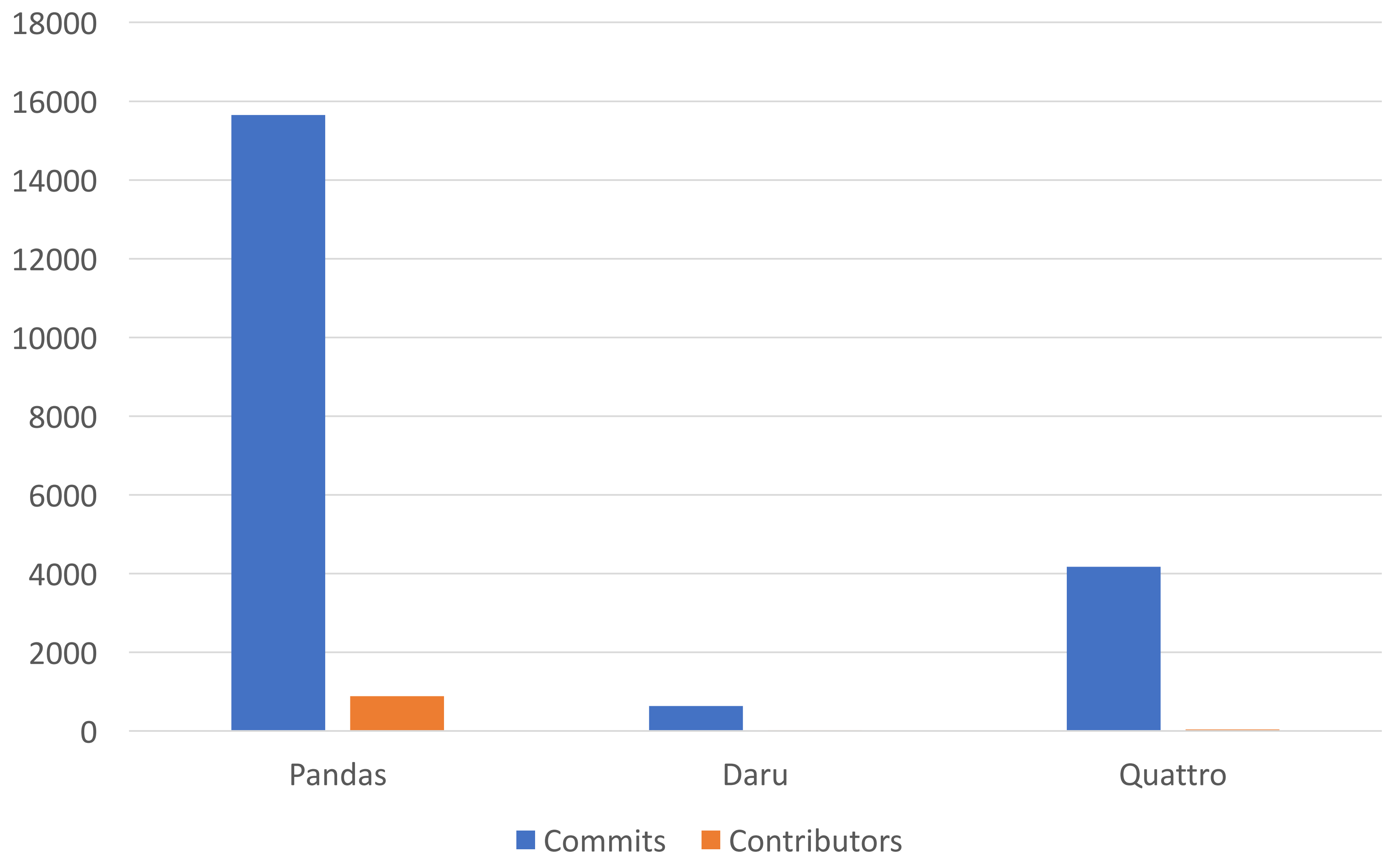
```
gem install daru
```

- Data Analysis in RUby
- Open Source gem
- Part of the SciRuby foundation

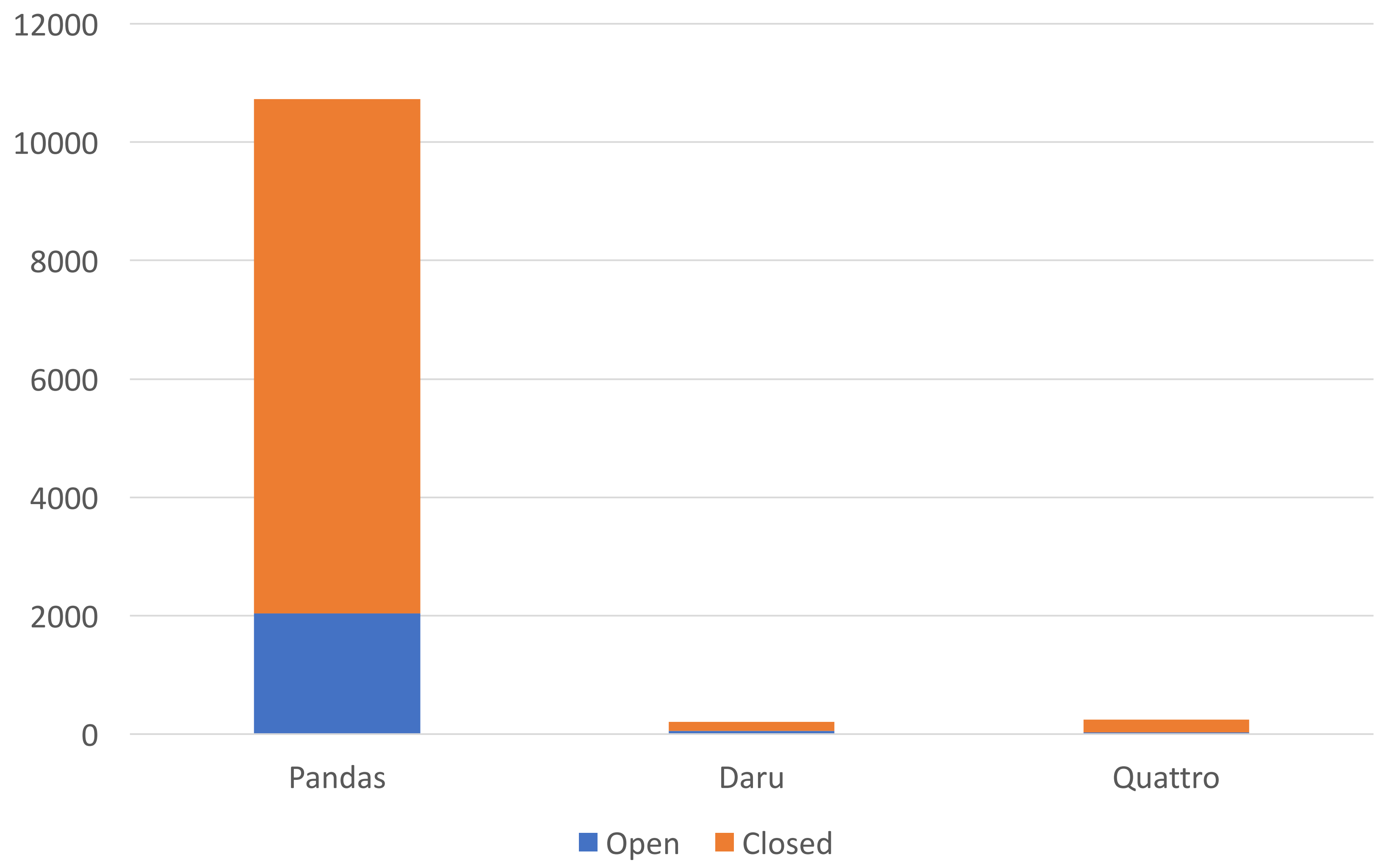
Daru

- Uses NMatrix as a data store for fast numerical operations
- Use DataFrame and Vector as main data structures
- Visualization libraries integrated: GnuPlotRB, NyaPlot, Gruff

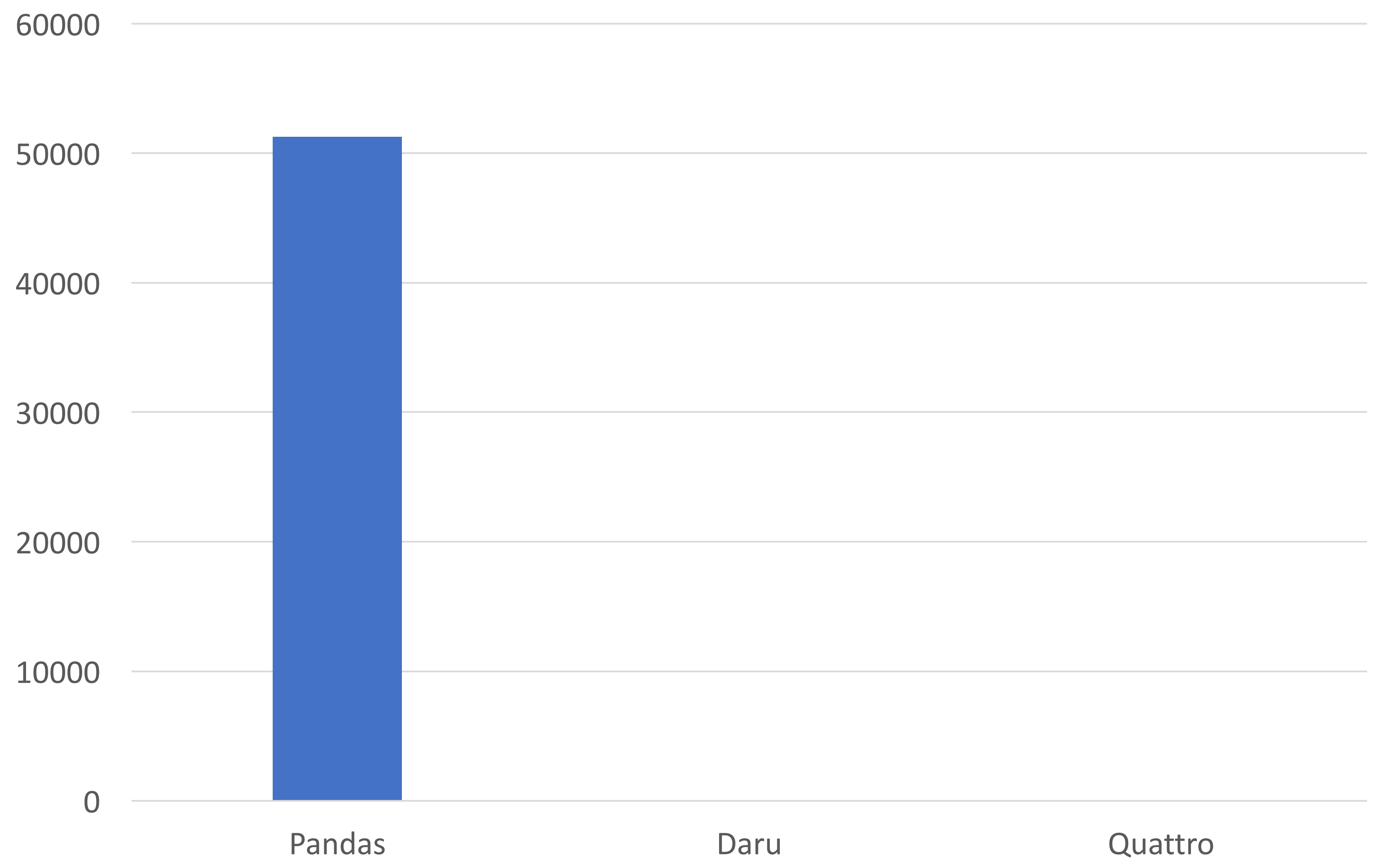
Commits & Contributors



GitHub Issues



StackOverflow Posts



Small communities can still be great communities

- Daru super easy to get involved with
- Very actively maintained
- PRs reviewed very quickly
- Gaining traction in SciRuby

Demo Time

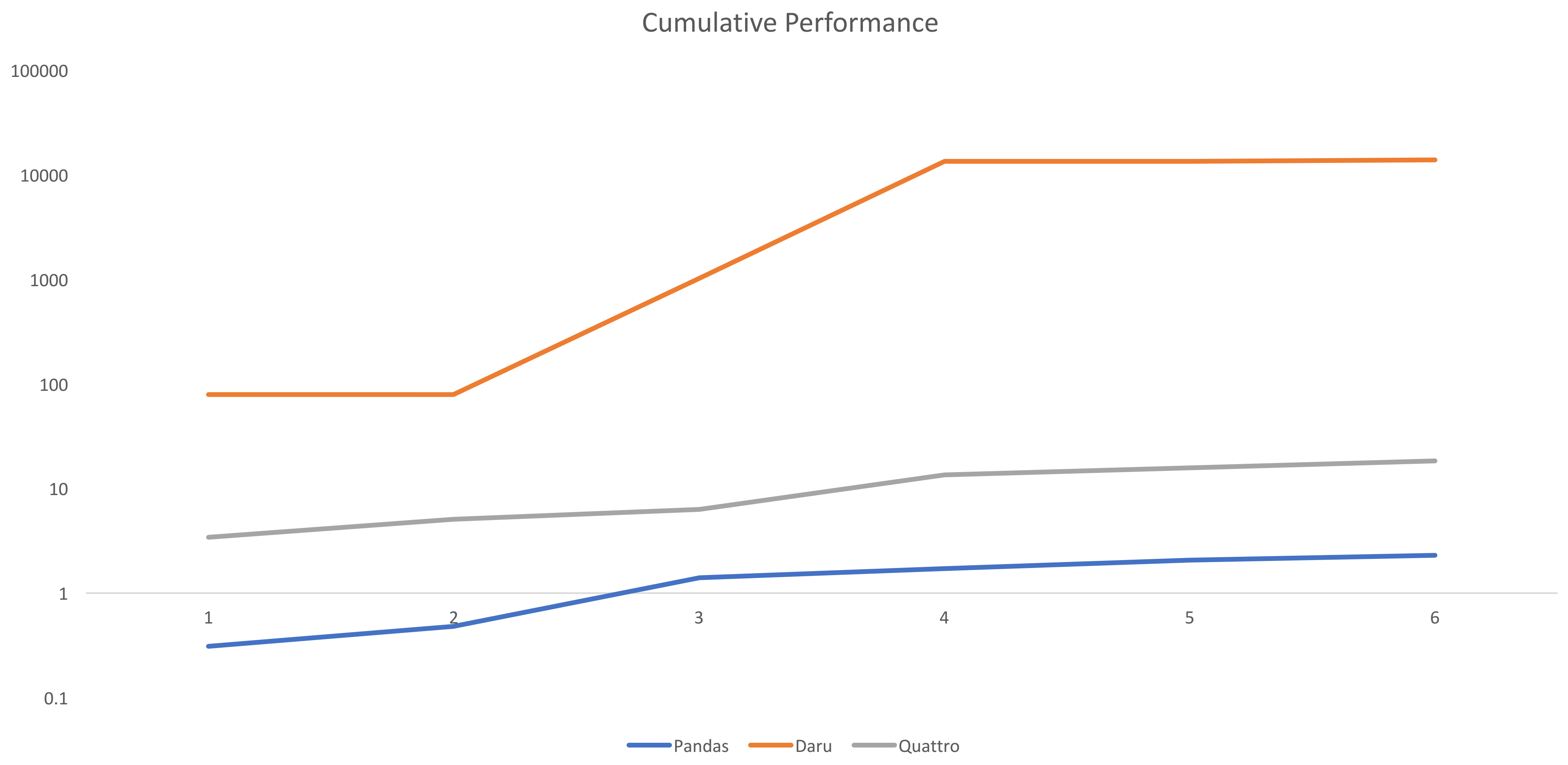
Performance

- **Daru** generally 2+ orders of magnitude slower
- **Quattro** ≈ Pandas + Overhead (1ms + 1-4ms * nodes) [Naïve benchmarking)

Performance 100 runs (s)

	Pandas	Daru	Quattro
From CSV	0.31	79.21	3.42
From Dict / Hash	0.17	0.24	1.64
Drop Duplicates / Uniq	0.93	937.50	1.24
Merge on Index	0.31	12455.56	7.30
Filter on string values	0.36	24.40	2.22
GroupBy Mean, Sort, Head	0.22	390.64	2.56

Performance



Not Just Dumb Piping – We can do better!

- Single Worker Transactions
- Tree rewrites
- Index Partitioning

Performance - RAM

My rule of thumb for pandas is that you should have 5 to 10 times as much RAM as the size of your dataset. – Wes McKinney, 2017

http://wesmckinney.com/blog/apache-arrow-pandas-internals/

Future Development

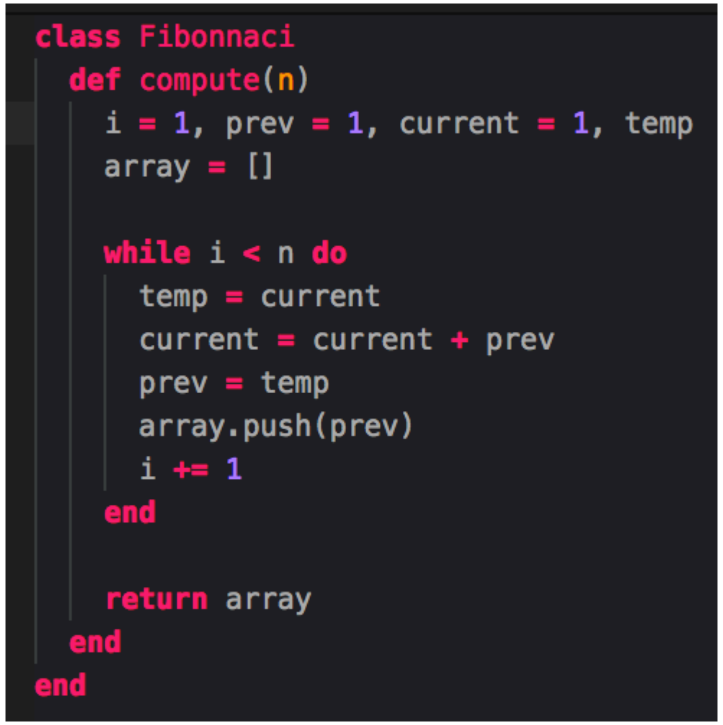
Daru
- V1.0 release, Rubex (C Extensions)

Pandas
- Numba (JIT LLVM)

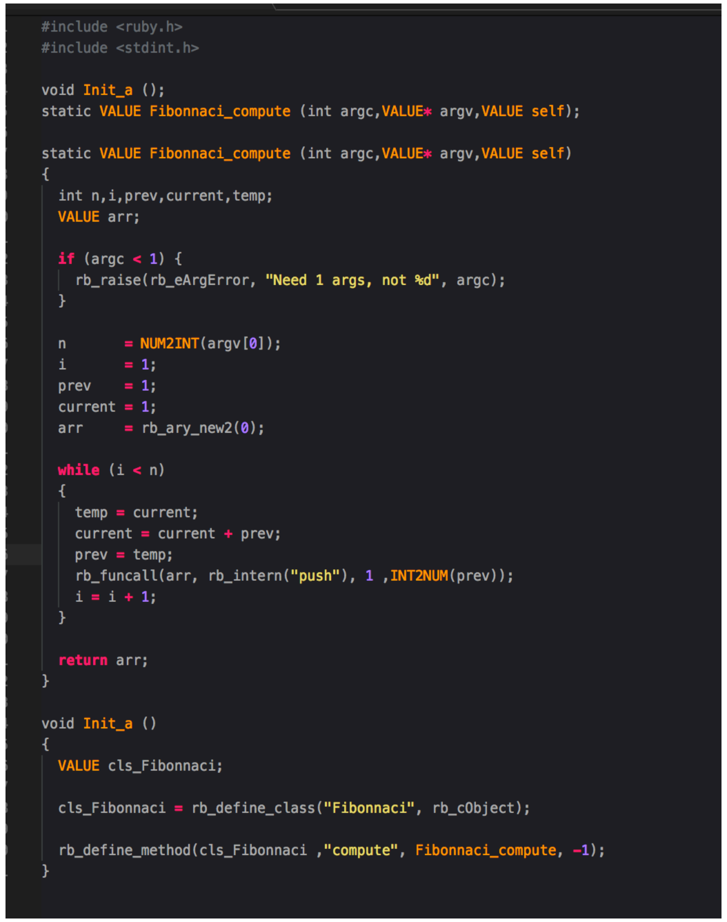
Quattro
- Open Sourcing (Watch this space!)

- **Arrow/Feather**

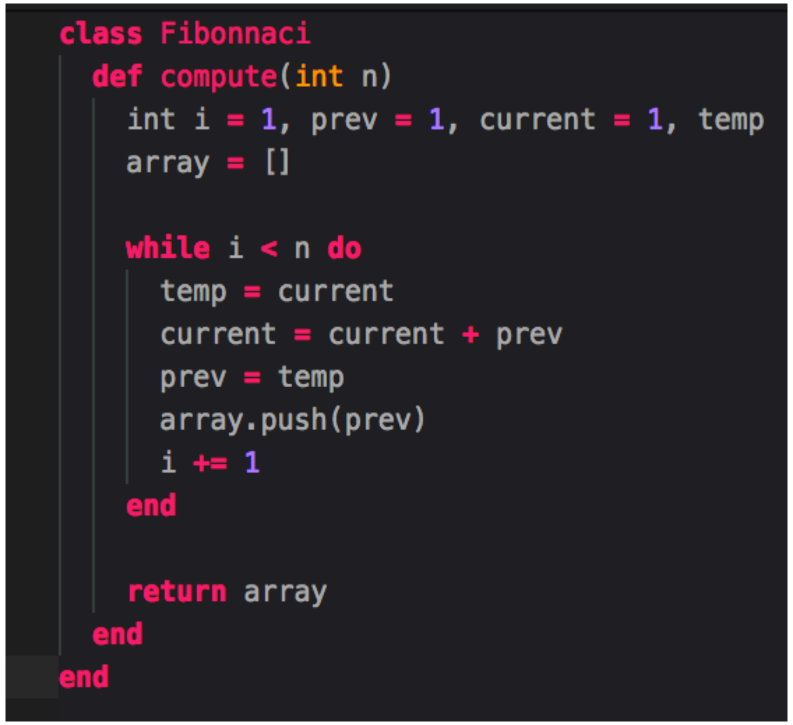
Rubex



```ruby
class Fibonnaci
  def compute(n)
    i = 1, prev = 1, current = 1, temp
    array = []

    while i < n do
      temp = current
      current = current + prev
      prev = temp
      array.push(prev)
      i += 1
    end

    return array
  end
end
```



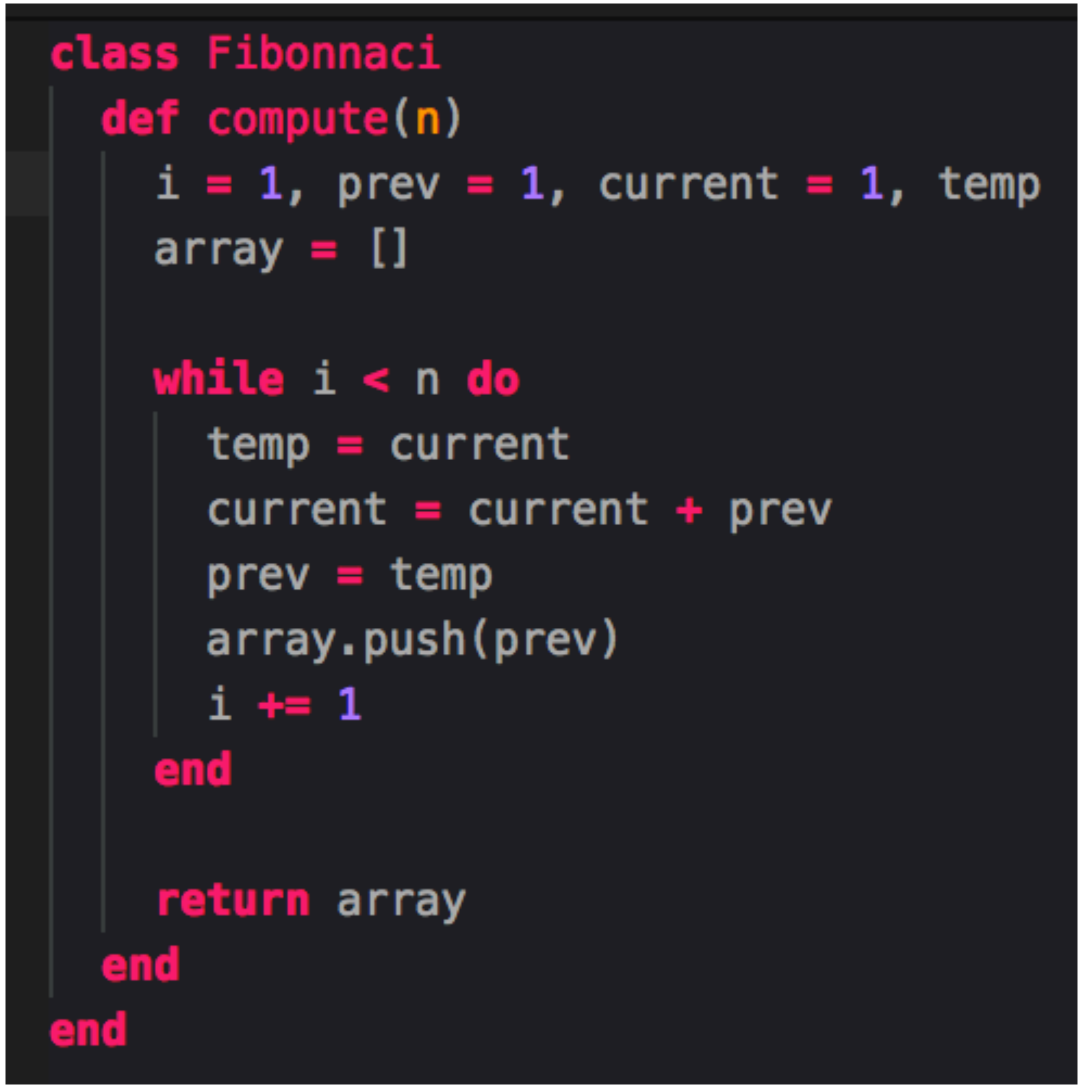
```c
#include <ruby.h>
#include <stdint.h>

void Init_a ();
static VALUE Fibonnaci_compute (int argc,VALUE* argv,VALUE self);

static VALUE Fibonnaci_compute (int argc,VALUE* argv,VALUE self)
{
  int n,i,prev,current,temp;
  VALUE arr;

  if (argc < 1) {
    rb_raise(rb_eArgError, "Need 1 args, not %d", argc);
  }

  n       = NUM2INT(argv[0]);
  i       = 1;
  prev    = 1;
  current = 1;
  arr     = rb_ary_new2(0);

  while (i < n)
  {
    temp = current;
    current = current + prev;
    prev = temp;
    rb_funcall(arr, rb_intern("push"), 1 ,INT2NUM(prev));
    i = i + 1;
  }

  return arr;
}

void Init_a ()
{
  VALUE cls_Fibonnaci;

  cls_Fibonnaci = rb_define_class("Fibonnaci", rb_cObject);

  rb_define_method(cls_Fibonnaci ,"compute", Fibonnaci_compute, -1);
}
```



```ruby
class Fibonnaci
  def compute(int n)
    int i = 1, prev = 1, current = 1, temp
    array = []

    while i < n do
      temp = current
      current = current + prev
      prev = temp
      array.push(prev)
      i += 1
    end

    return array
  end
end
```

https://github.com/SciRuby/rubex

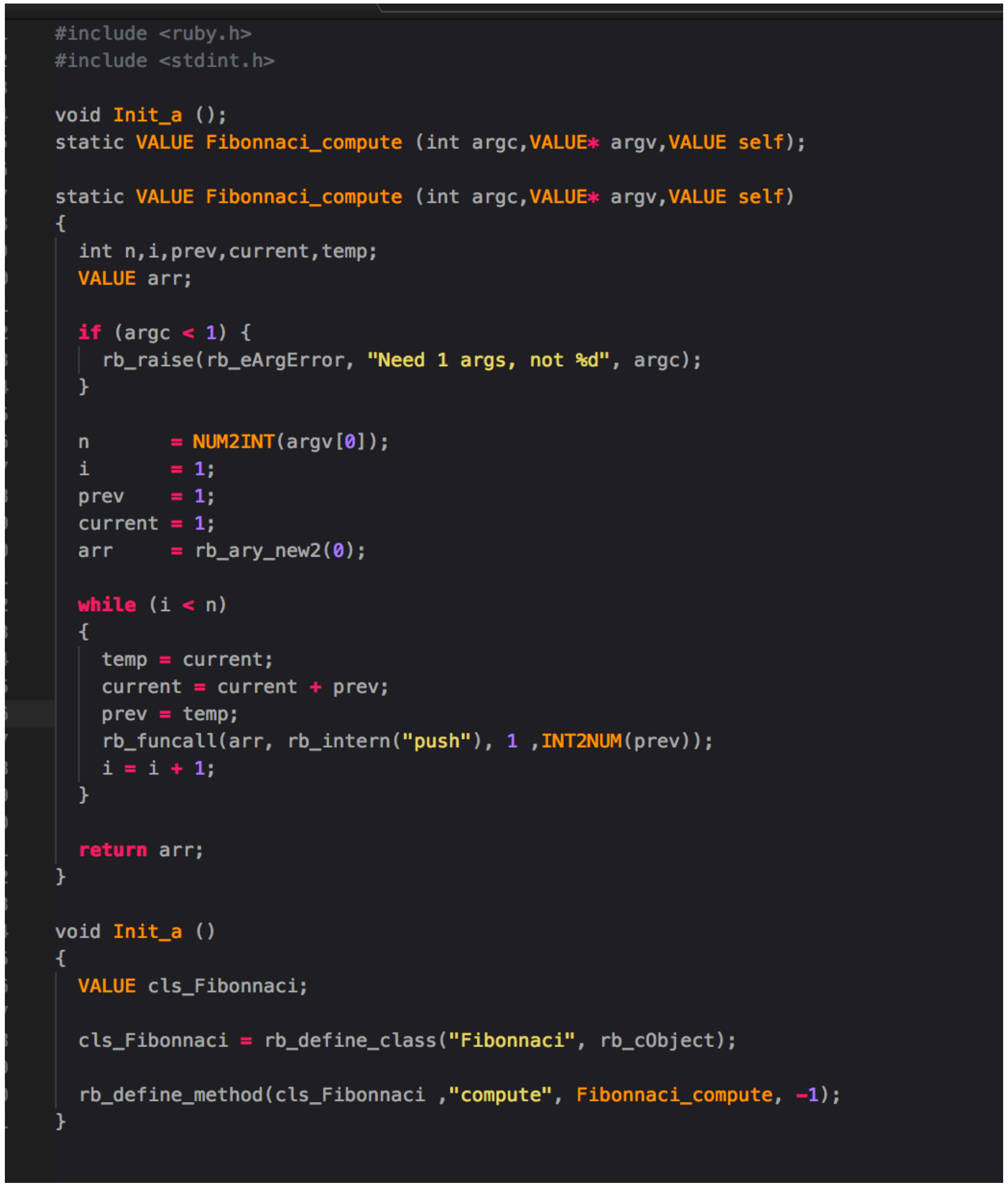
Rubex



```ruby
class Fibonnaci
  def compute(n)
    i = 1, prev = 1, current = 1, temp
    array = []

    while i < n do
      temp = current
      current = current + prev
      prev = temp
      array.push(prev)
      i += 1
    end

    return array
  end
end
```

https://github.com/SciRuby/rubex

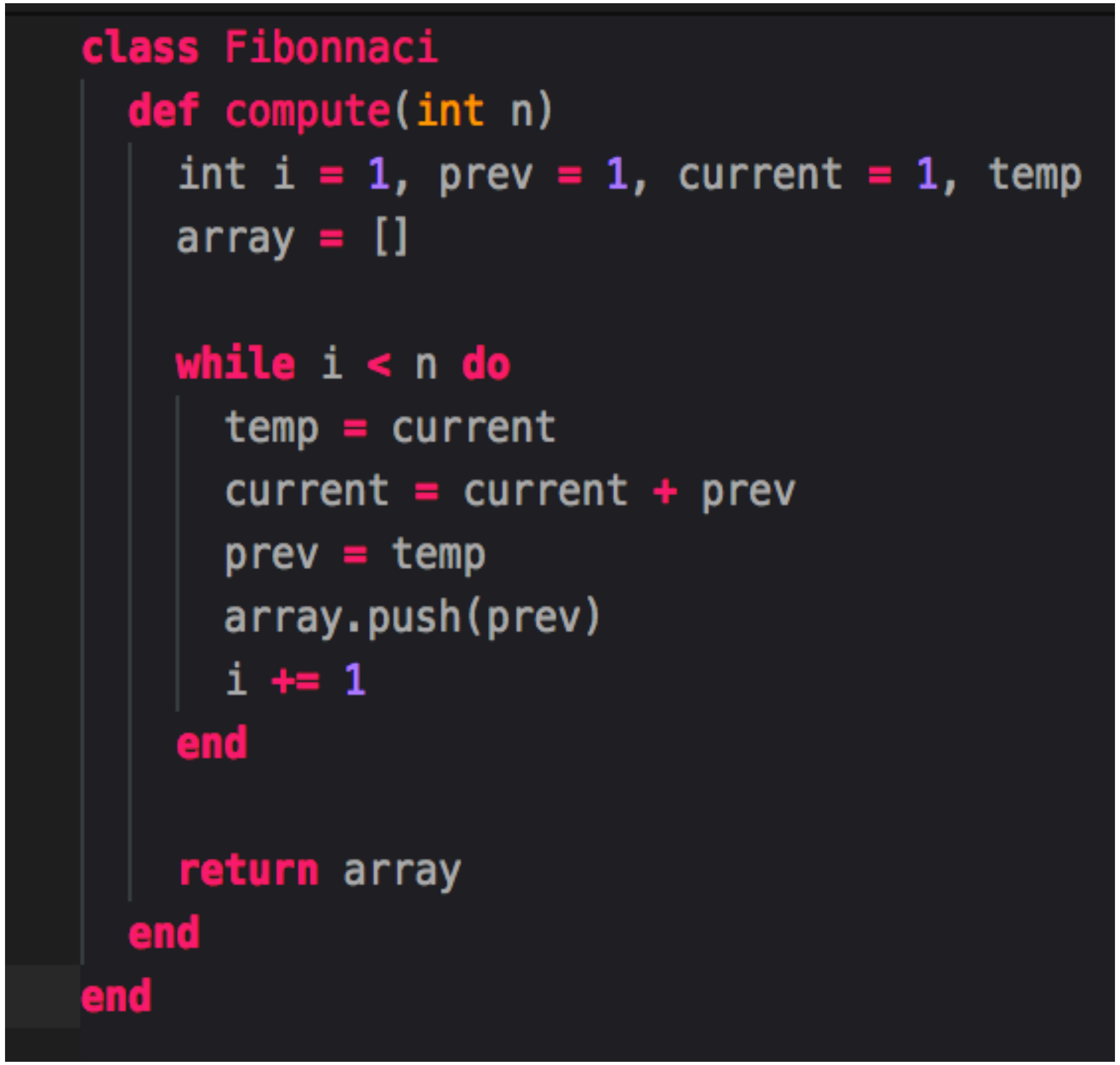
Rubex



```c
#include <ruby.h>
#include <stdint.h>

void Init_a ();
static VALUE Fibonnaci_compute (int argc,VALUE* argv,VALUE self);

static VALUE Fibonnaci_compute (int argc,VALUE* argv,VALUE self)
{
  int n,i,prev,current,temp;
  VALUE arr;

  if (argc < 1) {
    rb_raise(rb_eArgError, "Need 1 args, not %d", argc);
  }

  n       = NUM2INT(argv[0]);
  i       = 1;
  prev    = 1;
  current = 1;
  arr     = rb_ary_new2(0);

  while (i < n)
  {
    temp = current;
    current = current + prev;
    prev = temp;
    rb_funcall(arr, rb_intern("push"), 1 ,INT2NUM(prev));
    i = i + 1;
  }

  return arr;
}

void Init_a ()
{
  VALUE cls_Fibonnaci;

  cls_Fibonnaci = rb_define_class("Fibonnaci", rb_cObject);

  rb_define_method(cls_Fibonnaci ,"compute", Fibonnaci_compute, -1);
}
```

https://github.com/SciRuby/rubex

Rubex



```
class Fibonnaci
  def compute(int n)
    int i = 1, prev = 1, current = 1, temp
    array = []

    while i < n do
      temp = current
      current = current + prev
      prev = temp
      array.push(prev)
      i += 1
    end

    return array
  end
end
```

https://github.com/SciRuby/rubex

Some closing thoughts

- Large data set, high performance requirements -> Pandas/Quattro
- Prototyping, native Ruby -> Daru
- Pandas started 10 years behind R!
- Get involved!

Thank you RubyConf MY

Daniel Baark

https://github.com/baarkerlounger

Acknowledgments:

ZappiStore – (@brendon9x et al.)

SciRuby – (@v0dro, @zverok, @lokesh et al.)

SciPy – (@wesm et al.)