# Profiling Ruby

with Flamegraphs

Janusz Mordarski

# $ whoami

---

Janusz Mordarski

#ruby #rails #python #javascript #go #web #api #devops #rest #amqp #unix #agile #kanban #scrum

janusz@mordarski.eu

https://github.com/januszm

https://www.linkedin.com/in/janusz-mordarski/

http://mordarski.eu

# Code profiling

# Why?

---

“if you can’t **measure** it, you can’t **improve** it”

“**know** your system”

Why is my code slow?

(when should I ask this question?)

# What?

---

`What is profile ?`

“To **profile** someone means to give an account of that person's life and character.”

“Your **profile** is the outline of your face as it is seen when someone is looking at you from the side.”

“A record of a person's psychological or **behavioural characteristics**, preferences, etc.”

Source: https://www.collinsdictionary.com/dictionary/english/profile

Source: https://sofifa.com/player/188545

# R. Lewandowski (ID: 188545)

Robert Lewandowski ST Age 28 (Aug 21, 1988) 6'1" 174lbs

Overall Rating 91 | Potential 91 | Value €92M | Wage €355K

Preferred Foot Right
International Reputation 4
Weak Foot 4
Skill Moves 3
Work Rate High / Medium
Body Type Normal
Real Face Yes
Release clause €151.8M


FC Bayern Munich
86
Position ST
Jersey Number 9
Joined Jul 1, 2014
Contract Valid Until 2021

Poland
78
Position ST
Jersey Number 9

#Clinical Finisher

Follow (560) | Like (0) | Dislike (16)

Jump to

## Attacking

62 Crossing
91 Finishing
85 Heading Accuracy
85 Short Passing
89 Volleys

## Skill

85 Dribbling
77 Curve
84 FK Accuracy
65 Long Passing
89 Ball Control

## Movement

79 Acceleration
79 Sprint Speed
78 Agility
91 Reactions
80 Balance

## Power

88 Shot Power
84 Jumping
79 Stamina
84 Strength
84 Long Shots

# What?

---

## What can we profile?

https://en.wikipedia.org/wiki/Profiling_(computer_programming)

## What matters for “internet” applications?

- **CPU** usage
- memory consumption
- what else?

# Call stack

---

Who has the control over the CPU?

net => OS => http server =>

```
def call
 @record = Model.find_by(uuid: ...)
 begin
   crm.set_paid(@record.op_id, @transaction[:id])
 rescue StandardError => e
   Raven.capture_exception(e, extra: { id: ..., ...
 end
 AgentMailer.new_application(@household).deliver_later
end
```

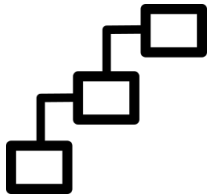
http server => OS => net

# Two approaches (that matter) [for us]

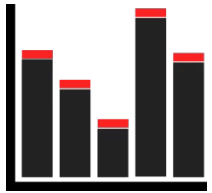
---

Profilers:

- **instrumenting** / tracing / deterministic
  - from function to function, track the execution time



- **sampling** / statistical
  - from time slice to time slice, call stack snapshots

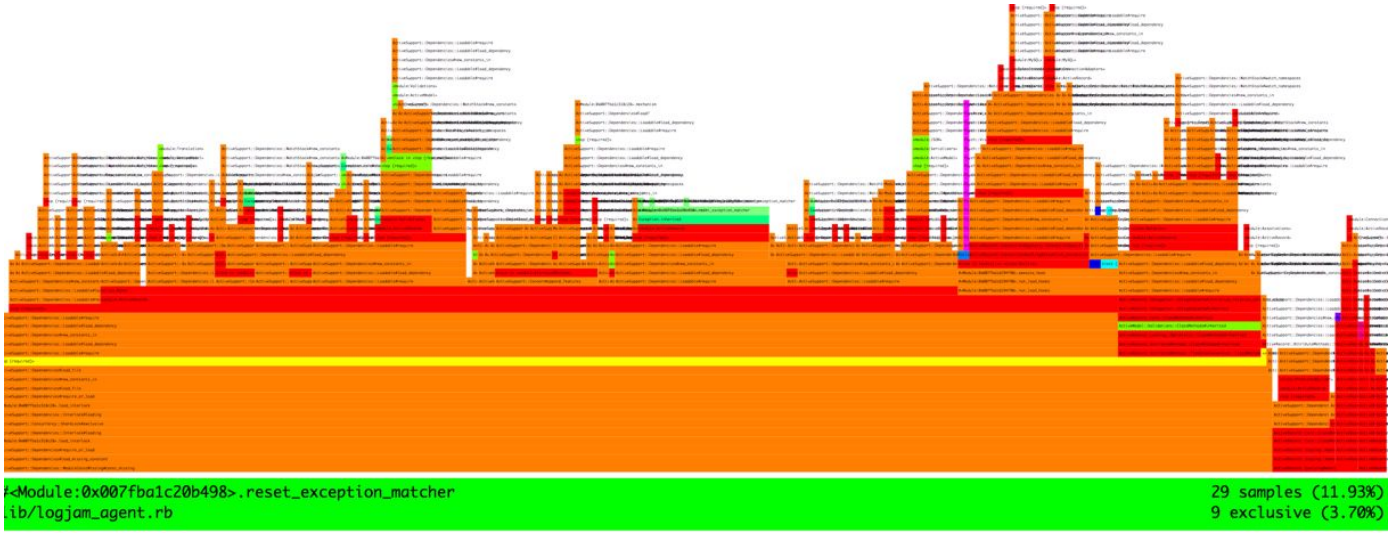


# Seeing is believing

---

http://www.brendangregg.com/flamegraphs.html

“Flame graphs are a visualization of profiled software, allowing the most frequent code-paths to be identified quickly and accurately.”



# is_a? Ruby

# Stackprof

---

**Sampling** profiler. Very **low footprint,** low accuracy.

Great for long running processes or methods (eg. processing data in a loop).

https://github.com/tmm1/stackprof

# ruby-prof

---

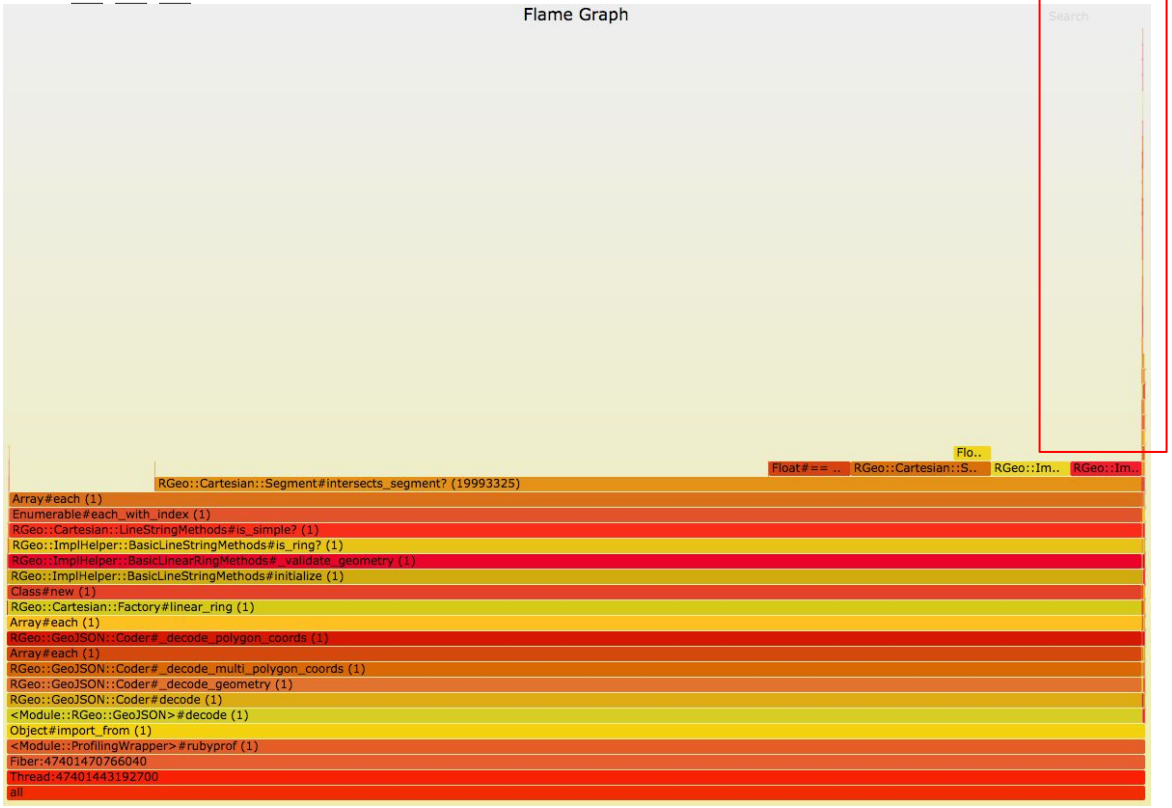
Instrumenting profiler, **accurate**, but may affect the processing time (observer effect, “heisenbugs”).

Great for **pinpointing** issues and profiling short processes or methods.

https://github.com/ruby-prof/ruby-prof

# sampling vs instrumenting (very short process)

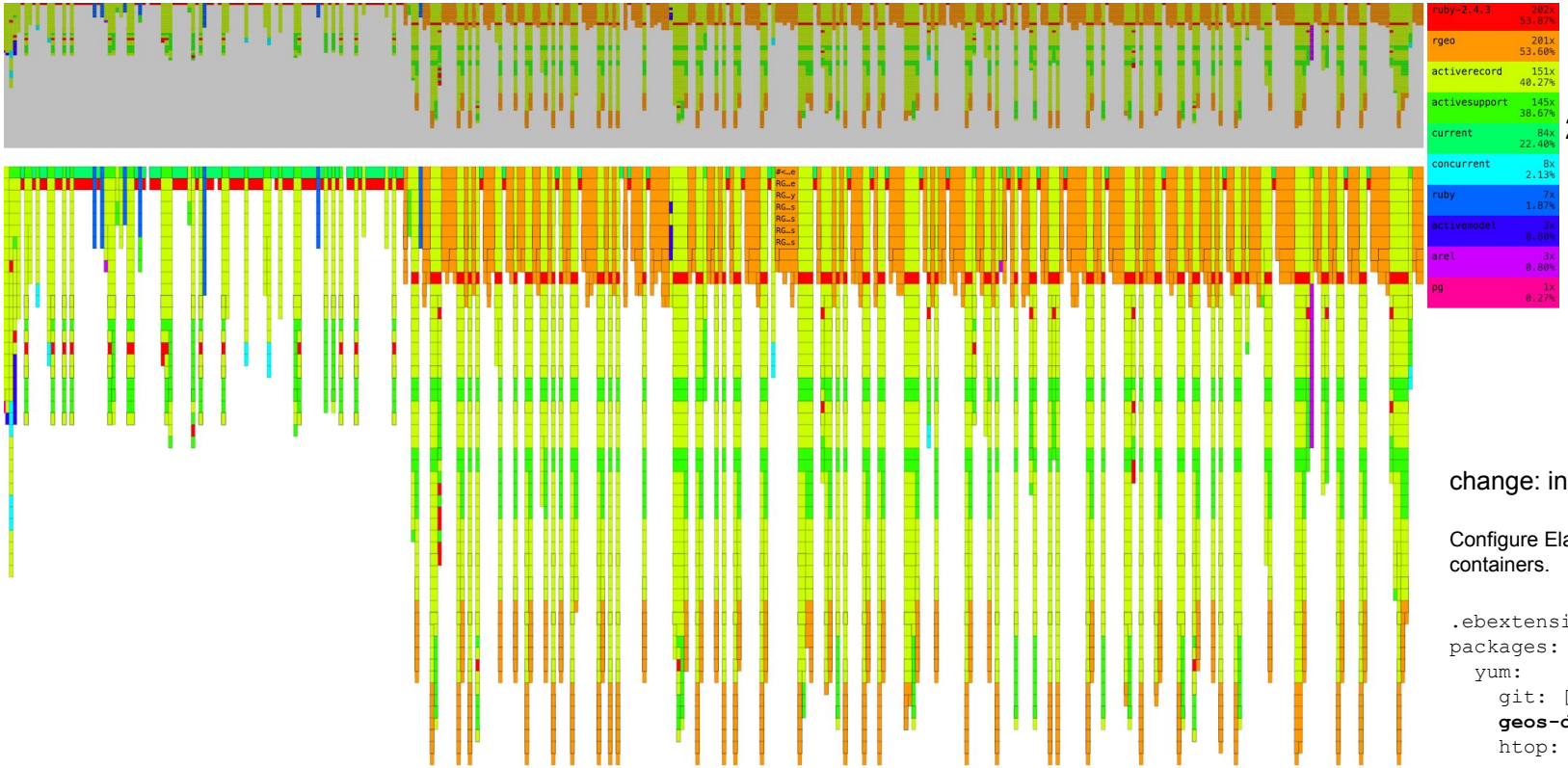
# sampling vs instrumenting (very short process)




# sampling vs instrumenting (short process looped)


something is wrong here, 99% of the time is spent in RGeo (https://github.com/rgeo/rgeo)

# sampling vs instrumenting (short process looped)




**30k** samples
vs
**200** samples!

150x
improvement

change: install geos-devel

Configure Elastic Beanstalk containers.

```
.ebextensions/...
packages:
  yum:
    git: []
    geos-devel: []
    htop: []
```

# Wrapping it up

```
require "ruby-prof"
require "stackprof"

module ProfilerWrapper
 def self.rubyprof(name)
   GC.disable
   result = ::RubyProf.profile do
     yield
   end
   printer = ::RubyProf::FlameGraphPrinter.new(result)
   printer.setup_options(print_file: true)
   File.open("tmp/#{Time.now.to_i}_#{name}-rubyprof.dump", "w") do |file|
     printer.print(file, {})
   end
 ensure
   GC.enable
 end
```

```
# Gemfile
gem "ruby-prof"
gem "ruby-prof-flamegraph"
gem "stackprof"
```

# Wrapping it up

```
module ProfilerWrapper
 …
 def self.stackprof(name)
   GC.disable
   ::StackProf.run(
     mode: :cpu,
     raw: true, # for the flamegraph
     out: "tmp/#{Time.now.to_i}_#{name}-stackprof.dump", # Thread.current.object_id
     interval: 1000 # microseconds
   ) do
     yield
   end
 ensure
   GC.enable
 end
```

# Wrapping it up

```
ProfilerWrapper.rubyprof("some_module") do
 ...
end

messages.each do |amqp_message|
 if amqp_message.payload["foo"] == "bar"
   ProfilerWrapper.rubyprof("bar") { process(amqp_message) }
 else
   process(amqp_message)
 end
end
```

```
ProfilerWrapper.stackprof("some_other_module") do
 ...
end

ProfilerWrapper.stackprof("messages") do
 messages.each do |amqp_message|
   process_message(amqp_message)
 end
end
```

# Flamegraphs

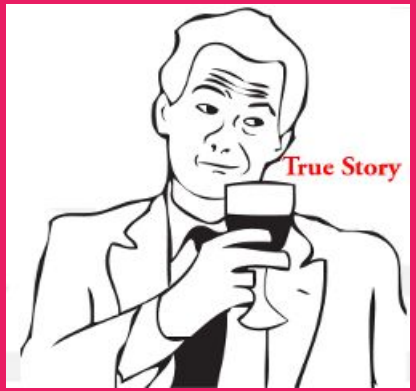
```
# RubyProf
$ bin/flamegraph.pl --countname=ms tmp/12193520_1490104692_myname-rubyprof.dump >
tmp/12193520_1490104692_planned-rubyprof.svg

# Stackprof
$ stackprof --flamegraph tmp/11464440_1490025005-stackprof-cpu-myname.dump >
tmp/11464440_1490025005-stackprof-cpu-process_planned.flame
$ stackprof --flamegraph-viewer=tmp/11464440_1490025005-stackprof-cpu-myname.flame
> open
file://path_to/ruby-2.4.3@mygemset/gems/stackprof-0.2.11/lib/stackprof/flamegraph/viewer.html?data=path_to_project/tmp/11464440_1490
025005-stackprof-cpu-myname.flame

(stackprof --flamegraph internally uses the flamegraph.pl script anyway)
```

# GC - side effects

short-lived process
with garbage collector enabled:


True Story

# Use case (amqp processing, background jobs)

---

“Planned”, daily push notifications

processing large numbers of records in an event driven environment.

Stack: Rails + AMQP + MySQL

Cause: major increase in the volume of the content pushed to users (push notifications).

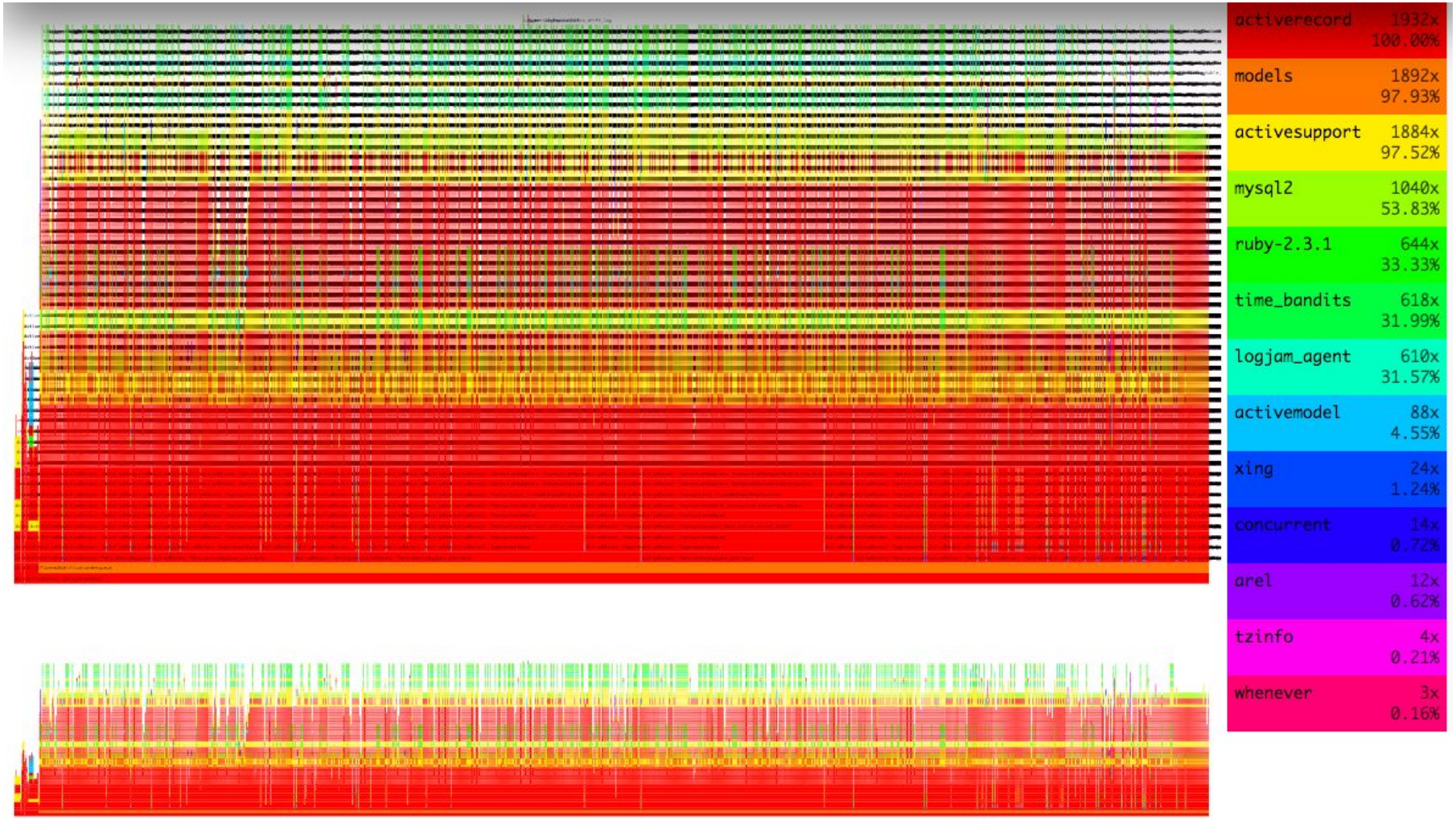
Goal: to be 10x faster.

# Use case (amqp processing, background jobs)

---

Steps:

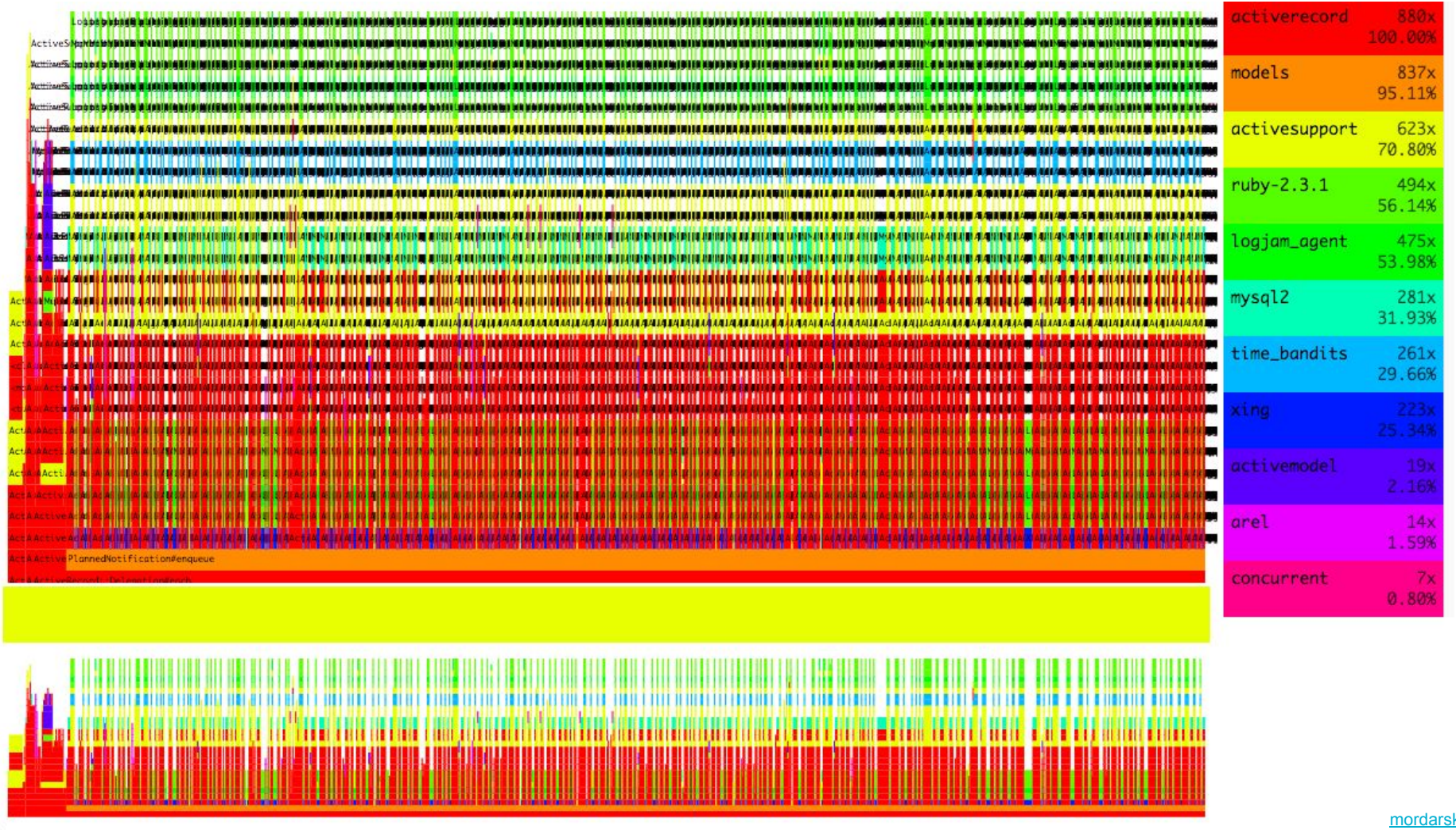
- wrap the important part of the code in the profilers
- collect results (production env, 1 of XX worker servers)
- generate flamegraphs
- refactor the code


activerecord 1932x 100.00%
models 1892x 97.93%
activesupport 1884x 97.52%
mysql2 1040x 53.83%
ruby-2.3.1 644x 33.33%
time_bandits 618x 31.99%
logjam_agent 610x 31.57%
activemodel 88x 4.55%
xing 24x 1.24%
concurrent 14x 0.72%
arel 12x 0.62%
tzinfo 4x 0.21%
whenever 3x 0.16%

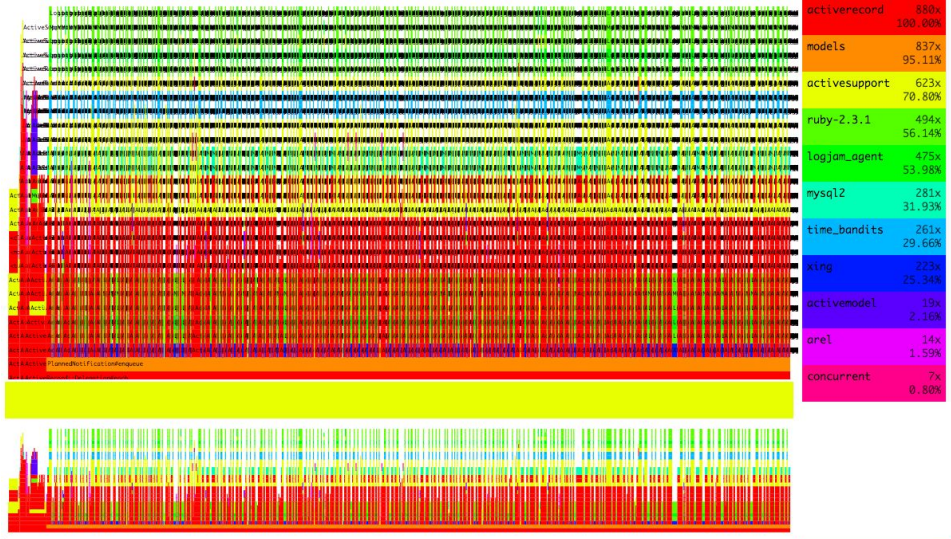
# ActiveRecord => SQL

```
- update_attribute(:enqueued_at, Time.now)
  AMQP::Messenger.publish(:xxx, { foo: :bar }.to_json)


+ update_column(:enqueued_at, Time.now)
  AMQP::Messenger.publish(:xxx, { foo: :bar }.to_json)
```

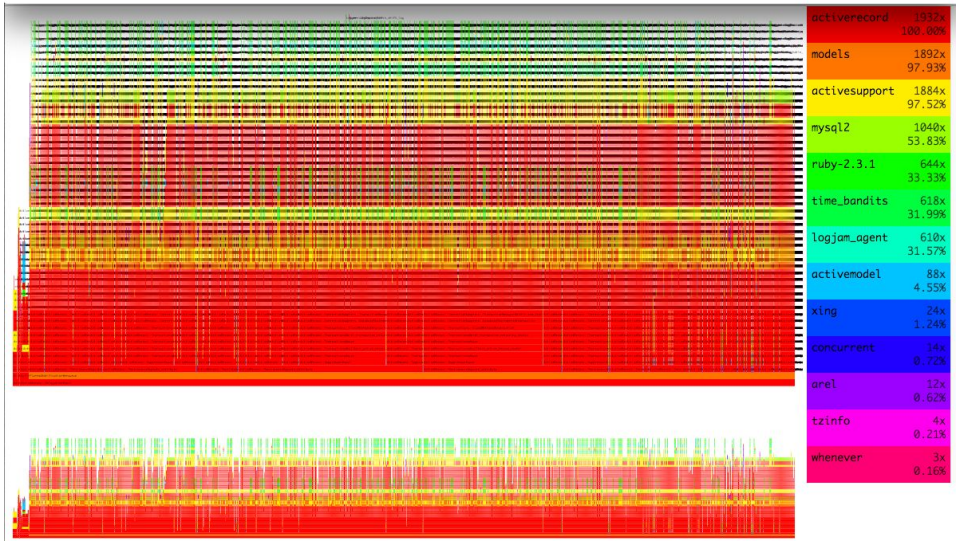

activerecord 880x 100.00%
models 837x 95.11%
activesupport 623x 70.80%
ruby-2.3.1 494x 56.14%
logjam_agent 475x 53.98%
mysql2 281x 31.93%
time_bandits 261x 29.66%
xing 223x 25.34%
activemodel 19x 2.16%
arel 14x 1.59%
concurrent 7x 0.80%
PlannedNotification#enqueue

after:




ActiveRecord in 880 samples

before:




ActiveRecord in 1932 samples

# ActiveRecord => SQL

---

```
sql = %{ UPDATE … #{ActiveRecord::Base.sanitize(Time.now) …
+ Model.connection.execute(sql)
  AMQP::Messenger.publish(...
```

**multiple** AR=>SQL == low memory footprint => batch processing (**~50x speed improvement**)

# Use case (request-response cycle)

```
# Gemfile
gem "rack-mini-profiler"
gem "flamegraph"
gem "stackprof"
```

st?pp=flar

```
before_action do
  Rack::MiniProfiler.authorize_request
end
# OR
around_action :wrap_in_profiler

private

def wrap_in_profiler
  ProfilingWrapper.stackprof("request") do
    yield
  end
end
```

# unless Ruby

# Use case (Python)

running ML models with Flask

Inspired by: http://www.alexandrejoseph.com/blog/2015-12-17-profiling-werkzeug-flask-app.html

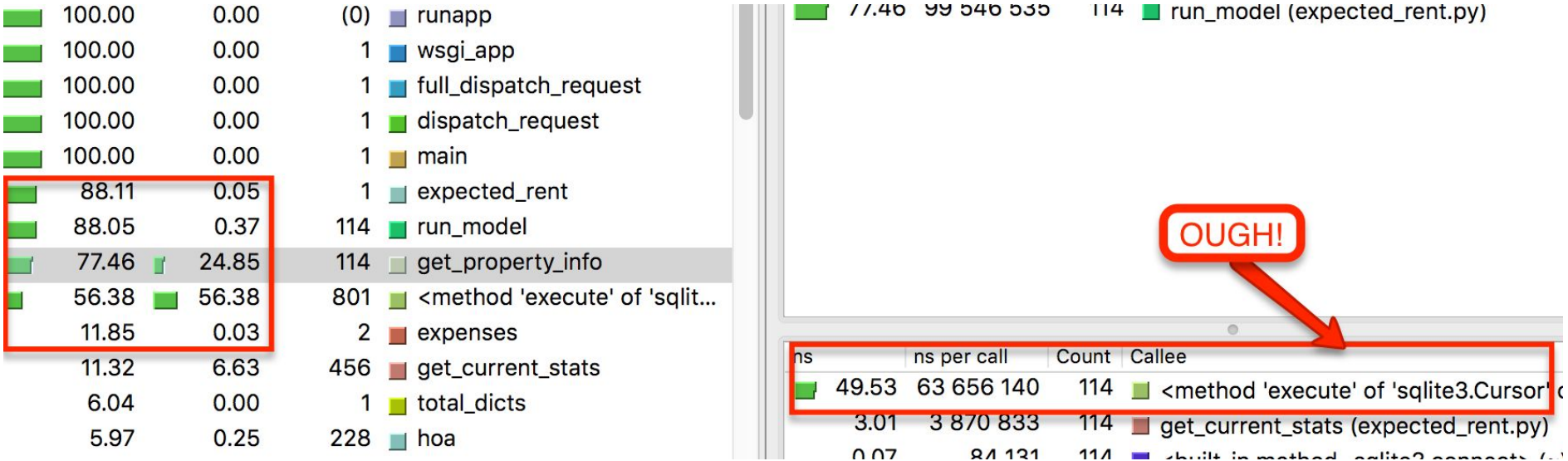
Tools: werkzeug (WSGI) + cProfile / callgrind / qcachegrind

```
+++ b/app/app.py
 import pandas as pd
 from io import BytesIO
+from werkzeug.contrib.profiler import
ProfilerMiddleware

 app = Flask(__name__)
+app.wsgi_app = ProfilerMiddleware(app.wsgi_app,
profile_dir='profile')

 @app.route('/', methods=['GET', 'POST'])
 def main():
```

# Use case (Python)



# Use case (Javascript)

---

Live example:

Chrome Developer Tools >
    Performance recording
    More Tools > Javascript Profiler

Node.js: https://nodejs.org/en/blog/uncategorized/profiling-node-js/

# Bonus: APM and Benchmarking

# Benchmarking

---

```
time = Benchmark.realtime { expensive_method }
```

```
> foo: 0.00286

> bar: 0.05678

> save: 1.0366e-05
```

Not sexy

# Stats aggregation

```
# gem 'statsd-ruby', require: 'statsd'

$statsd = Statsd.new(STATSD_HOST, STATSD_PORT)
$statsd.namespace = "com.mycompany.myproduct.#{Rails.env}"


$statsd.timing(
 "mymodule.#{something}.expensive_method",
 (time*1000).round
)
```

# Graphing FTW!

---

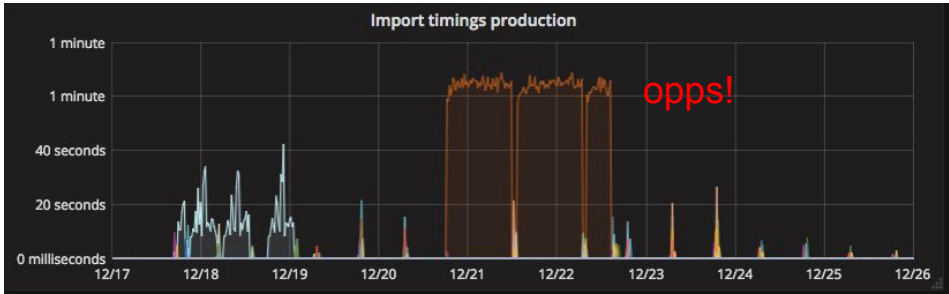
## Hosted solutions:

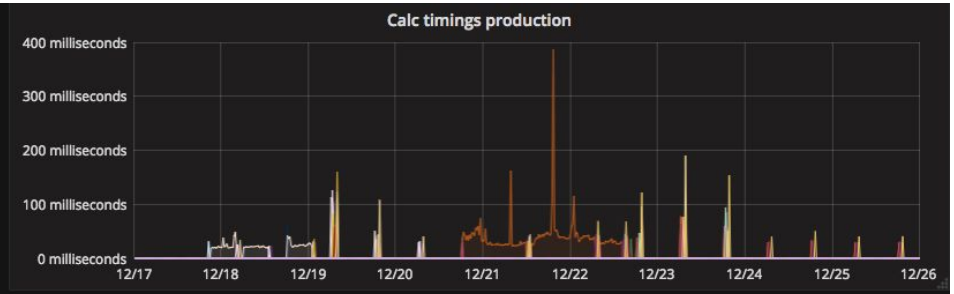
NewRelic, DataDog, Logjam (https://demo.logjam.io)

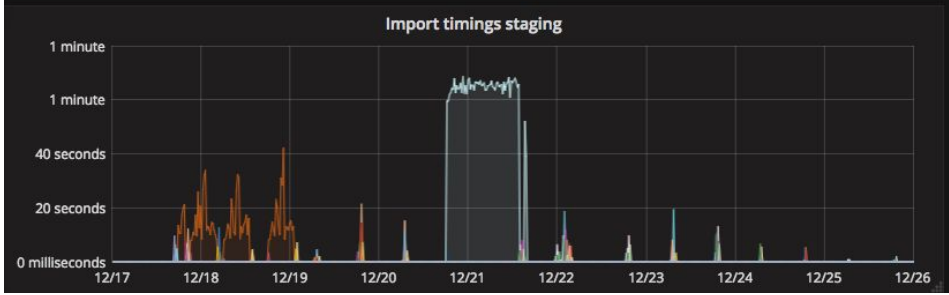
## Custom builds:

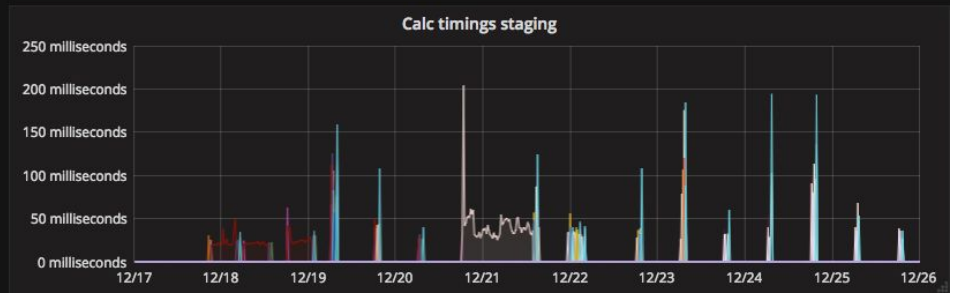
timers from Benchmark => [aggregator] => [graph]

- StatsD/Graphite => Grafana
- InfluxData/InfluxDB => Grafana


Import timings production
1 minute
1 minute
40 seconds
20 seconds
0 milliseconds
opps!
12/17
12/18
12/19
12/20
12/21
12/22
12/23
12/24
12/25
12/26


Calc timings production
400 milliseconds
300 milliseconds
200 milliseconds
100 milliseconds
0 milliseconds
12/17
12/18
12/19
12/20
12/21
12/22
12/23
12/24
12/25
12/26


Import timings staging
1 minute
1 minute
40 seconds
20 seconds
0 milliseconds
12/17
12/18
12/19
12/20
12/21
12/22
12/23
12/24
12/25
12/26


Calc timings staging
250 milliseconds
200 milliseconds
150 milliseconds
100 milliseconds
50 milliseconds
0 milliseconds
12/17
12/18
12/19
12/20
12/21
12/22
12/23
12/24
12/25
12/26

# Final thoughts

---

When?

Rules of thumb:

- run in production (!= live)
- start with stackprof, adjust interval
- wrap more code or switch to ruby-prof if stackprof is accurate enough
- use rack-miniprofiler for http requests with html responses
- use stackprof/ruby-prof for {background,worker,scheduled} jobs
- disable GC

# Thank you!

---

Questions?

Janusz Mordarski
janusz@mordarski.eu
https://github.com/januszm
https://www.linkedin.com/in/janusz-mordarski/
http://mordarski.eu