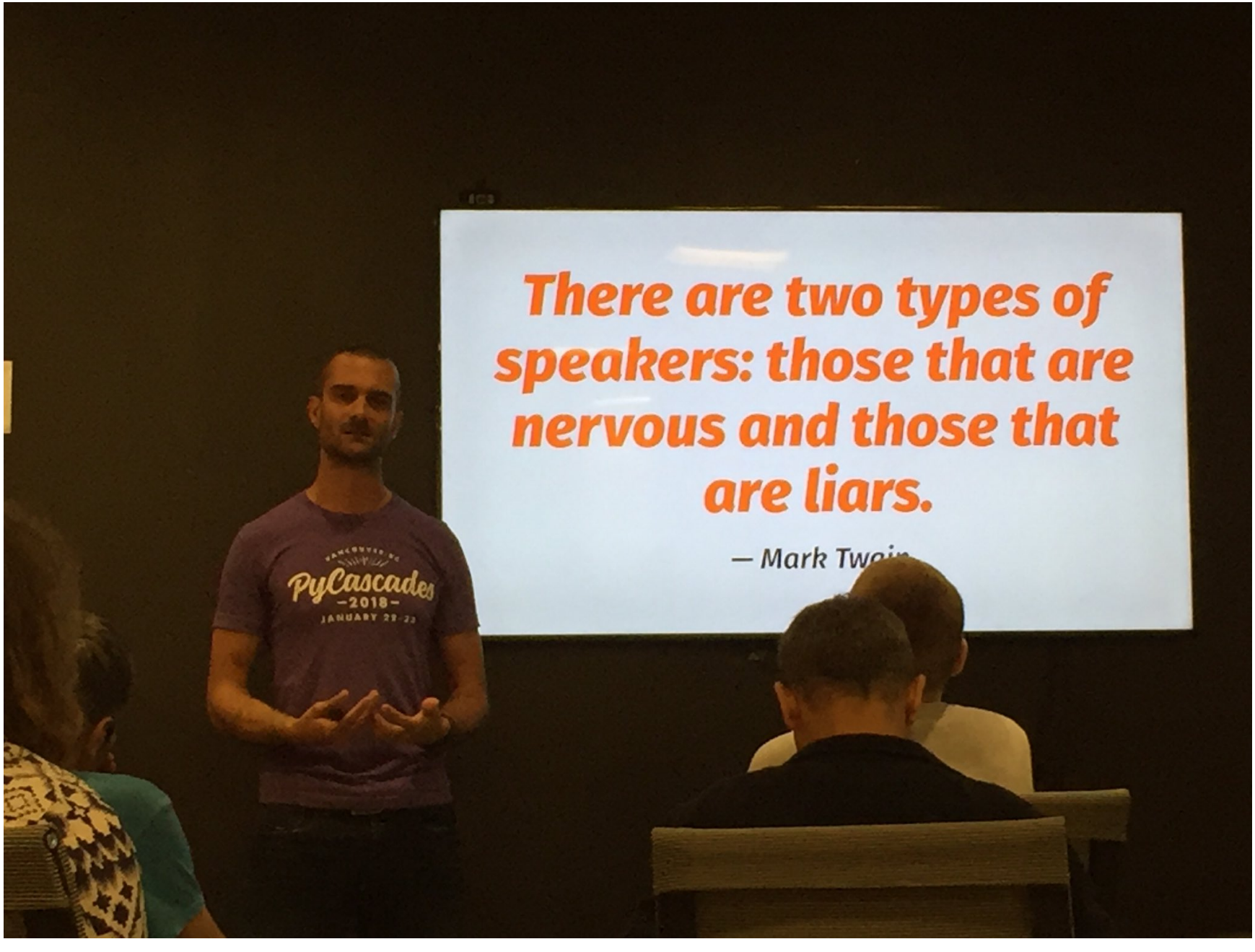
The Modern Prometheus

Piotr Szotkowski



Dan Bader's photo of Sebastian Vetter's talk

Jerzy Strzelecki, Schloss Frankenstein

FRANKENSTEIN;

OR,

THE MODERN PROMETHEUS.

IN THREE VOLUMES.

(Darmstadt, Hessen)

Frankenstein Programming

as in

~seamlessly combining more than one language

John T. McInnes
@lyebot

Common misconception; "Big Ben" is actually the name of the clock's creator. Its correct name is "Big Ben's Monster"

Synthetic benchmarks tell us sweet FA about real world performance of code, **architecture** being a much more significant consideration than the proportion of raw MIPS a given language will deliver on a given platform.
The average netbook could happily run all of Teller's fusion bomb models along with the full telemetry analysis of all the Apollo missions in the pauses between loading XKCD comics and binning junk mail without the user being any the wiser.

— Eleanor McHugh

```ruby
require 'active_record'
require 'benchmark/ips'

ActiveRecord::Base.establish_connection adapter: 'postgresql'

class CreateWhatevers < ActiveRecord::Migration[5.2]
  def change
    create_table(:whatevers) { |table| table.text :text }
  end
end

CreateWhatevers.migrate :up
at_exit { CreateWhatevers.migrate :down }

Whatever = Class.new(ActiveRecord::Base)

Benchmark.ips do |bench|
  bench.report('exception miss') { raise '💥' if false          }
  bench.report('exception hit')  { raise '💥' rescue nil        }
  bench.report('SELECT')         { Whatever.first               }
  bench.report('INSERT')         { Whatever.create(text: 'meh') }
  bench.compare!
end
```

```
Warming up --------------------------------------
      exception miss   445.207k i/100ms
       exception hit   107.302k i/100ms
              SELECT   546.000  i/100ms
              INSERT    23.000  i/100ms
Calculating -------------------------------------
      exception miss     13.660M (± 3.6%) i/s -     68.562M in   5.026944s
       exception hit      1.375M (± 2.3%) i/s -      6.975M in   5.074398s
              SELECT      5.108k (± 4.5%) i/s -     25.662k in   5.034214s
              INSERT    241.202  (± 9.1%) i/s -      1.219k in   5.090564s

Comparison:
      exception miss: 13660293.4 i/s
       exception hit:  1375266.0 i/s - 9.93x  slower
              SELECT:     5108.2 i/s - 2674.19x  slower
              INSERT:      241.2 i/s - 56634.16x  slower
```

Isaac Sloan
@elorest

968,824.35 req/sec: 32 cores at 2.7Ghz #AmberFramework on @CrystalLanguage. I love @rubyonrails but it and @elixirphoenix seem like turtles.

```
ubuntu@ip-172-31-0-70:~/bench$ wrk -d 60 -t 20 -c 1015 http://localhost:3000
Running 1m test @ http://localhost:3000
  20 threads and 1015 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     1.86ms    2.88ms  56.61ms   87.54%
    Req/Sec    48.73k     6.01k   88.40k    68.28%
  58225168 requests in 1.00m, 4.01GB read
Requests/sec: 968824.35
Transfer/sec:     68.37MB
```

Tweet Quotes
@QuoteTwitts

If we have data, let's look at data. If all we have are opinions, let's go with mine. 👄 Jim Barksdale #Quotes

1. ALWAYS BENCHMARK

2. ADJUST ALGORITHM

3. IMPLEMENTATIONS?

(BUT IF YOU DID
AND IT'S *STILL* TOO SLOW
IT MIGHT BE WORTHWHILE
TO LOOK INTO OTHER LANGUAGES...)

Jamie Gaskins
@jamie_gaskins

Pivotal is the company that created the story-tracking software. The actual name of the software is Pivotal's Monster.

BINARY REPRESENTATION

LET'S HAVE A SET OF ELEMENTS (E.G., INTEGERS)

LET'S REPRESENT IT BY 'SET' BITS
(ONES IN ITS BINARY REPRESENTATION)

$\{0,2,3,5\} \rightarrow 0b101101$

SET OPERATIONS ARE NOW BINARY OPERATIONS

$\{0,2,3,5\} \cap \{2,3,4\} \rightarrow 0b101101 \;\&\; 0b011100$

$0b101101 \;\&\; 0b011100 \rightarrow 0b001100 \rightarrow \{2,3\}$

POPULATION COUNT

LET'S SAY WE NEED THE SIZE OF OUR SET

NUMBER OF ONES IN BINARY REPRESENTATION

```
|{0,2,3,5}| → popcount(0b101101) → 4
  |{2,3,4}| → popcount(0b011100) → 3
    |{2,3}| → popcount(0b001100) → 2
```

LET'S FREEDOM-PATCH Integer

```ruby
class Integer

  def popcount_to_s
    to_s(2).count('1')
  end

  def popcount_cont_shift
    count  = 0
    number = self
    until number.zero?
      count += number & 1
      number >>= 1
    end
    count
  end

end
```

```ruby
class Integer

  def popcount_bit_elim
    count  = 0
    number = self
    until number.zero?
      number &= number - 1
      count += 1
    end
    count
  end

  def popcount_prog_shift
    number  = self
    number -= (number >> 1) & 0x5555555555555555
    number  = (number & 0x3333333333333333) + ((number >> 2) & 0x3333333333
    number  = (number + (number >>  4)) & 0x0f0f0f0f0f0f0f0f
    number  = (number + (number >>  8)) & 0x00ff00ff00ff00ff
    number  = (number + (number >> 16)) & 0x0000ffff0000ffff
    number += number >> 32
    (number + (number >> 64)) & 0xff
  end

end
```

Rust via Helix? sure!

```
#[macro_use]
extern crate helix;

ruby! {
    class Popcount {
        def count(int: u64) -> u32 {
            int.count_ones()
        }
    }
}
```

C? SSE4.2? WHY NOT!

```
class Integer

  inline do |builder|
    builder.c 'int popcount_bit_elim_c() {
      long number = NUM2LONG(self);
      int  count;
      for (count = 0; number; count++) number &= number - 1;
      return count;
    }'
  end

  inline do |builder|
    builder.c 'int popcount_builtin() {
      return __builtin_popcountl(NUM2LONG(self));
    }'
  end

end
```

ALGORITHMS? APPROACH!

```ruby
class Integer

  POPCOUNT_CACHE = (0x0000..0xffff).map { |number| number.to_s(2).count('1')

  def popcount_cached
    POPCOUNT_CACHE[self & 0xffff] +
    POPCOUNT_CACHE[self >> 16 & 0xffff] +
    POPCOUNT_CACHE[self >> 32 & 0xffff] +
    POPCOUNT_CACHE[self >> 48]
  end

end
```

LET'S BENCH! THAT INTROSPECTION!

```ruby
require 'benchmark/ips'

numbers = Array.new(1_000) { rand(2**62) }
methods = Integer.instance_methods.grep(/^popcount_/)

raise 'oops' unless methods.map { |meth| numbers.map(&meth) }.uniq.size == 1

Benchmark.ips do |bench|
  methods.each do |meth|
    bench.report(meth[9..-1]) { numbers.map(&meth) }
  end
  bench.compare!
end
```

```
Ruby 2.5:
            builtin:    29915.7 i/s
          bit_elim_c:    17011.6 i/s - 1.76x  slower
              cached:     4874.4 i/s - 6.14x  slower
                rust:     3133.3 i/s - 9.55x  slower
          prog_shift:     2878.5 i/s - 10.39x  slower
                to_s:     1131.3 i/s - 26.44x  slower
            bit_elim:      714.5 i/s - 41.87x  slower
          cont_shift:      317.3 i/s - 94.27x  slower

JRuby 9.1:
              cached:     5700.0 i/s
          prog_shift:     4894.2 i/s - 1.16x  slower
            bit_elim:     1337.3 i/s - 4.26x  slower
                to_s:      870.2 i/s - 6.55x  slower
          cont_shift:      716.2 i/s - 7.96x  slower

JRuby 9.1 + 30 s warm-up + invoke dynamic:
          prog_shift:     6749.9 i/s
              cached:     6566.6 i/s - same-ish: difference falls within error
            bit_elim:     2587.5 i/s - 2.61x  slower
          cont_shift:     1644.5 i/s - 4.10x  slower
                to_s:      855.6 i/s - 7.89x  slower
```

```
require "benchmark"

struct Int
  POPCOUNT_CACHE = (0x0000..0xffff).map { |number| number.to_s(2).count('1')
  def popcount_bit_elim    # …
  def popcount_cached      # …
  def popcount_cont_shift  # …
  def popcount_prog_shift  # …
  def popcount_to_s        # …
end

numbers = Array.new(1_000) { rand(2u64**62) }

Benchmark.ips do |bench|
  bench.report("popcount")    { numbers.map(&.popcount)             }
  bench.report("prog_shift") { numbers.map(&.popcount_prog_shift) }
  bench.report("cached")      { numbers.map(&.popcount_cached)      }
  bench.report("bit_elim")    { numbers.map(&.popcount_bit_elim)    }
  bench.report("cont_shift") { numbers.map(&.popcount_cont_shift) }
  bench.report("to_s")        { numbers.map(&.popcount_to_s)        }
end
```

⚠ Ruby times include Ruby-side call overhead ⚠

```
Ruby 2.5 (+ Rust, C, SSE4.2…):
            builtin:    29915.7 i/s
          bit_elim_c:    17011.6 i/s - 1.76x  slower
              cached:     4874.4 i/s - 6.14x  slower
                rust:     3133.3 i/s - 9.55x  slower
          prog_shift:     2878.5 i/s - 10.39x  slower
                to_s:     1131.3 i/s - 26.44x  slower
            bit_elim:      714.5 i/s - 41.87x  slower
          cont_shift:      317.3 i/s - 94.27x  slower

Crystal 0.26:
  popcount 267.47k (  3.74µs) (± 1.13%)   8048 B/op        fastest
prog_shift 211.22k (  4.73µs) (± 1.26%)   8048 B/op   1.27× slower
    cached 149.92k (  6.67µs) (± 3.60%)   4048 B/op   1.78× slower
  bit_elim  33.96k ( 29.45µs) (± 1.12%)   4048 B/op   7.88× slower
cont_shift  26.54k ( 37.68µs) (± 0.90%)   4048 B/op  10.08× slower
      to_s   8.14k (122.78µs) (± 3.15%)  84159 B/op  32.84× slower
```

Lord Pinky
@HiddenPinky

"Cookie" is the name of the scientist who created him. It's Cookie's Monster, not "Cookie Monster". Read a book sometime.

ESCAPE ENTITIES

MAKE TEXT 'HTML SAFE'

< → <

> → >

& → &

' → '

" → "

```ruby
class String

  ENTITIES = { '<' => '&lt;', '>' => '&gt;', '&' => '&amp;',
               "'" => '&apos;', '"' => '&quot;' }

  def escape_each_char
    ''.tap do |result|
      each_char { |char| result << ENTITIES.fetch(char, char) }
    end
  end

  def escape_gsub
    gsub(/[<>&'"]/, ENTITIES)
  end

  def escape_cgi
    CGI.escape_html(self)
  end

end
```

ZERO-COST ABSTRACTIONS... 

```
#[macro_use]
extern crate helix;

ruby! {
    class HTML {
        def map_escape(input: String) -> String {
            input.chars().map(|chr| {
                match chr {
                    '<'  => String::from("&lt;"),
                    '>'  => String::from("&gt;"),
                    '&'  => String::from("&amp;"),
                    '\'' => String::from("&apos;"),
                    '"'  => String::from("&quot;"),
                    _    => chr.to_string(),
                }
            }).collect()
        }
    }
}
```

```
#[macro_use]
extern crate helix;

ruby! {
    class HTML {
        def push_escape(input: String) -> String {
            let mut result = String::with_capacity(2 * input.len());
            for chr in input.chars() {
                match chr {
                    '<'  => result.push_str("&lt;"),
                    '>'  => result.push_str("&gt;"),
                    '&'  => result.push_str("&amp;"),
                    '\'' => result.push_str("&apos;"),
                    '"'  => result.push_str("&quot;"),
                    _    => result.push(chr),
                }
            }
            result
        }
    }
}
```

```
Ruby 2.5:
           rust_push:   286450.1 i/s
                cgi:   128455.5 i/s - 2.23x  slower
               gsub:    13743.6 i/s - 20.84x  slower
            rust_map:     7878.5 i/s - 36.36x  slower
           each_char:     4540.9 i/s - 63.08x  slower

JRuby 9.1:
                cgi:   122654.7 i/s
               gsub:    18363.9 i/s - 6.68x  slower
           each_char:     5596.8 i/s - 21.92x  slower

JRuby 9.1 + 30 s warm-up + invoke dynamic:
                cgi:   121956.6 i/s
               gsub:    17826.9 i/s - 6.84x  slower
           each_char:     6063.1 i/s - 20.11x  slower
```

Meanwhile in Crystal...

```
class String

  ENTITIES = { '<' => "&lt;", '>' => "&gt;", '&' => "&amp;",
               '\'' => "&apos;", '"' => "&quot;" }

  def escape_gsub
    gsub(ENTITIES)
  end

  def escape_html
    HTML.escape(self)
  end

  def escape_io
    io = IO::Memory.new
    HTML.escape(self, io)
    io.to_s
  end
```

```
Crystal 0.23 (2017):
  io  67.56k (  14.8µs) (± 1.38%)       fastest
gsub  60.62k (  16.5µs) (± 2.13%)  1.11× slower
html  57.82k ( 17.29µs) (± 1.58%)  1.17× slower

Crystal 0.24 (2018):
  io  35.84k (  27.9µs) (± 2.84%)  1.18× slower
gsub  42.28k ( 23.65µs) (± 1.05%)  1.00× slower
html  42.36k ( 23.61µs) (± 1.34%)       fastest

Crystal 0.26 (2018):
  io  41.23k ( 24.26µs) (± 2.09%)  10816 B/op   1.15× slower
gsub  47.54k ( 21.03µs) (± 2.33%)   9824 B/op        fastest
html  46.49k ( 21.51µs) (± 1.64%)   9744 B/op   1.02× slower

Ruby 2.5:
           rust_push:   286450.1 i/s
                cgi:   128455.5 i/s - 2.23x  slower
               gsub:    13743.6 i/s - 20.84x  slower
           rust_map:     7878.5 i/s - 36.36x  slower
          each_char:     4540.9 i/s - 63.08x  slower
```

FOR SUB-μS TIMES: OPTIMISING STRING PROCESSING IN RUST

andrew dongs
@mustlovedongs

[explaining the theory of relativity to physics students] no no no, einstein is the name of its creator, you mean einstein's monster

BOUNDARY CROSSING

NOT PASSING ANYTHING

BOOLEANS

NUMBERS (> 2^{62}? BigDecimals?)

STRINGS (LONGER THAN 23 BYTES?)

ARRAYS (...OF WHAT? CONTAINING self?)

HASHES (...OF WHAT ⟶ WHAT? WITH self KEY?)

STRUCTS | OBJECTS

Ruby (franken)gems

json

JSON::Pure (Ruby) vs JSON::Ext (C | Java)

levenshtein

C with transparent fallback to Ruby

two Strings | two Arrays | two [#hash + #eql?]#each

fast_blank

faster String#blank?

Settle In, Babies
@alexandraerin

Actually "gif" is the name of the creator. The file format is "gif's file format".

World Domination Plans

1. Benchmark 2. Algorithm 3. Implementations?

Readability, performance, dynlang call overhead

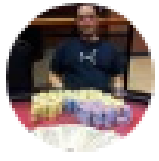
Read the source! (esp. for self-hosted languages)

Never create Ruby strings longer than 23 characters

Writing Ruby gems with Rust and Helix

How to write Ruby extensions with Crystal

Ruby and Go Sitting in a Tree



Andrew Heister
@andrew_heister

Actually Frankenstein was the name of the scientist.
I, the person correcting you on this trivial point, am the monster.

rebased

@chastell rebased.com

THANKS!

PIOTR SZOTKOWSKI

talks.chastell.net