What the fork?

Reporting Metrics in a Multi-process Environment

Daniel Magliola

@dmagliola

GOCARDLESS

PromCon EU 2019

What the fork?

Reporting Metrics in a Multi-process Environment

Daniel Magliola

@dmagliola

GOCARDLESS

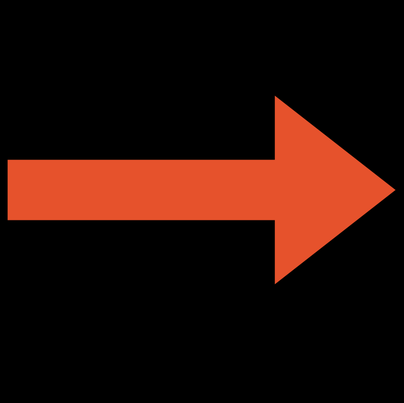
PromCon EU 2019

Hi

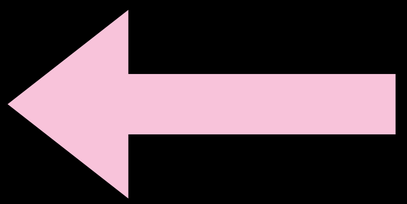
GOCARDLESS

Prometheus


```
http_requests_total[code: 200] = 1000
http_requests_total[code: 500] = 2

payments_total[currency: 'USD'] = 1024
payments_total[currency: 'EUR'] = 128
```



```
http_requests_count[code: 200] = 1000
http_requests_count[code: 500] = 2

payments_count[currency: 'USD'] = 1024
payments_count[currency: 'EUR'] = 128
```

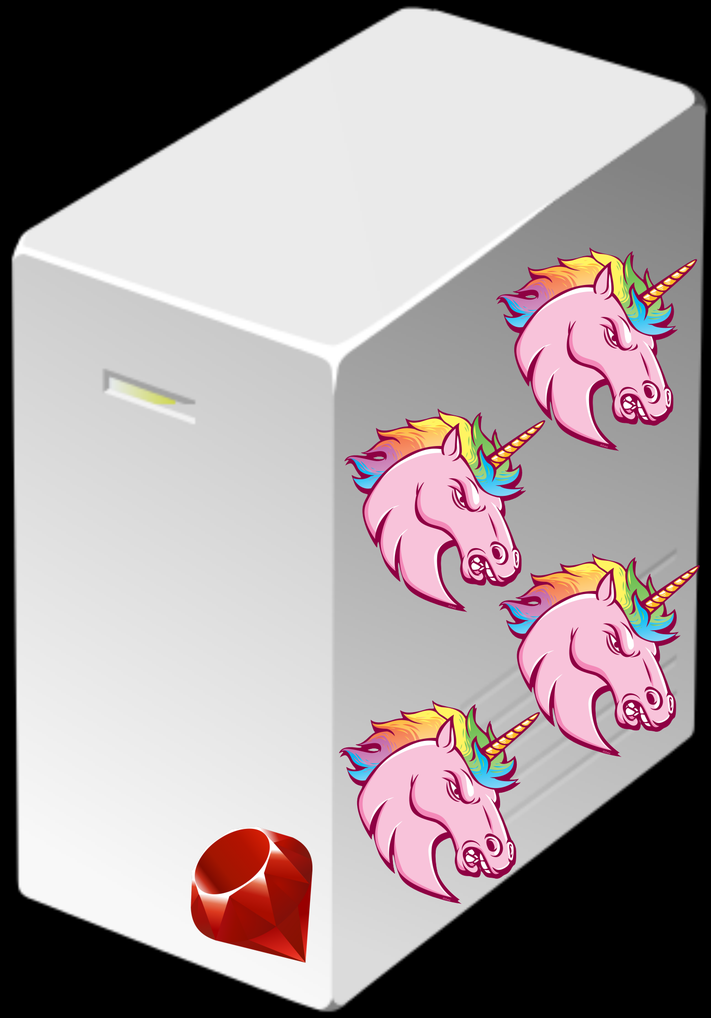


```
http_requests_count{code="200"} 1000
http_requests_count{code="500"} 2
payments_count{currency="USD"} 1024
payments_count{currency="EUR"} 128
```

```
http_requests_count[code: 200] = 1000
http_requests_count[code: 500] = 2

payments_count[currency: 'USD'] = 1024
payments_count[currency: 'EUR'] = 128
```

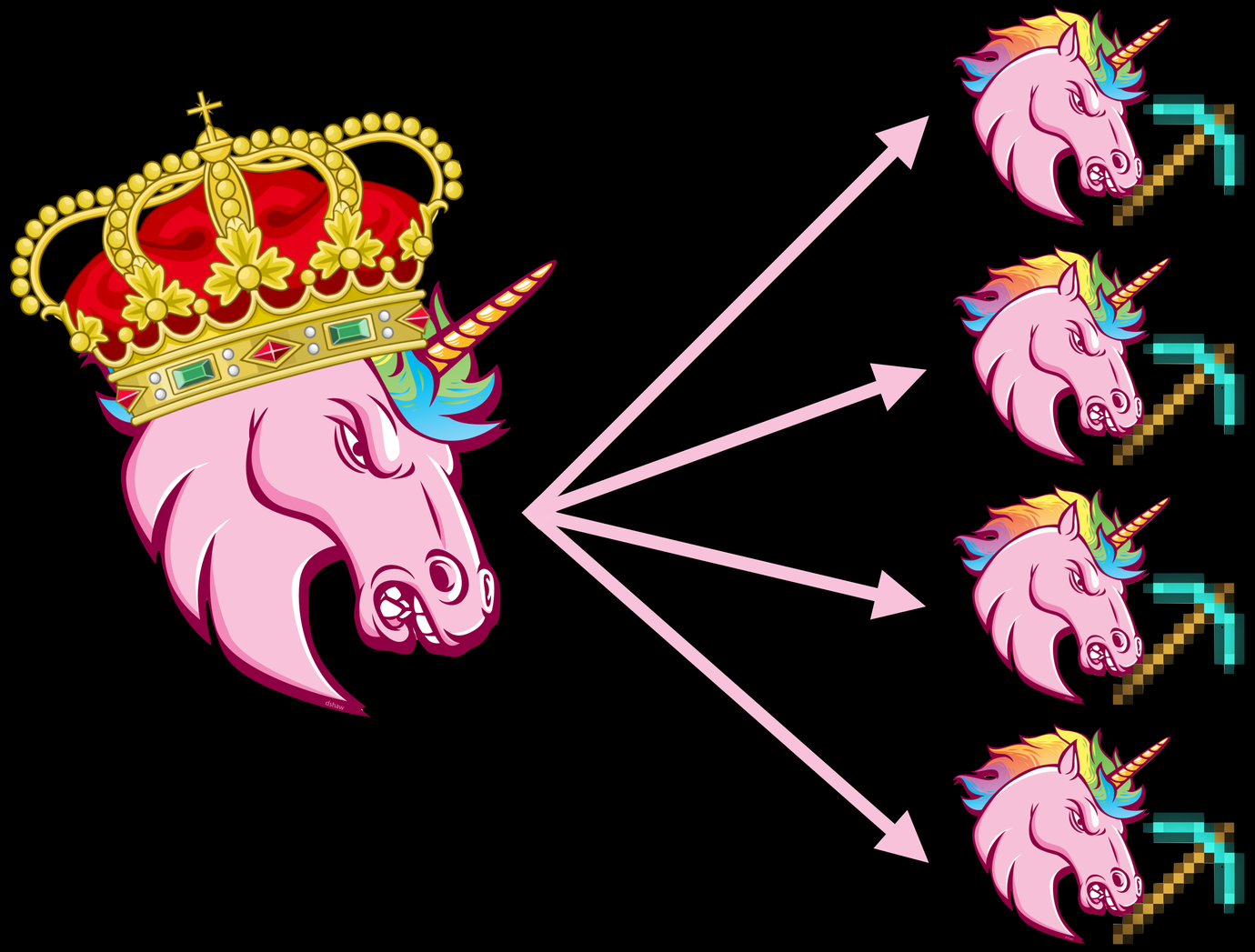
Pre-fork servers

Pre-fork servers

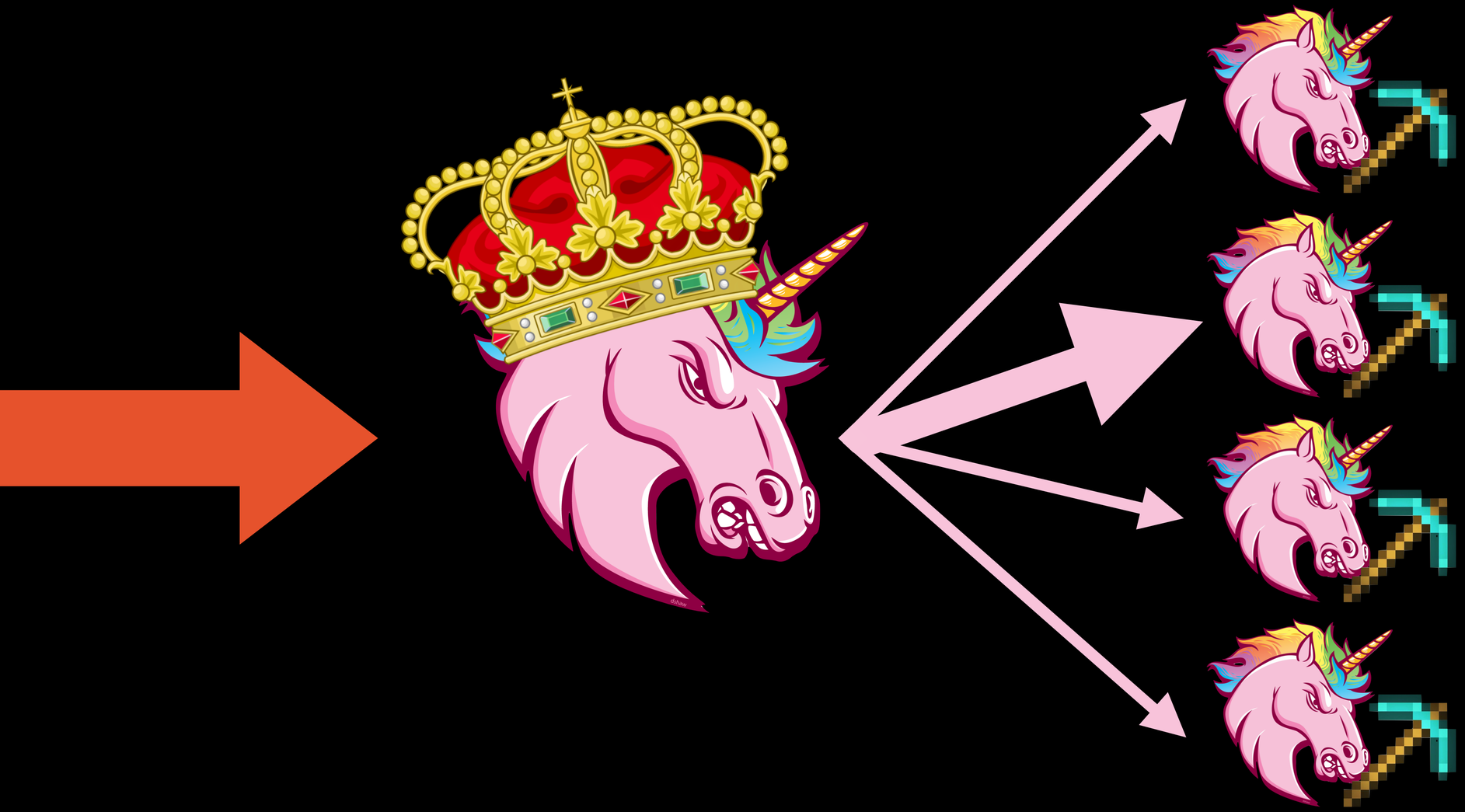


Pre-fork servers

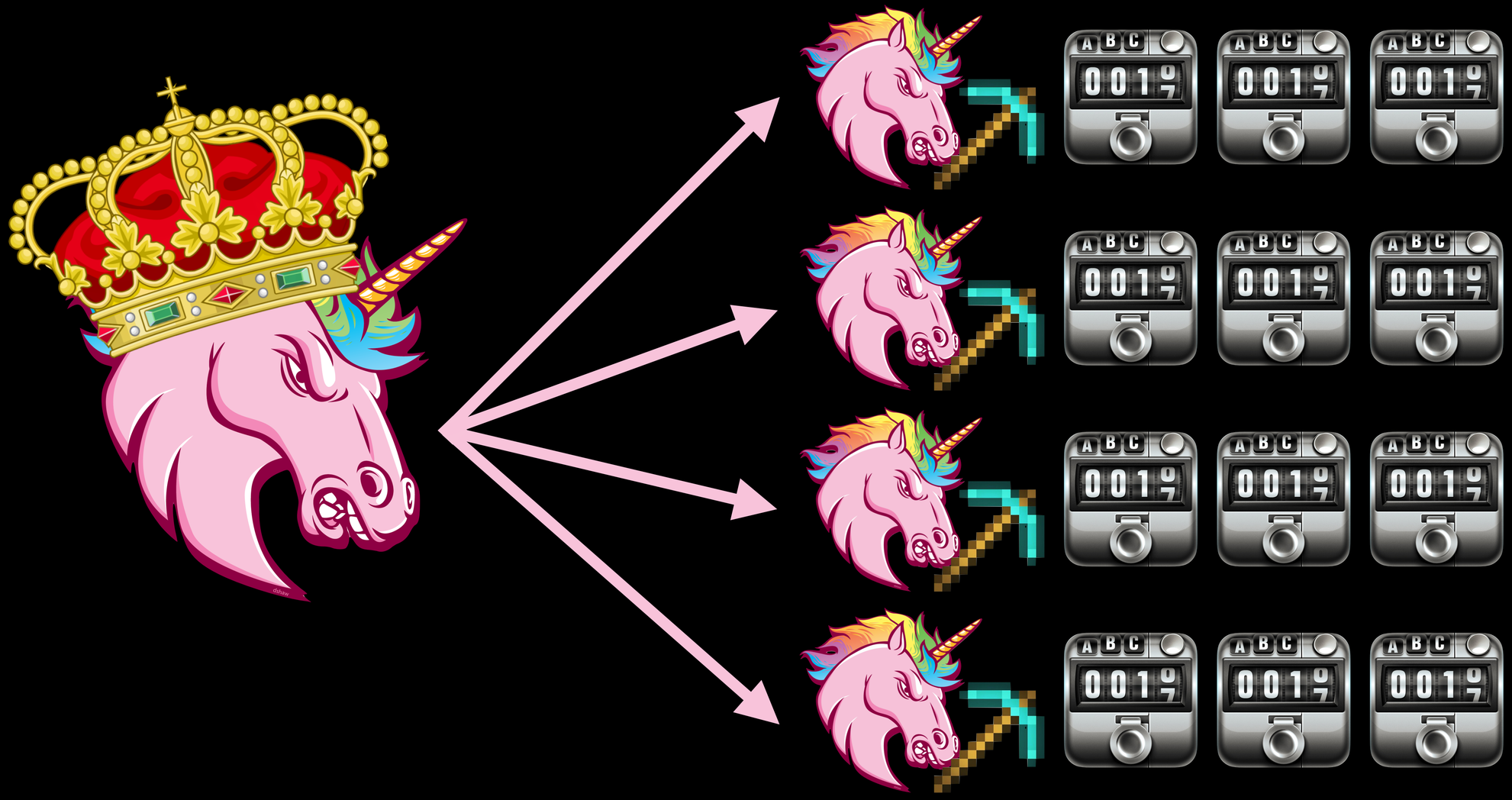
Pre-fork servers



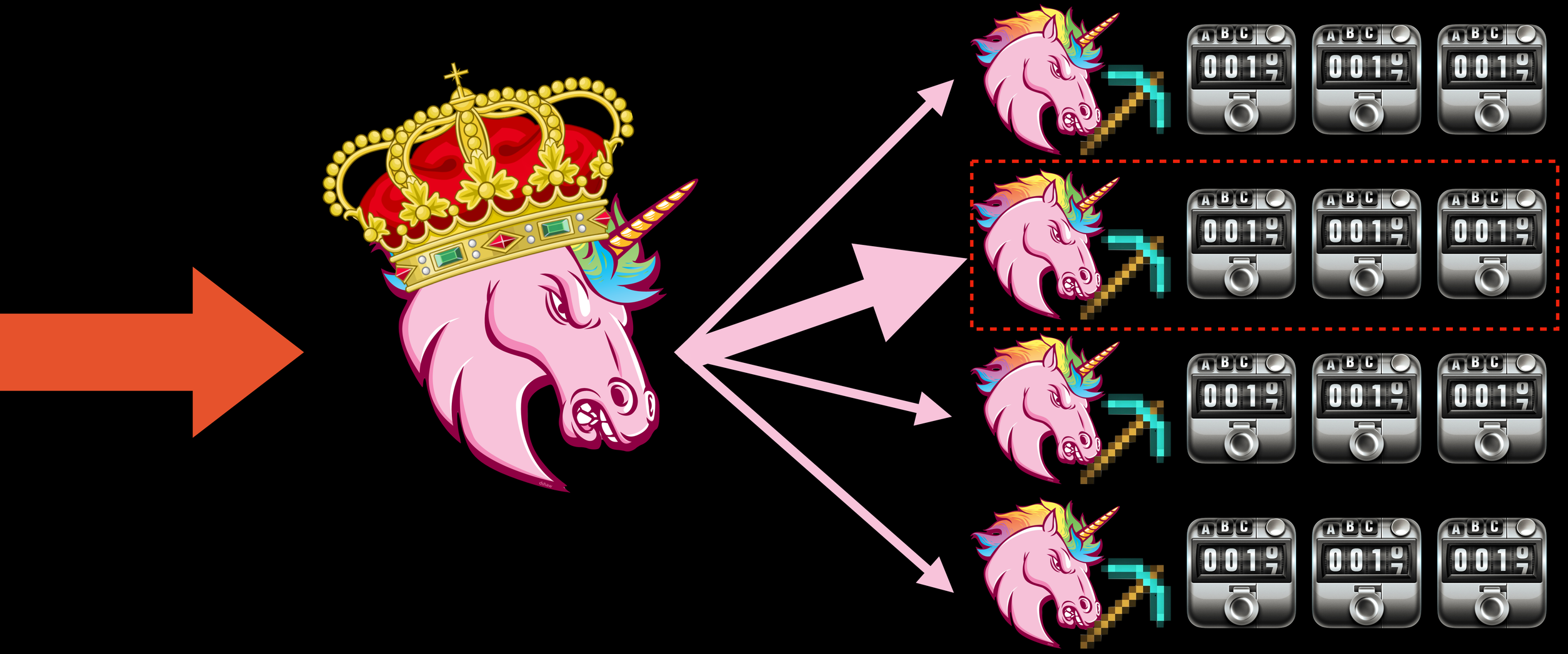
Pre-fork servers



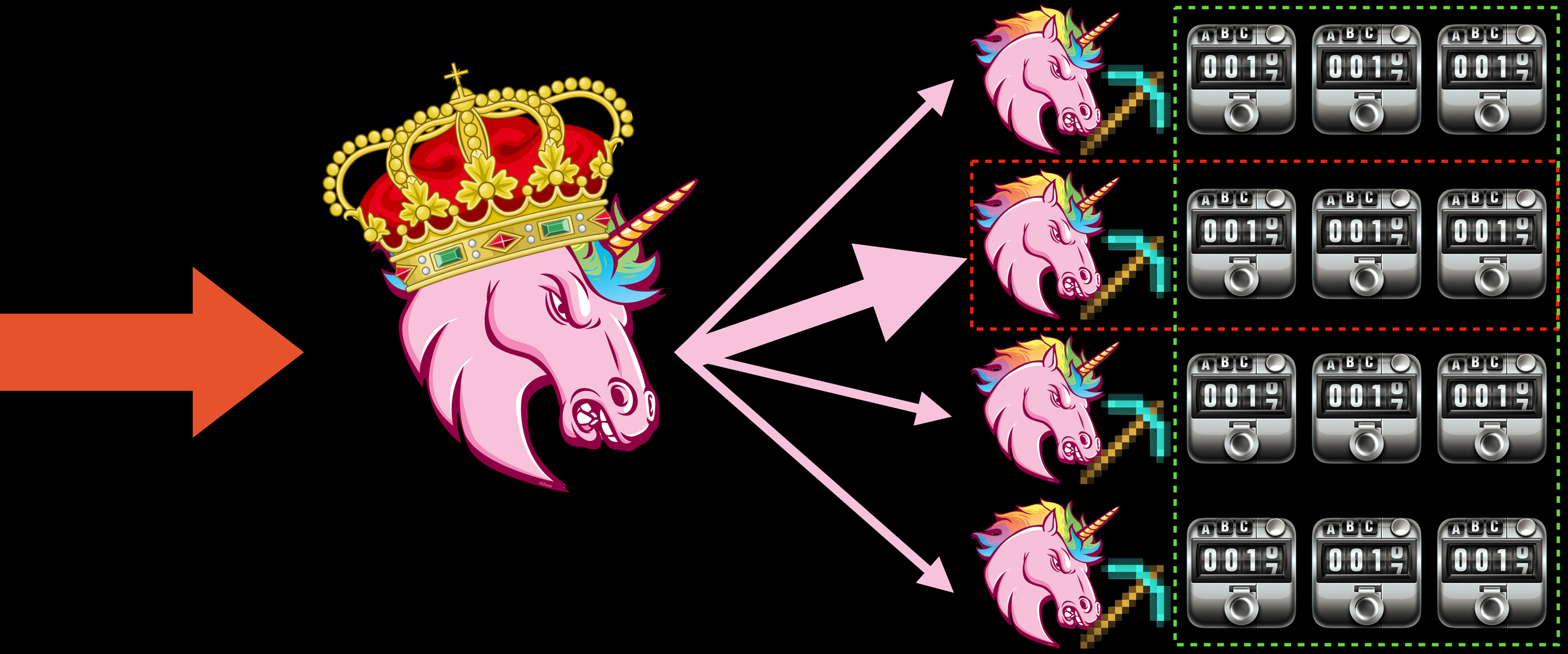
Pre-fork servers



Pre-fork servers

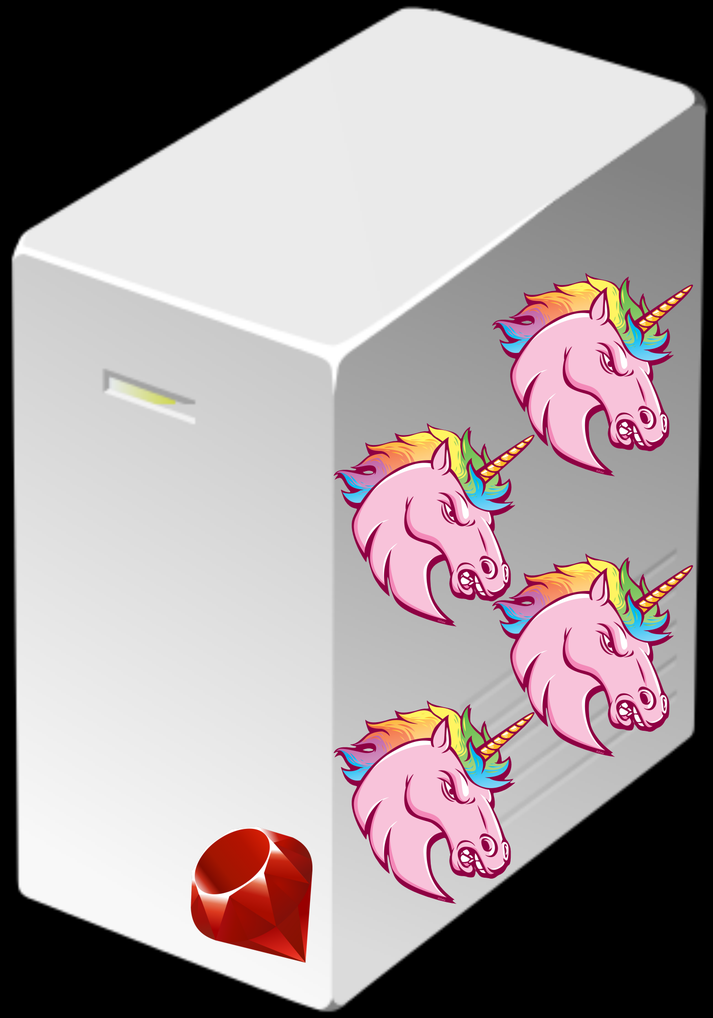


Pre-fork servers

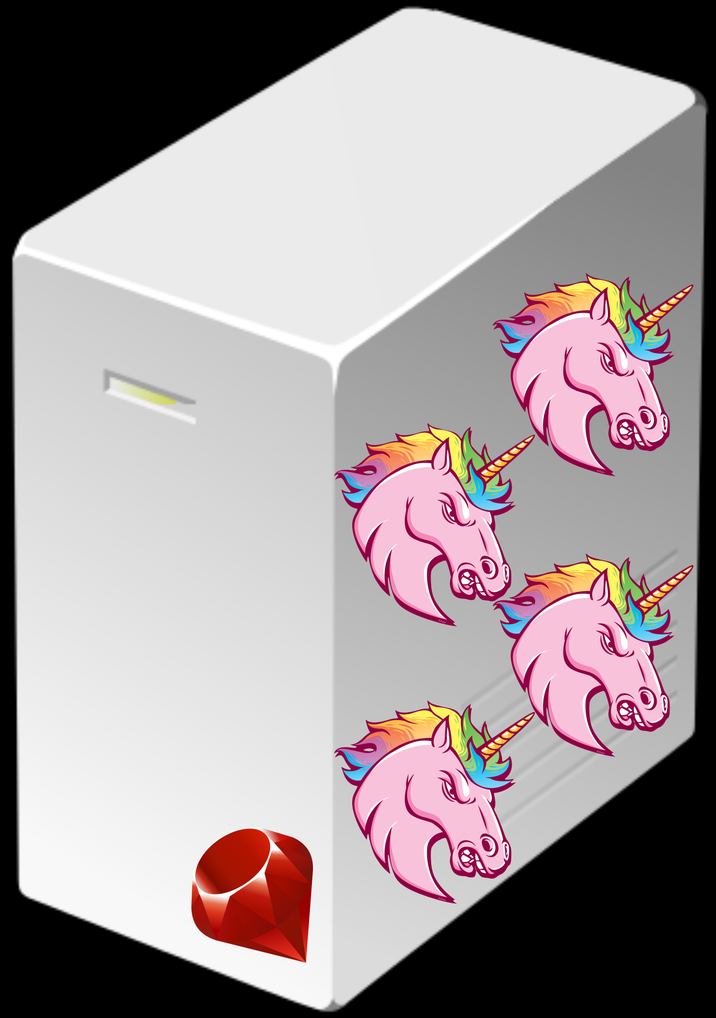


The problem

Everyone:



The problem



The problem

Support pre-fork servers #9

Open **atombender** opened this issue on 8 Feb 2015 77 comments

atombender commented on 8 Feb 2015

If you use this gem with a multi-process Rack server such as Unicorn, surely each worker will be returning just a percentage of the correct results (eg., number of requests served, total time), thus making the exposed metrics fairly meaningless?

To solve this the easiest solution is to create a block of shared memory in the master process that all workers share, instead of using instance variables.

The problem

Support pre-fork servers #9

Open **atombender** opened this issue on 8 Feb 2015 · 77 comments

atombender commented on 8 Feb 2015

If you use this gem with a multi-process Rack server such as Unicorn, surely each worker will be returning just a percentage of the correct results (eg., number of requests served, total time), thus making the exposed metrics fairly meaningless?

To solve this the easiest solution is to create a block of shared memory in the master process that all workers share, instead of using instance variables.

GitLab


Discourse

Performance

bert commented on 17 Oct 2016 **Member**

.468ms is still way to slow. Instrumentation should cost less than a microsecond.

The Python multi-proc solution is at about 1.2us. This is due to Python's mutexes being surprisingly slow (should be 1-2 orders of magnitude faster).

Performance

bert commented on 17 Oct 2016 Member

.468ms is still way to slow. Instrumentation should cost less than a microsecond.

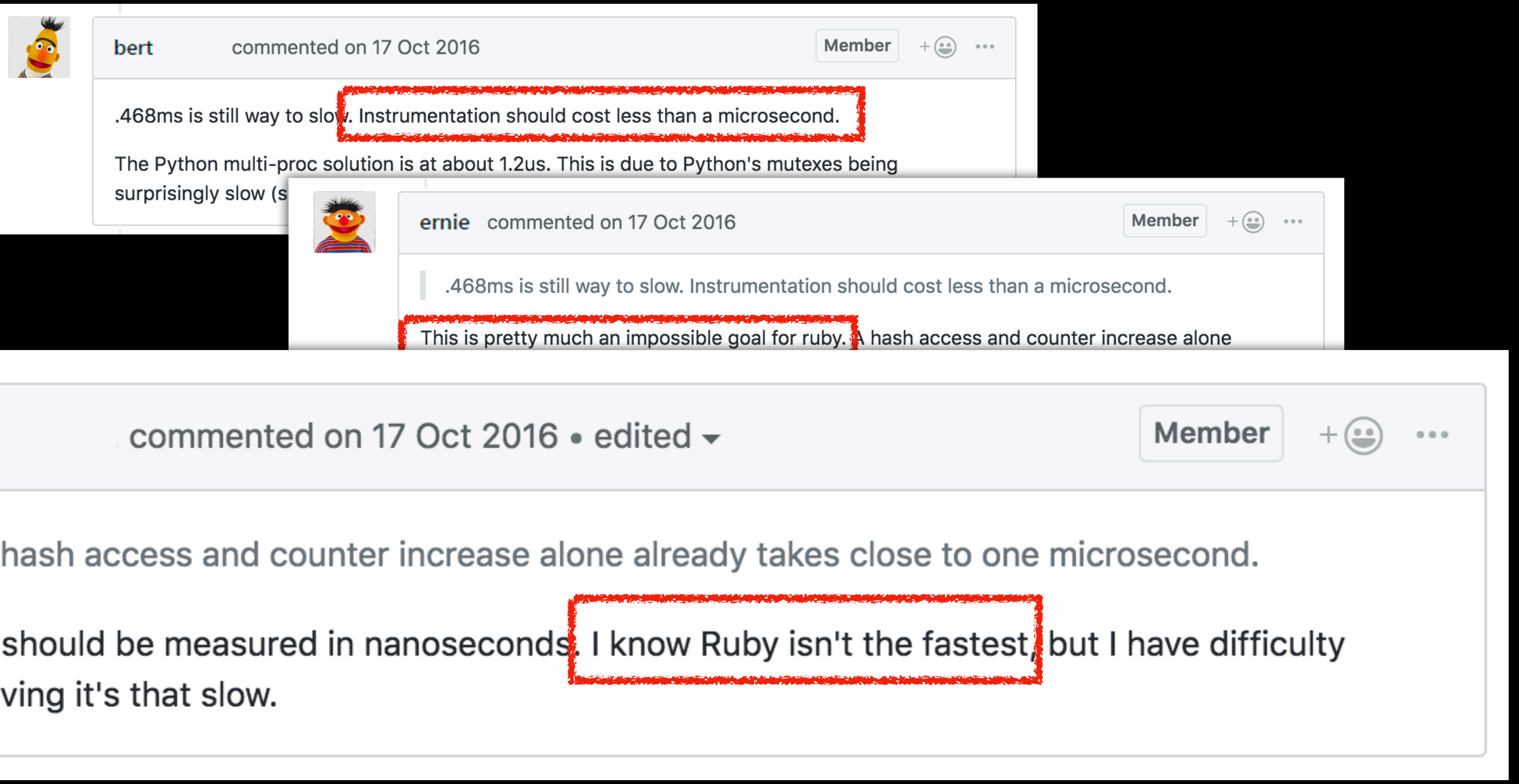
The Python multi-proc solution is at about 1.2us. This is due to Python's mutexes being surprisingly slow (should be 1-2 orders of magnitude faster).

ernie commented on 17 Oct 2016 Member

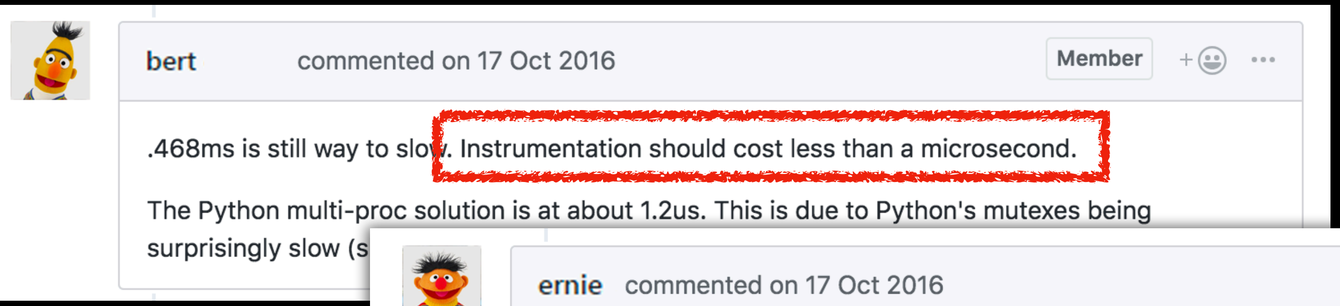
> .468ms is still way to slow. Instrumentation should cost less than a microsecond.

This is pretty much an impossible goal for ruby. A hash access and counter increase alone already takes close to one microsecond. This is without any Mutex involved.

Performance



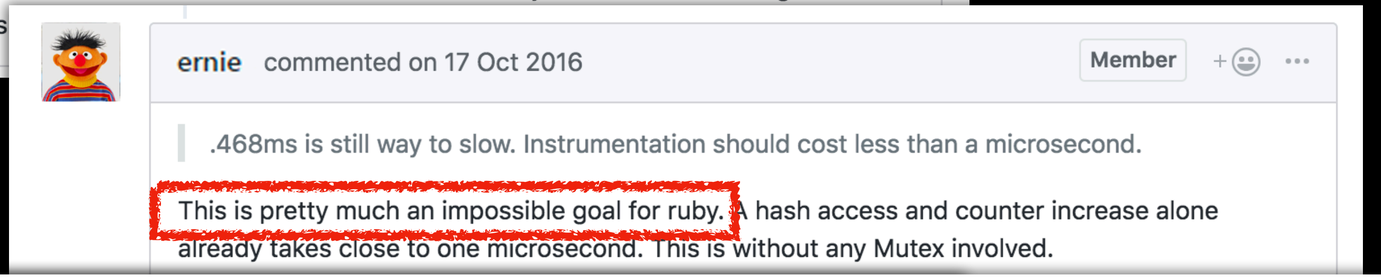
Performance



bert commented on 17 Oct 2016 Member

.468ms is still way to slow. Instrumentation should cost less than a microsecond.

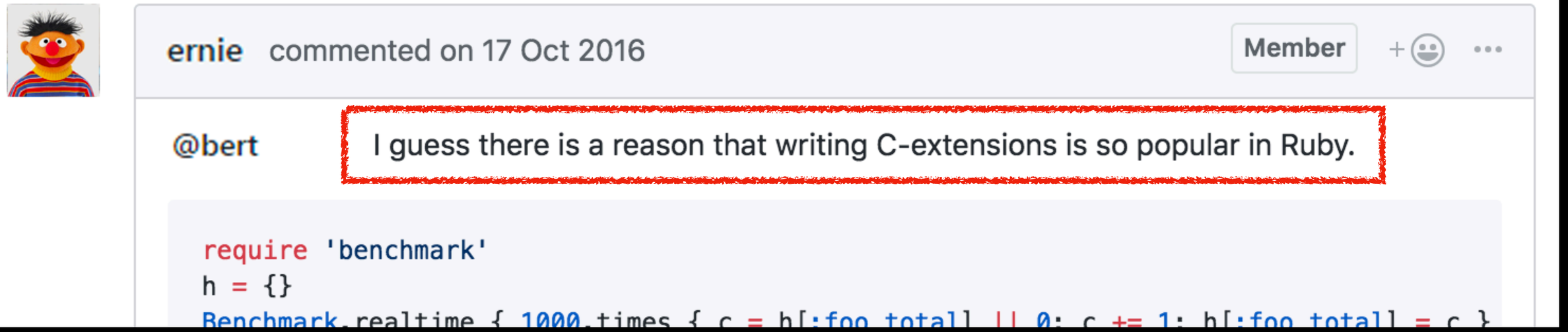
The Python multi-proc solution is at about 1.2us. This is due to Python's mutexes being surprisingly slow (s



ernie commented on 17 Oct 2016 Member

> .468ms is still way to slow. Instrumentation should cost less than a microsecond.

This is pretty much an impossible goal for ruby. A hash access and counter increase alone already takes close to one microsecond. This is without any Mutex involved.

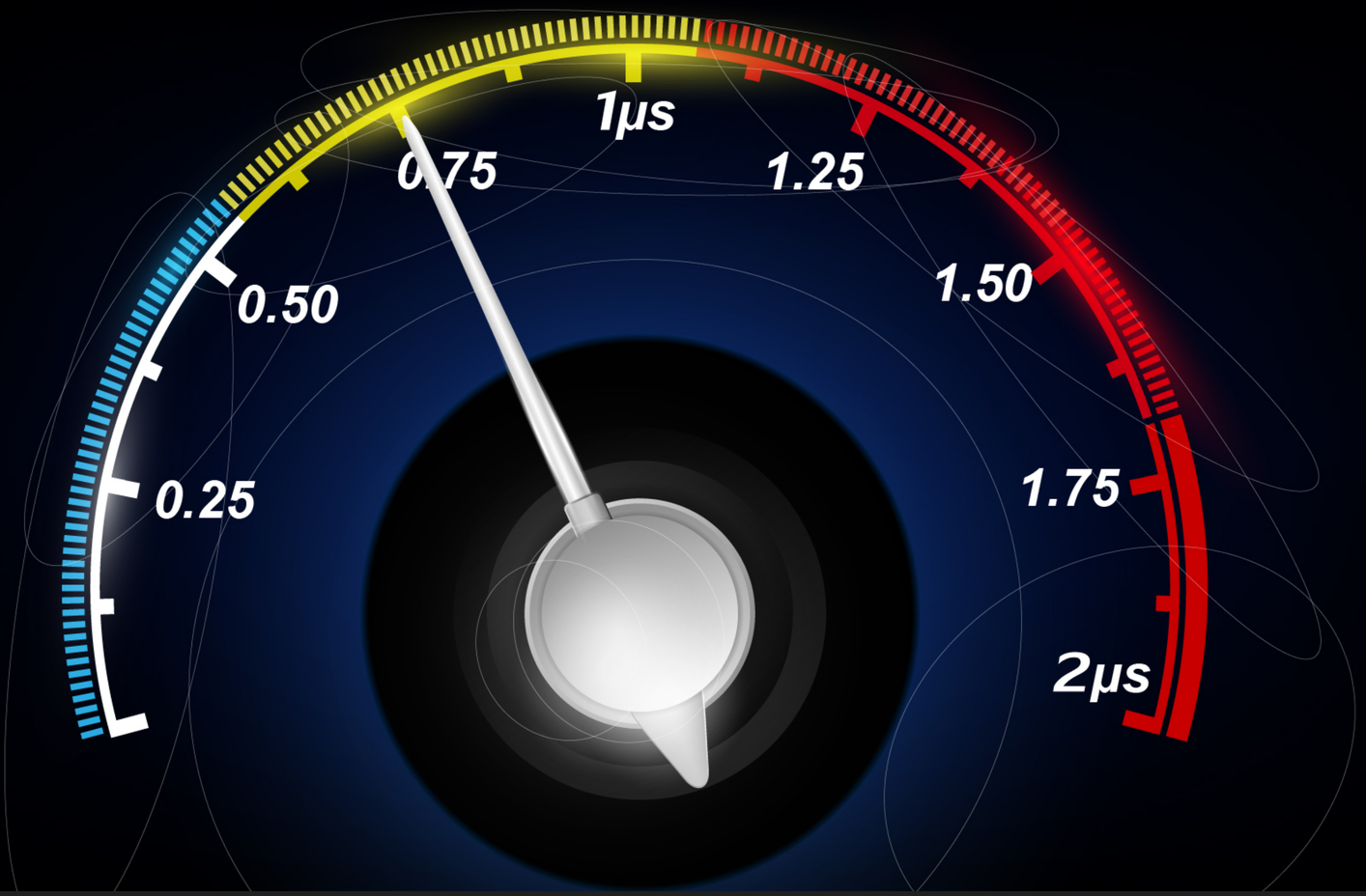


ernie commented on 17 Oct 2016 Member

@bert I guess there is a reason that writing C-extensions is so popular in Ruby.

```
require 'benchmark'
h = {}
Benchmark.realtime { 1000.times { c = h[:foo_total] || 0; c += 1; h[:foo_total] = c }
```

Performance



Separate storage

```
class Metric
  @value = 0

  def increment
    @value += 1
  end
end
```

Separate storage

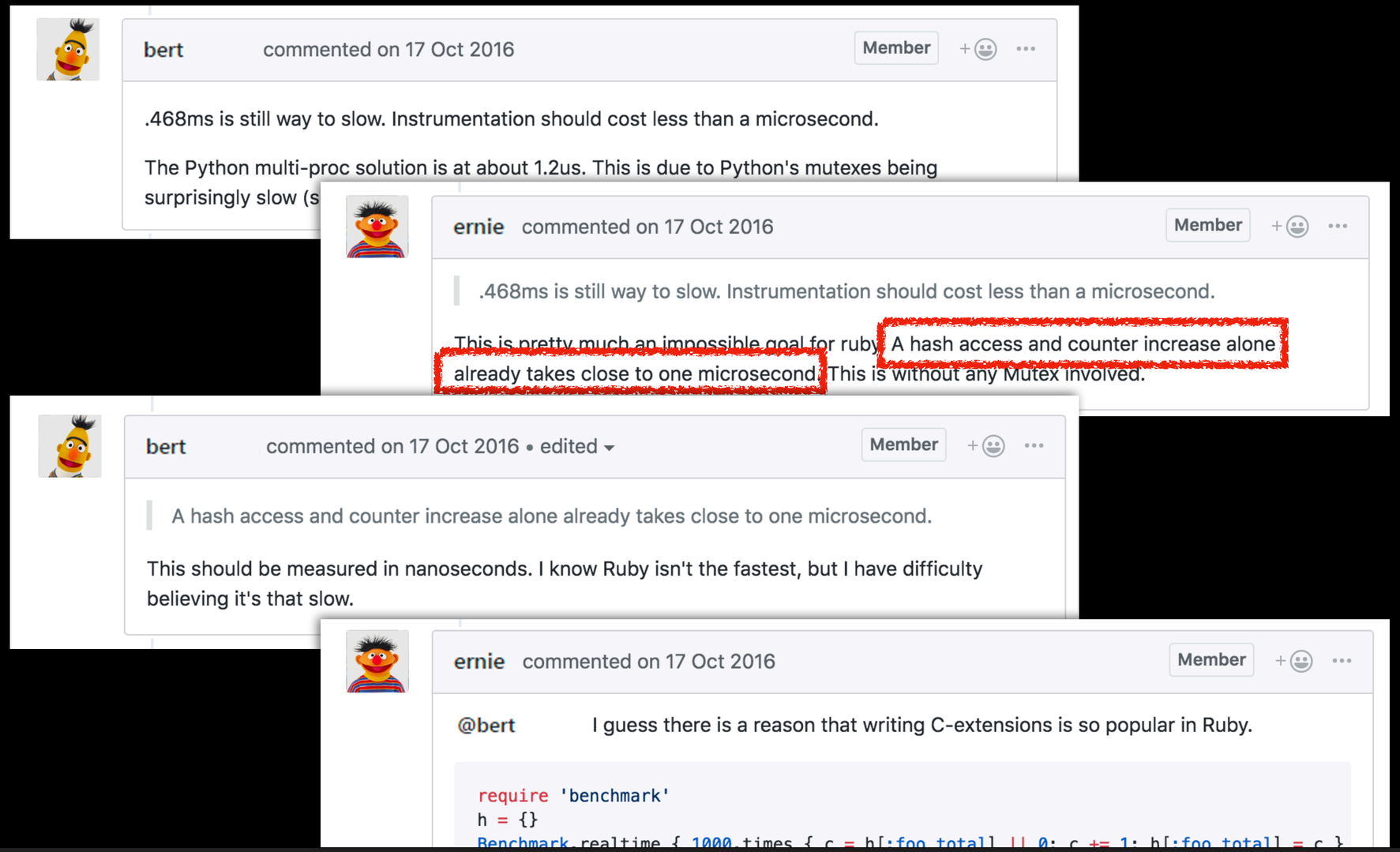
```ruby
class Metric
  @value = 0

  def increment
    @value += 1
  end
end
```

```ruby
class Metric
  @store = SomeStore.new

  def increment
    @store.increment(metric_name, 1)
  end
end
```

The road ahead

- Support pre-fork servers
- Abstract away storage of numbers
- Follow current best practices
- Be fast

The road ahead



Can we even do this?

```
while (i < ITERS)
  HASH[:x] += 1
  i += 1
end
```

≈ 0.1 µs

Can we even do this?

```
while (i < ITERS)
  HASH[:x] += 1
  i += 1
end
```

≈ 0.1 µs

```
while (i < ITERS)
 i += 1
end
```

≈ 0.0175 µs

Can we even do this?

```
HASH[:x] += 1
```

0.0825 µs

Can we even do this?

Can we even do this?

Labels

```
payments_count[{currency: "USD"}] = 1030
payments_count[{currency: "EUR"}] += 143

http_requests[{path: "/payments", status: 200}] = 100
http_requests[{path: "/payments", status: 500}] = 1
http_requests[{path: "/nope", status: 404}] = 1
```

Hash

With a Hash for a key

Can we even do this?

```
HASH[:x] += 1
```

0.0825 µs 

```
HASH[{a: 1, b: 2, c: 3}] += 1
```

Can we even do this?

```
HASH[:x] += 1
```

0.0825 µs 🎉

60x slower

```
HASH[{a: 1, b: 2, c: 3}] += 1
```

5.0000 µs 😭

Can we even do this?

```
HASH[:x] += 1
```

0.0825 µs 🎉

```
HASH[{a: 1, b: 2, c: 3}] += 1
```

5.0000 µs 😭


```
HASH[{}] += 1
```

0.7500 µs 😢

Can we even do this?

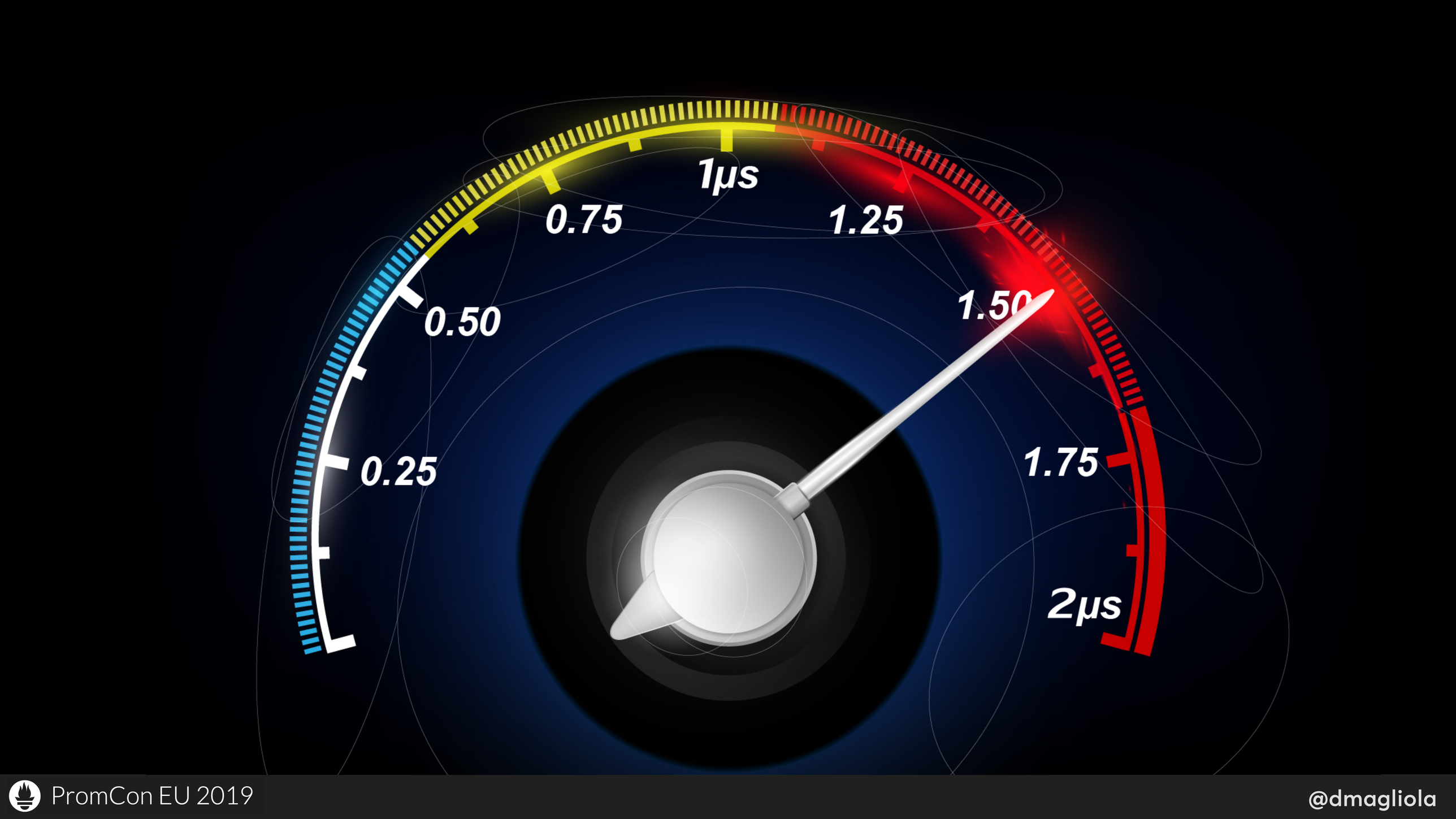
```
def increment(some_labels)
  HASH[labels] += 1
end
```

Can we even do this?

```
def increment(some_labels)
  validate(some_labels)
  labels = process(some_labels)

  MUTEX.synchronize do
    HASH[labels] += 1
  end
end
```

1.5 µs 😢

Can we even do this?


0.25
0.50
0.75
1µs
1.25
1.50
1.75
2µs


Physics

Home Classes Methods

In Files

pstore.rb

Parent

Object

Namespace

CLASS PStore::Error

Methods

::new
#[]
#[]=
#abort
#commit
#delete
#fetch
#path

PStore

PStore implements a file based persistence mechanism based on a Hash. User code can store hierarchies of Ruby objects (values) into the data store file by name (keys). An object hierarchy may be just a single object. User code may later read values back from the data store or even update data, as needed.

The transactional behavior ensures that any changes succeed or fail together. This can be used to ensure that the data store is not left in a transitory state, where some values were updated but others were not.

Behind the scenes, Ruby objects are stored to the data store file with Marshal. That carries the usual limitations. Proc objects cannot be marshalled, for example.

Usage example:¶ ↑

```
require "pstore"

# a mock wiki object...
class WikiPage
  def initialize( page_name, author, contents )
```

Experiment: ■ □ □ □ PStore

```ruby
store = PStore.new("/tmp/mydata.whatever")

store.transaction do
  store[:x] += 1
end
```

Experiment: ■ □ □ □ PStore

Hash without lock:	0.9 µs
Hash with a Monitor lock:	1.5 µs
PStore:	35.0 µs

Tiago 09:55

Why don't we use SQLite?

Experiment: ■ ■ □ □ Redis

Experiment: ■ ■ □ □ Redis

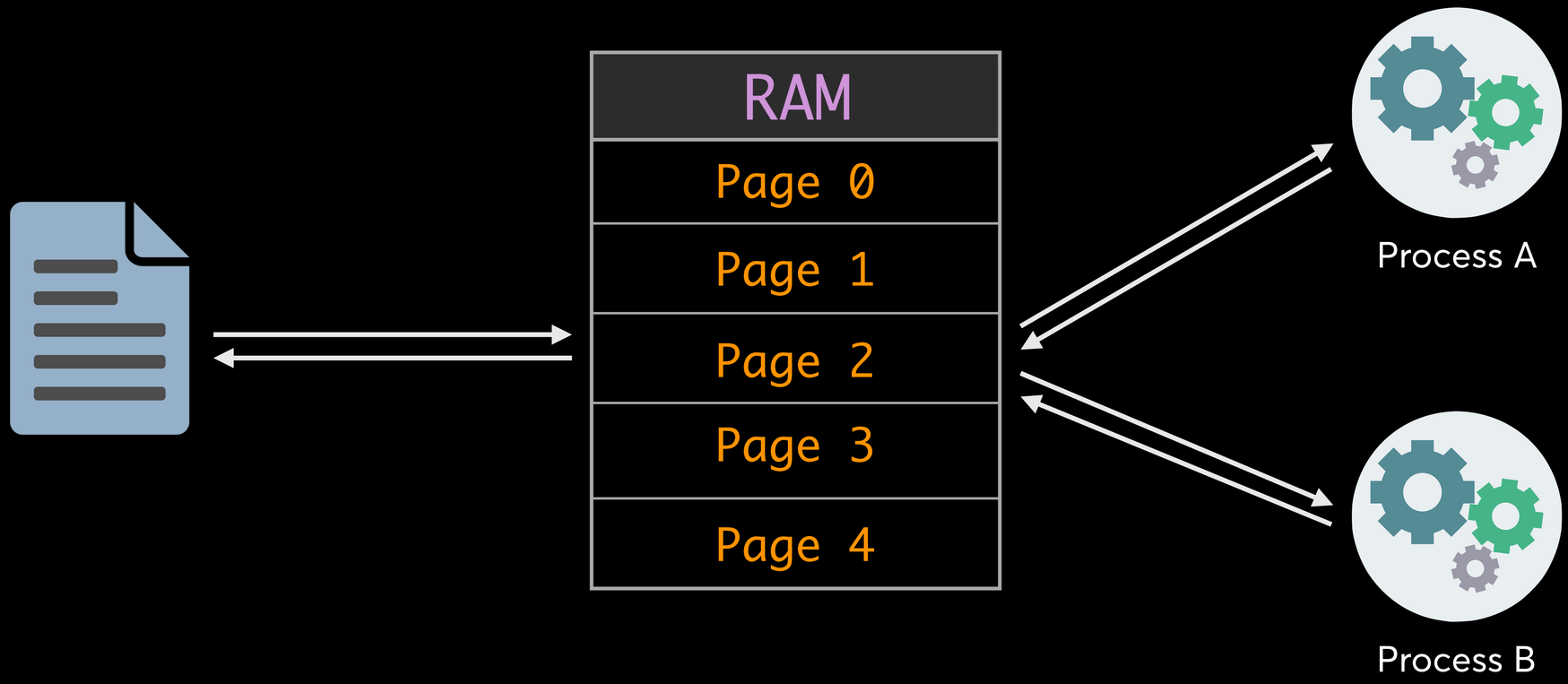
Hash without lock:	0.9 µs
Hash with a Monitor lock:	1.5 µs
PStore:	35.0 µs
Redis:	65.0 µs

Experiment: ■ ■ □ □ Redis

```
Hash without lock:                0.9 µs

Hash with a Monitor lock:         1.5 µs

PStore:                          35.0 µs

Redis:                           65.0 µs
```

Photo by **David Blackwell** on Foter .com / CC BY-ND

Experiment: ■ ■ ■ □ MMaps



Experiment: ■ ■ ■ □ MMaps

less ⌥⌘2

```
MMAP(2)                   BSD System Calls Manual                  MMAP(2)

NAME
     mmap -- allocate memory, or map files or devices into memory

LIBRARY
     Standard C Library (libc, -lc)

SYNOPSIS
     #include <sys/mman.h>

     void *
     mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);

DESCRIPTION
     The mmap() system call causes the pages starting at addr and continuing for at most len bytes to be mapped from the
     object described by fd, starting at byte offset offset.  If offset or len is not a multiple of the pagesize, the mapped
     region may extend past the specified range.  Any extension beyond the end of the mapped object will be zero-filled.

     The addr argument is used by the system to determine the starting address of the mapping, and its interpretation is
     dependent on the setting of the MAP_FIXED flag.  If MAP_FIXED is specified in flags, the system will try to place the
     mapping at the specified address, possibly removing a mapping that already exists at that location.  If MAP_FIXED is
     not specified, then the system will attempt to use the range of addresses starting at addr if they do not overlap any
     existing mappings, including memory allocated by malloc(3) and other such allocators.  Otherwise, the system will
     choose an alternate address for the mapping (using an implementation dependent algorithm) that does not overlap any
     existing mappings.  In other words, without MAP_FIXED the system will attempt to find an empty location in the address
     space if the specified address range has already been mapped by something else.  If addr is zero and MAP_FIXED is not
:
```

Experiment: ■ ■ ■ □ MMaps

method **syscall**

Ruby latest stable (v2_5_5) - 0 notes - Class: Kernel

What's this?

1_8_6_287 1_8_7_72 1_8_7_330 1_9_1_378 1_9_2_180 1_9_3_125 1_9_3_392 2_1_10 2_2_9 2_4_6 2_5_5 2_6_3

Flowdock

`syscall(*args)` *public*

Calls the operating system function identified by *num* and returns the result of the function or raises SystemCallError if it failed.

Arguments for the function can follow *num*. They must be either `String` objects or `Integer` objects. A `String` object is passed as a pointer to the byte sequence. An `Integer` object is passed as an integer whose bit size is same as a pointer. Up to nine parameters may be passed.

The function identified by *num* is system dependent. On some Unix systems, the numbers may be obtained from a header file called syscall.h.

```
syscall 4, 1, "hello\n", 6   # '4' is write(2) on our box
```

produces:

```
hello
```

Feel free to shoot your foot.

Calling `syscall` on a platform which does not have any way to an arbitrary system function just fails with NotImplementedError.

Note: `syscall` is essentially unsafe and unportable. Feel free to shoot your foot. The DL (Fiddle) library is preferred for safer and a bit more portable programming.

Show source

Register or log in to add new notes.

Experiment: MMaps

tenderlove / mmap

Watch 2 | Star 34 | Fork 12

Code | Issues 3 | Pull requests 4 | Actions | Projects 0 | Wiki | Security | Insights

A wrapper around mmap

19 commits | 1 branch | 0 packages | 0 releases | 2 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

tenderlove Merge pull request #2 from wjessop/add_modern_boilerplate ... Latest commit 9632c6e on 28 Sep 2016

ext/mmap	cleaning up compiler warnings	4 years ago
lib	fixing up docs	10 years ago
test	skip the modify tests for now	4 years ago
.gitignore	Add modern gem boilerplate	3 years ago
Changes	initial checkin	10 years ago
Gemfile	Add modern gem boilerplate	3 years ago
Manifest.txt	removing runit	10 years ago
README.rdoc	fixing up docs	10 years ago
Rakefile	fixing gem install and readme url	10 years ago

Experiment: ■ ■ ■ □ MMaps

Commits on Mar 2, 2016

cleaning up compiler warnings  **tenderlove** committed on 2 Mar 2016	ce999c9
skip the modify tests for now  **tenderlove** committed on 2 Mar 2016	f4332d6
tests mostly fixed  **tenderlove** committed on 2 Mar 2016	c7b4f3a
fix string free error  **tenderlove** committed on 2 Mar 2016	2a2f11c
fixing some segvs  **tenderlove** committed on 2 Mar 2016	0a748e7
compiling seems to work  **tenderlove** committed on 2 Mar 2016	6ab5433

Commits on Oct 22, 2009

fixing up docs  **tenderlove** committed on 22 Oct 2009	7288913
finish C documentation  **tenderlove** committed on 22 Oct 2009	93a992b

Experiment: ■ ■ ■ □ MMaps

```
@f = File.open('/tmp/whatevs', 'a+b')

@f.truncate(INITIAL_MMAP_SIZE) # Init to 1Mb

m = Mmap.new('/tmp/whatevs', 'rw', Mmap::MAP_SHARED)

raw_bytes = m[pos..pos + 3] # Read 4 bytes

m[pos..pos + 3] = raw_bytes # Write 4 bytes
```

Experiment: ■ ■ ■ □ MMaps

```
@f = File.open('/tmp/whatevs', 'a+b')

@f.truncate(INITIAL_MMAP_SIZE) # Init to 1Mb

m = Mmap.new('/tmp/whatevs', 'rw', Mmap::MAP_SHARED)

raw_bytes = m[pos..pos + 3] # Read 4 bytes

m[pos..pos + 3] = raw_bytes # Write 4 bytes
```

Experiment: ■ ■ ■ □ MMaps

```
key = { currency: "USD", country: "US" }
my_float = 1234.5
last_pos = 128 # How much of the file we've used so far

key = "currency=USD&country=US"
```

Experiment: ■ ■ ■ □ MMaps

```
key = { currency: "USD", country: "US" }
my_float = 1234.5
last_pos = 128 # How much of the file we've used so far

key = "currency=USD&country=US"

# Pad the key to an 8-byte boundary
padded = key + (' ' * (8 - (key.length + 4) % 8))

# Pack the length, the key and the float
packed = [key.length, padded, my_float].pack("lA#{padded.length}d")
```

Experiment: ■ ■ ■ □ MMaps

```
key = { currency: "USD", country: "US" }
my_float = 1234.5
last_pos = 128 # How much of the file we've used so far

key = "currency=USD&country=US"

# Pad the key to an 8-byte boundary
padded = key + (' ' * (8 - (key.length + 4) % 8))

# Pack the length, the key and the float
packed = [key.length, padded, my_float].pack("lA#{padded.length}d")

# Store in the  MMap
m[last_pos..last_pos + packed.length] = value
```

Experiment: ■ ■ ■ □ MMaps

```
key = { currency: "USD", country: "US" }
my_float = 1234.5
last_pos = 128 # How much of the file we've used so far

key = "currency=USD&country=US"

# Pad the key to an 8-byte boundary
padded = key + (' ' * (8 - (key.length + 4) % 8))

# Pack the length, the key and the float
packed = [key.length, padded, my_float].pack("lA#{padded.length}d")

# Store in the  MMap
m[last_pos..last_pos + packed.length] = value

last_pos += value.length # Update how much space we've used.
@positions[key] = last_pos - 8 # Keep track of where our float is
```

Experiment: ■ ■ ■ □ MMaps

```
key = { currency: "USD", country: "US" }
key = "currency=USD&country=US"

pos = @positions[key]

m[pos..pos + 7] = [my_float].pack('d')
```

Experiment: ■ ■ ■ □ MMaps

Hash without lock:	0.9 µs
Hash with a Monitor lock:	1.5 µs
PStore:	35.0 µs
MMaps:	6.1 µs

Experiment: ■ ■ ■ □ MMaps

Experiment: ■ ■ ■ □ MMaps

```
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:221: [BUG] Bus Error at 0x000000010c4853c0
ruby 2.4.2p198 (2017-09-14 revision 59899) [x86_64-darwin16]

-- Crash Report log information --------------------------------------------
   See Crash Report log file under the one of following:
     * ~/Library/Logs/DiagnosticReports
     * /Library/Logs/DiagnosticReports
   for more details.
Don't forget to include the above Crash Report log file in bug reports.

-- Control frame information -----------------------------------------------
c:0011 p:---- s:0053 e:000052 CFUNC  :[]
c:0010 p:---- s:0050 e:000049 CFUNC  :[]
c:0009 p:0057 s:0045 e:000044 METHOD /Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:221
c:0008 p:0012 s:0039 e:000038 BLOCK  /Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:96
c:0007 p:0006 s:0035 e:000034 BLOCK  /Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:137
c:0006 p:0040 s:0032 e:000031 METHOD /Users/danielmagliola/.rbenv/versions/2.4.2/lib/ruby/gems/2.4.0/gems/concurrent-ruby-1.1.2/lib/concurrent/atomic/reentrant_read
c:0005 p:0011 s:0028 e:000027 METHOD /Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:137
c:0004 p:0019 s:0024 e:000023 METHOD /Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:95
c:0003 p:0057 s:0016 e:000015 METHOD /Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/counter.rb:17
c:0002 p:0050 s:0008 e:000007 BLOCK  high_cardinality_test.rb:23 [FINISH]
c:0001 p:---- s:0003 e:000002 (none) [FINISH]

-- Ruby level backtrace information ----------------------------------------
high_cardinality_test.rb:23:in `block in <main>'
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/counter.rb:17:in `increment'
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:95:in `increment'
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:137:in `in_process_sync'
/Users/danielmagliola/.rbenv/versions/2.4.2/lib/ruby/gems/2.4.0/gems/concurrent-ruby-1.1.2/lib/concurrent/atomic/reentrant_read_write_lock.rb:147:in `with_write_lock'
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:137:in `block in in_process_sync'
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:96:in `block in increment'
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:221:in `read_value'
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:221:in `[]'
/Users/danielmagliola/Documents/prometheus_client_ruby/lib/prometheus/client/data_stores/mmap_store.rb:221:in `[]'

-- Machine register context ------------------------------------------------
 rax: 0x00007fed01817500 rbx: 0x00007fed018174f0 rcx: 0x00007fed008cfe20
 rdx: 0x0000000000000000 rdi: 0x00007fed01817500 rsi: 0x000000010c4853c0
 rbp: 0x000070000e9722e0 rsp: 0x000070000e9722e0  r8: 0x0000000000000000
  r9: 0x00007fecff6003a0 r10: 0x000070000e9724c8 r11: 0x00007febf5392140
```

Experiment: ■ ■ ■ □ MMaps

Experiment: ■ ■ ■ ■ Files

```
@f = File.open('/tmp/whatevs', 'a+b')

@f.truncate(INITIAL_MMAP_SIZE) # Init to 1Mb

m = Mmap.new('/tmp/whatevs', 'rw', Mmap::MAP_SHARED)

m[pos..pos + 7] = [my_float].pack('d') # Write our float
```

Experiment: ■ ■ ■ ■ Files

```
@f = File.open('/tmp/whatevs', 'a+b')

@f.truncate(INITIAL_MMAP_SIZE) # Init to 1Mb

m = Mmap.new('/tmp/whatevs', 'rw', Mmap::MAP_SHARED)

m[pos..pos + 7] = [my_float].pack('d') # Write our float
@f.seek(pos)
@f.write([my_float].pack('d'))
```

Experiment:  Files

```
pos = @positions[key]

@f.seek(pos)
@f.write([my_float].pack('d'))
```

Experiment:  Files

Hash without lock:	0.9 µs
Hash with a Monitor lock:	1.5 µs
PStore:	35.0 µs
MMaps:	6.1 µs
Boring, SLOW, old files:	8.8 µs

Files vs MMaps

- No external dependencies

Files vs MMaps

- No external dependencies
- No risky C code

Files vs MMaps

- No external dependencies
- No risky C code
- No segfaults

Files vs MMaps

- No external dependencies
- No risky C code
- No segfaults
- Compatible with all versions of Ruby

Files vs MMaps

- No external dependencies
- No risky C code
- No segfaults
- Compatible with all versions of Ruby
- 100% understandable

Files vs MMaps

- No external dependencies
- No risky C code
- No segfaults
- Compatible with all versions of Ruby
- 100% understandable
- No things that keep me up at night

The end result

- 3 built-in stores
  - Unsynchronized Hash
  - Synchronized Hash
  - FileStore

The end result

- 3 built-in stores
- Aggregation options for metrics in multi-process scenarios

The end result

- 3 built-in stores
- Aggregation options for metrics in multi-process scenarios
- Label declaration and validation following best practices

The end result

- 3 built-in stores
- Aggregation options for metrics in multi-process scenarios
- Label declaration and validation following best practices
- Better aligned with Ruby conventions (e.g. kwargs)

The end result

- 3 built-in stores
- Aggregation options for metrics in multi-process scenarios
- Label declaration and validation following best practices
- Better aligned with Ruby conventions (e.g. kwargs)
- Tons of performance optimizations all over the place

The end result

- 3 built-in stores
- Aggregation options for metrics in multi-process scenarios
- Label declaration and validation following best practices
- Better aligned with Ruby conventions (e.g. kwargs)
- Tons of performance optimizations all over the place
- Benchmark script for testing performance

prometheus / client_ruby

Watch 22 | Star 284 | Fork 99

Code | Issues 15 | Pull requests 4 | Projects 0 | Security | Insights

Abstract Data Storage away from Metric objects, introduce Swappable Data Stores, and support Multiprocess Pre-fork Servers #95

Merged **Sinjo** merged 18 commits into `prometheus:master` from `gocardless:pluggable_data_stores` on 24 Apr

Conversation 71 | Commits 18 | Checks 1 | Files changed 36 | +2,359 −346

dmagliola commented on 22 Oct 2018 • edited — Collaborator

This PR attempts to address the first set of changes required for the objectives outlined in Issue 94

This is an excerpt from that issue, please check the full text for more context:

As it currently stands, the Prometheus Ruby Client has a few issues that make it hard to adopt in mainstream Ruby projects, particularly in Web applications:

1. Pre-fork servers can't report metrics, because each process has their own set of data, and what gets reported to Prometheus depends on which process responds to the scrape request.
2. The current Client, being one of the first clients created, doesn't follow several of the Best Practices and Guidelines.

Objectives

- Follow client conventions and best practices

Reviewers

lzap ✓
errm
Sinjo
brian-brazil

Assignees

No one assigned

Labels

None yet

Projects

VERSIONS:

1.0 - November 4, 2019 (23 KB)
0.11.0.pre.alpha.1 - October 28, 2019 (23 KB)
0.10.0 - October 07, 2019 (23 KB)
0.10.0.pre.alpha.2 - June 25, 2019 (23 KB)
0.10.0.pre.alpha.1 - May 23, 2019 (23 KB)
0.9.0 - January 25, 2019 (12 KB)
0.4.2 - May 18, 2015 (10.5 KB)

Future work

- Init timeseries / labelsets to 0

Future work

- Init timeseries / labelsets to 0
- "Starting with an empty dir" situation

Future work

- Init timeseries / labelsets to 0
- "Starting with an empty dir" situation
- The "too many files" problem

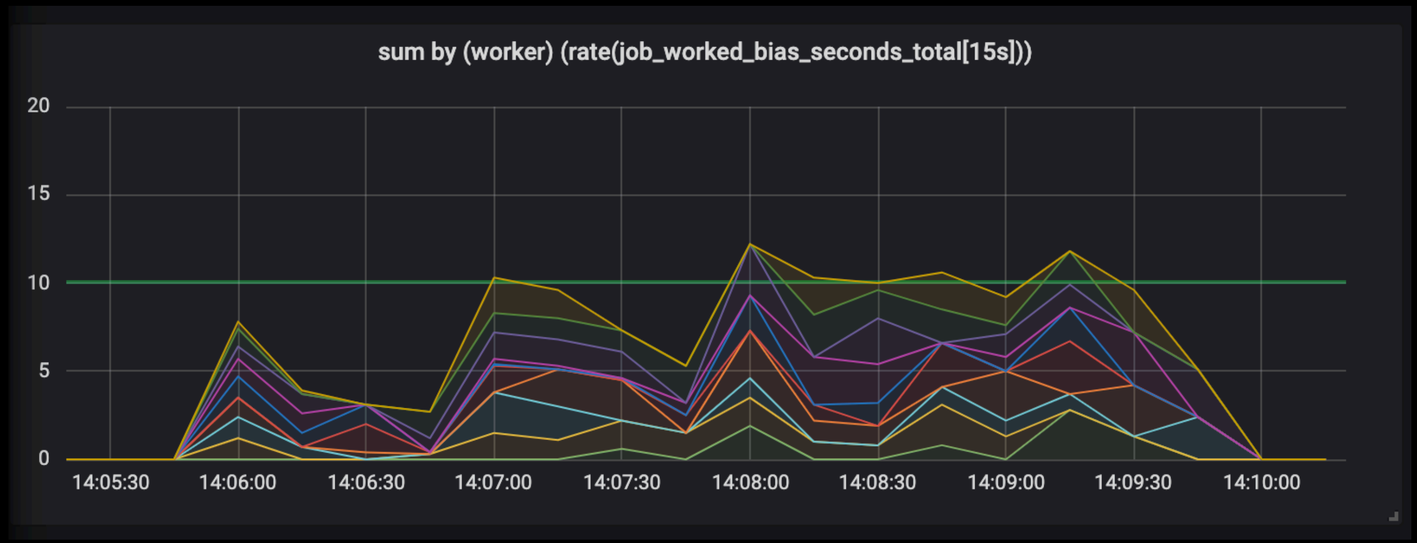
Future work

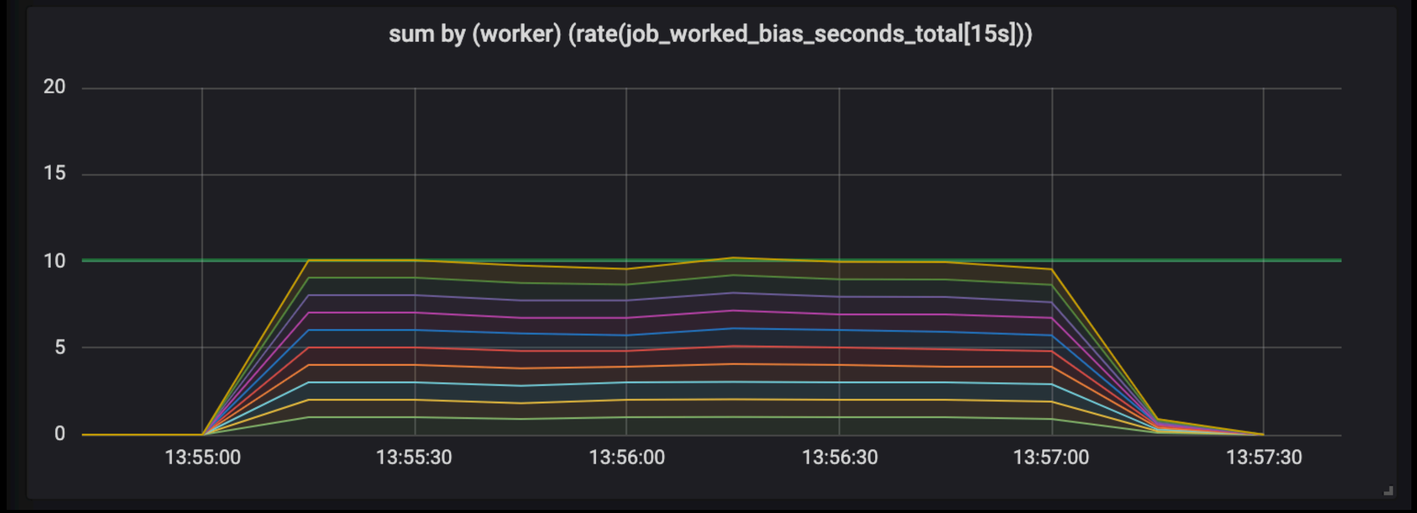
- Init timeseries / labelsets to 0
- "Starting with an empty dir" situation
- The "too many files" problem
- Slow exports

New "Tracer" pattern

```
start = Time.now
really_long_job.run
duration = Time.now - start

JobWorkedSecondsTotal.increment(
  by: duration,
  labels: { job: job.class }
)
```

New "Tracer" pattern





New "Tracer" pattern

Lawrence Jones

https://blog.lawrencejones.dev/incremental-measurement/

https://github.com/lawrencejones/prometheus-client-tracer-ruby

Shout-outs

Shout-outs

Jeffery Utter (@jeffutter)

Kevin Lyda (@lyda)

Sam Saffron (@samsaffron)

Julius Volz (@juliusvolz)

Tobias Schmidt (@dagrobie)

Thank you

What the fork?

Reporting Metrics in a Multi-process Environment

PromCon EU 2019

https://github.com/prometheus/client_ruby

https://blog.lawrencejones.dev/incremental-measurement/

Daniel Magliola

@dmagliola

@GoCardlessEng