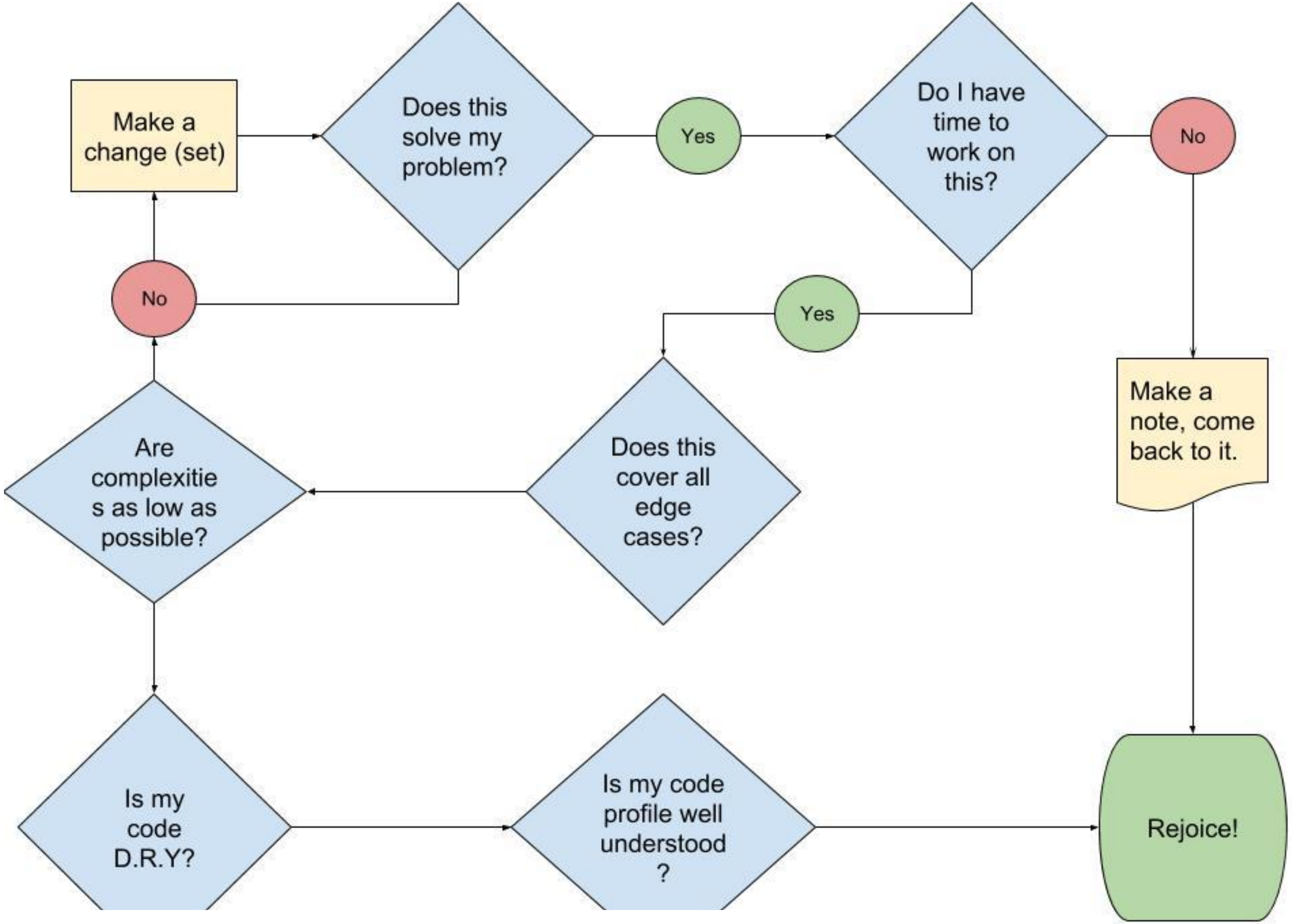
Code Profiling

(With a focus on Perl and Python)

Author: Joseph Wysk


Make a change (set)
Does this solve my problem?
Yes
Do I have time to work on this?
No
No
Yes
Make a note, come back to it.
Are complexitie s as low as possible?
Does this cover all edge cases?
Is my code D.R.Y?
Is my code profile well understood ?
Rejoice!

Adapted from https://medium.freecodecamp.org/time-is-complex-but-priceless-f0abd015063c

Taking a step beyond time*

```
> which time
/usr/bin/time
> man time
```

“The time command runs the specified program command with the given arguments.
When command finishes, time writes a message to standard error giving timing statistics about this program run.
These statistics consist of

(i) the elapsed real time between invocation and termination,

(ii) the user CPU time (the sum of the tms_utime and tms_cutime values in a struct tms as returned by times(2))

(iii) the system CPU time (the sum of the tms_stime and tms_cstime values in a struct tms as returned by times(2)).”

Perl

Perl: Out-of-the-box - Devel::DProf*

```
# The produces a rather non-human friendly file, “tmon.out”
> perl -d:DProf example.pl

# We can extract something human readable from tmon.out by simply calling
> dprofpp

# Or you can cut to the chase with
> dprofpp -p example.pl

# As you’d expect, you can learn about dprofpp via
> perldoc -F /usr/bin/dprofpp
```

http://perldoc.perl.org/Devel/DProf.html

*Although fun fact; this is deprecated and even Perldoc.org recommends using Devel::NTYProf

tmon.out:

```
#fOrTyTwO
$hz=100;
$XS_VERSION='DProf 20080331.00';
# All values are given in HZ
$over_utime=2; $over_stime=-1; $over_rtime=3;
$over_tests=10000;
$rrun_utime=10; $rrun_stime=0; $rrun_rtime=12;
$total_marks=2338

PART2
& 2 main BEGIN
+ 2
- 2
& 3 main BEGIN
+ 3
@ 0 0 1
& 4 strict bits
+ 4
- 4
& 5 strict import
+ 5
- 5
- 3
& 6 main BEGIN
+ 6
```

dprofpp:

```
Total Elapsed Time = 0.107659 Seconds
  User+System Time = 0.117659 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 25.5   0.030  0.030       3   0.0100  0.0100  utf8::SWASHNEW
 17.0   0.020  0.020      36   0.0006  0.0006  main::__ANON__
 17.0   0.020  0.020      36   0.0006  0.0006  Safe::share_from
 8.50   0.010  0.010       1   0.0100  0.0100  utf8::AUTOLOAD
 8.50   0.010  0.010      15   0.0007  0.0006  Safe::BEGIN
 8.50   0.010  0.029      36   0.0003  0.0008  Safe::rdo
 0.00       - -0.000       1        -       -  main::check_vol
 0.00       - -0.000       1        -       -  List::Util::bootstrap
 0.00       - -0.000       1        -       -  B::bootstrap
 0.00       - -0.000       1        -       -  subs::import
 0.00       - -0.000       1        -       -  Opcode::bootstrap
 0.00       - -0.000       1        -       -  Opcode::opset_to_ops
 0.00       - -0.000       1        -       -  UNIVERSAL::VERSION
 0.00       - -0.000       1        -       -  utf8::upgrade
```

Perl: w/ Devel::NYTProf

```
# profile code and write database to ./nytprof.out
> perl -d:NYTProf some_perl.pl

# convert database into a set of html files, e.g., ./nytprof/index.html
# and open a web browser on the nytprof/index.html file
> nytprofhtml --open

# or into comma separated files, e.g., ./nytprof/*.csv
> nytprofcsv
```

From: https://metacpan.org/pod/Devel::NYTProf

Perl: w/ Devel::NYTProf (cont'd.)

# For a sufficient example we'll go on a field trip
http://timbunce.github.io/devel-nytprof/sample-report/nytprof-20160319/index.html

Perl: w/ Devel::NYTProf (cont'd.)

"The NYTProf profiler is written almost entirely in C and great care has been taken to ensure it's very efficient." - https://metacpan.org/pod/Devel::NYTProf

In case this wasn't enough on it's own, there's a module extension for Apache

#Just add one line near the start of your httpd.conf file:

```
> PerlModule Devel::NYTProf::Apache
```

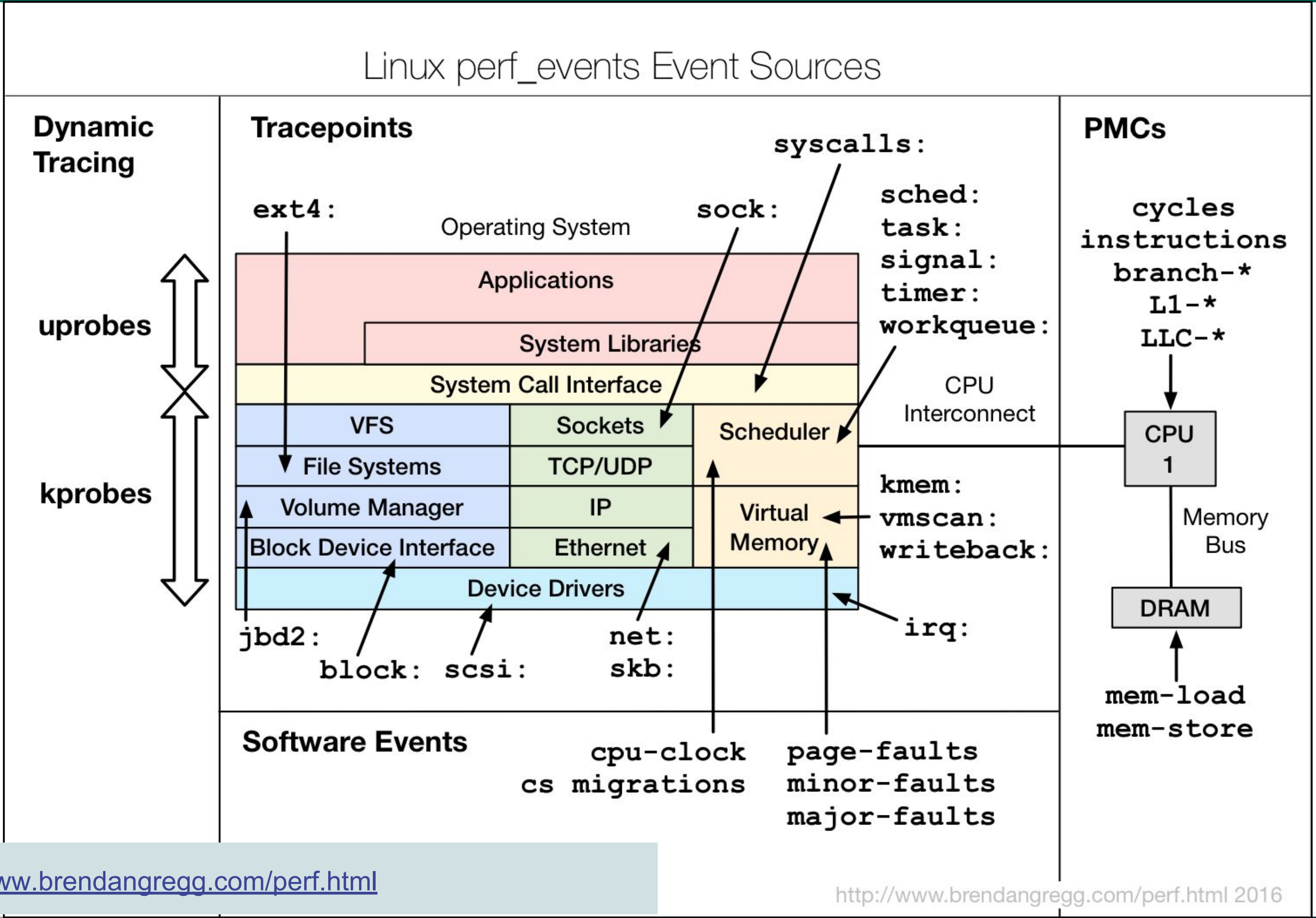
From: https://metacpan.org/pod/Devel::NYTProf::Apache

Taking a (perf)ect detour

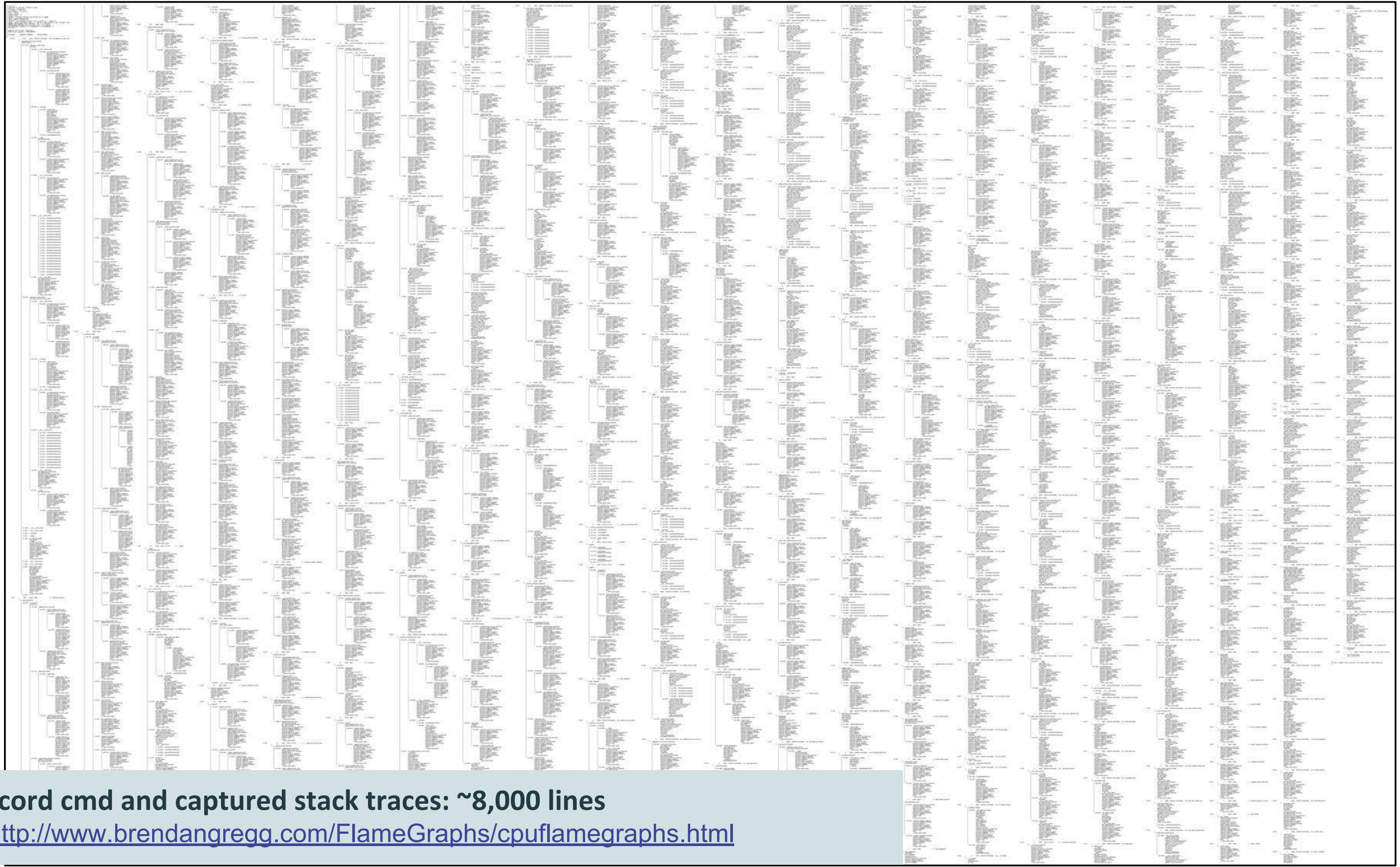
```
> which perf?
```

“...perf is powerful: it can instrument CPU performance counters, tracepoints, kprobes, and uprobes (dynamic tracing). It is capable of lightweight profiling.
It is also included in the Linux kernel, under tools/perf, and is frequently updated and enhanced.

perf began as a tool for using the performance counters subsystem in Linux, and has had various enhancements to add tracing capabilities.” - https://perf.wiki.kernel.org/index.php/Main_Page



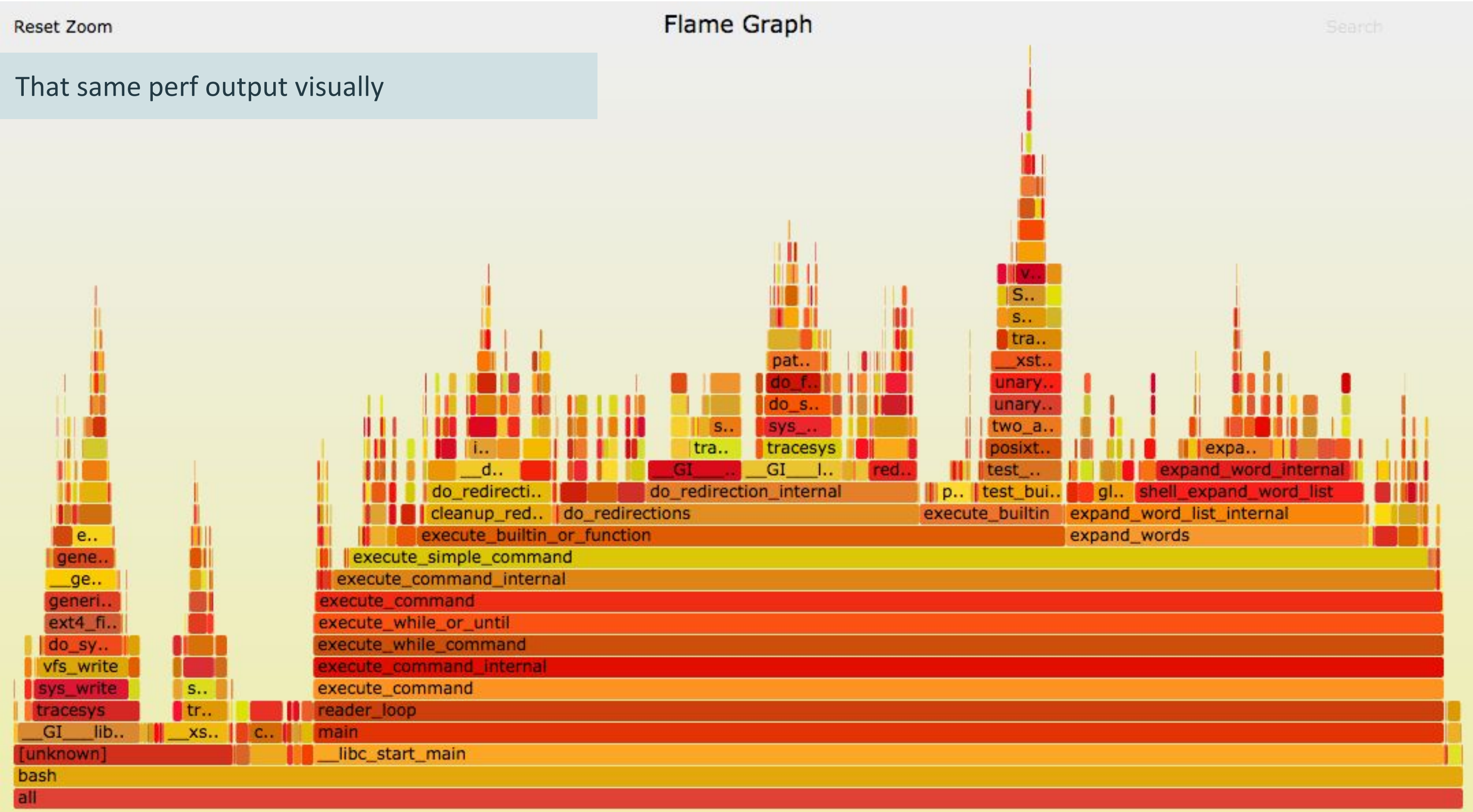
From: http://www.brendangregg.com/perf.html



perf record cmd and captured stack traces: ~8,000 lines

From: http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html

That same perf output visually



From: http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html

Perl: w/ Benchmark

Benchmark is a fantastic module for which I really should have an example.

Unfortunately, I don’t at this time, so I highly encourage those interested to visit the metacpan page.

https://metacpan.org/pod/Benchmark

Python

Python: Out-of-the-box - cProfile

```
# Basic usage
> import cProfile
> cProfile.run('myFunction()', 'myFunction.profile')

# We can extract something human readable via
> import pstats
> stats = pstats.Stats('myFunction.profile')
> stats.strip_dirs().sort_stats('time').print_stats()

# You can also output a profile file for future use
> python -m cProfile -o example.prof example.py
```

Also see: https://docs.python.org/2/library/profile.html

```
Tue Sep 25 17:28:06 2018scicig_profiling.out

         20096 function calls (20095 primitive calls) in 0.009 seconds

   Ordered by: cumulative time
   List reduced from 36 to 11 due to restriction <0.3>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      1      0.000 0.000 0.009 0.009 scicig_profiling_example.py:3(<module>)
      1      0.002 0.002 0.005 0.005 scicig_profiling_example.py:17(try_statement_test_case)
      1      0.000 0.000 0.004 0.004 scicig_profiling_example.py:3(if_statement_test_case)
      2/1    0.000 0.000 0.004 0.004 scicig_profiling_example.py:33(shuffle_array)
      1      0.001 0.001 0.004 0.004 random.py:40(<module>)
      10010 0.002 0.000 0.003 0.000 scicig_profiling_example.py:29(mk_array)
      1      0.002 0.002 0.002 0.002 hashlib.py:56(<module>)
      1      0.000 0.000 0.001 0.001 random.py:91(__init__)
      1      0.000 0.000 0.001 0.001 random.py:100(seed)
      10010 0.001 0.000 0.001 0.000 {method 'append' of 'list' objects}
      1      0.000 0.000 0.000 0.000 {function seed at 0x7f6597c6b500}
```

Python: - PyCallGraph

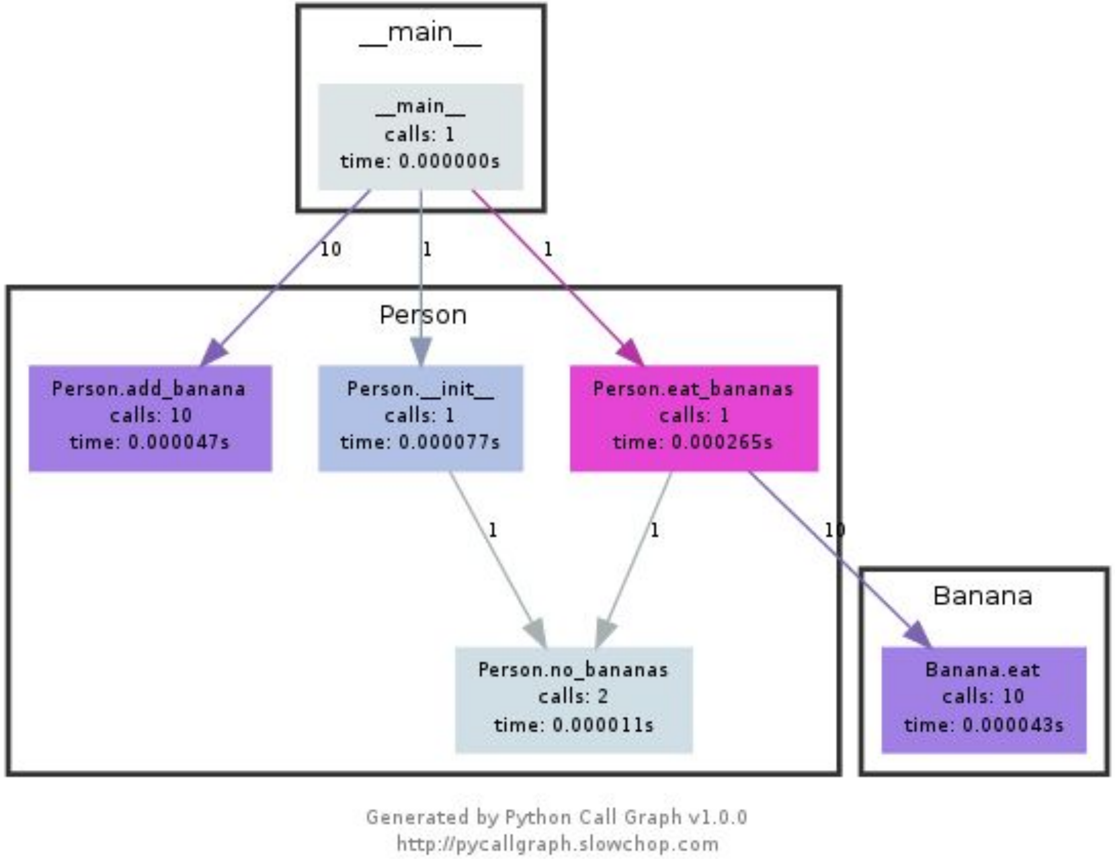
```
# This will generate an output *png callgraph
> pycallgraph graphviz -o output_graph.png ./example.py

# Perhaps use this if you want to bake something in or run on the fly
> from pycallgraph import PyCallGraph
> from pycallgraph.output import GraphvizOutput
```

From: https://pycallgraph.readthedocs.io/en/master/

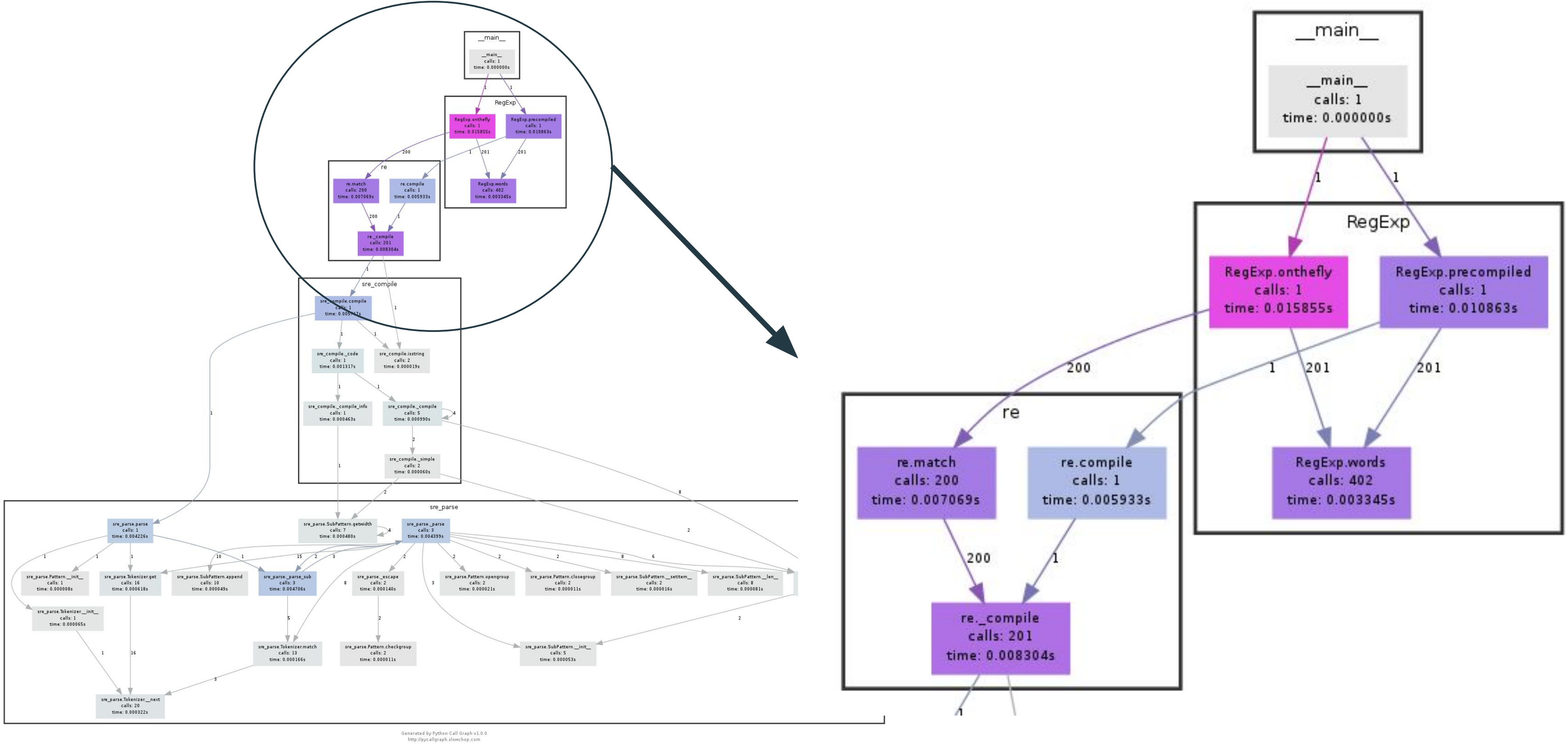
```
from pycallgraph import PyCallGraph
from pycallgraph.output import GraphvizOutput

class Banana:
    def eat(self):
        pass

class Person:
    def __init__(self):
        self.no_bananas()

    def no_bananas(self):
        self.bananas = []

    def add_banana(self, banana):
        self.bananas.append(banana)

    def eat_bananas(self):
        [banana.eat() for banana in self.bananas]
        self.no_bananas()
)
```

```
def main():
    graphviz = GraphvizOutput()
    graphviz.output_file = 'basic.png'

    with PyCallGraph(output=graphviz):
        person = Person()
        for a in xrange(10):
            person.add_banana(Banana())
        person.eat_bananas()

if __name__ == '__main__':
    main()
```

From: https://pycallgraph.readthedocs.io/en/master/examples/basic.html



https://pycallgraph.readthedocs.io/en/master/examples/basic.html

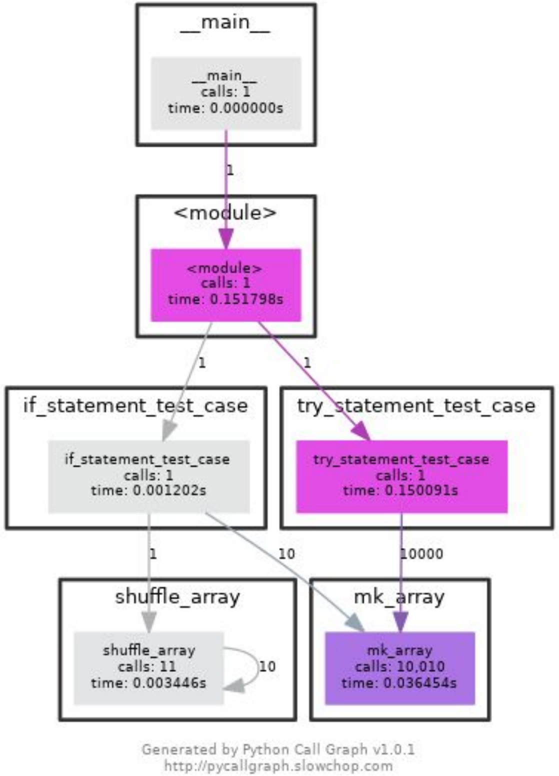


From: https://pycallgraph.readthedocs.io/en/master/examples/regexp_grouped.html

```
def if_statement_test_case():
    sample_array = []
      for x in xrange(0,10):
       if x != 0:
             value = x/x
             mk_array(sample_array, value)
       else:
             value = x*x
             mk_array(sample_array, value)

    shuffle_array(sample_array)
    return sample_array

def mk_array(arr,arr_input):
    arr.append(arr_input)
    return arr

def shuffle_array(arr):
    import random
    random.shuffle(arr)
      if arr[0] != 0:
       shuffle_array(arr)
    return arr
```

```
def try_statement_test_case():
    sample_array = []
      for x in xrange(0,10000):
       try:
             value = x/x
             mk_array(sample_array, value)
       except:
             value = x*x
             mk_array(sample_array, value)
    return sample_array

f __name__== '__main__':
      if_statement_test_case()
      try_statement_test_case()
```

scicig_profiling_example.py



```
time_output:
real    0m0.276s
user    0m0.258s
sys     0m0.017s
> time pycallgraph graphviz -o ./myprofile.png ./scicig_profiling_example.py
```

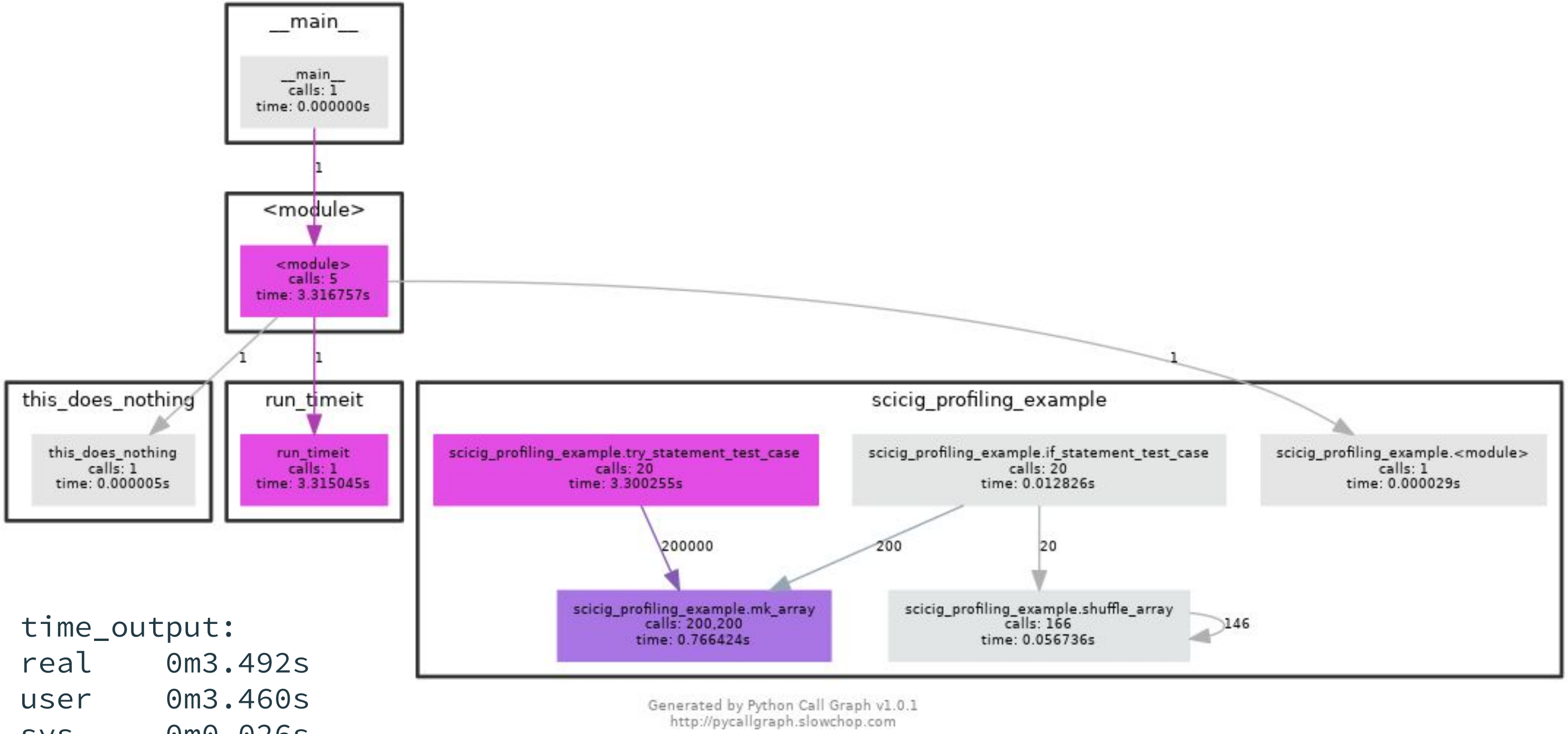
Python: - timeit

- timeit is a very handy module for iterating over snippets of code to establish a consistent execution time

https://docs.python.org/3/library/timeit.html

```
from scicig_profiling_example import if_statement_test_case,try_statement_test_case

import timeit

def run_timeit():
    print min(timeit.repeat(if_statement_test_case, repeat=2, number=10))
    print min(timeit.repeat(try_statement_test_case, repeat=2, number=10))

def this_does_nothing():
    Return

this_does_nothing()

if __name__ == '__main__':
    run_timeit()
```

timeit_test.py



```
time_output:
real    0m3.492s
user    0m3.460s
sys     0m0.026s

> time pycallgraph graphviz -o ./timeit_profile.png ./timeit_test.py
timeit of if_statement_test_case  = 0.00638699531555
timeit of try_statement_test_case = 1.64749479294
```

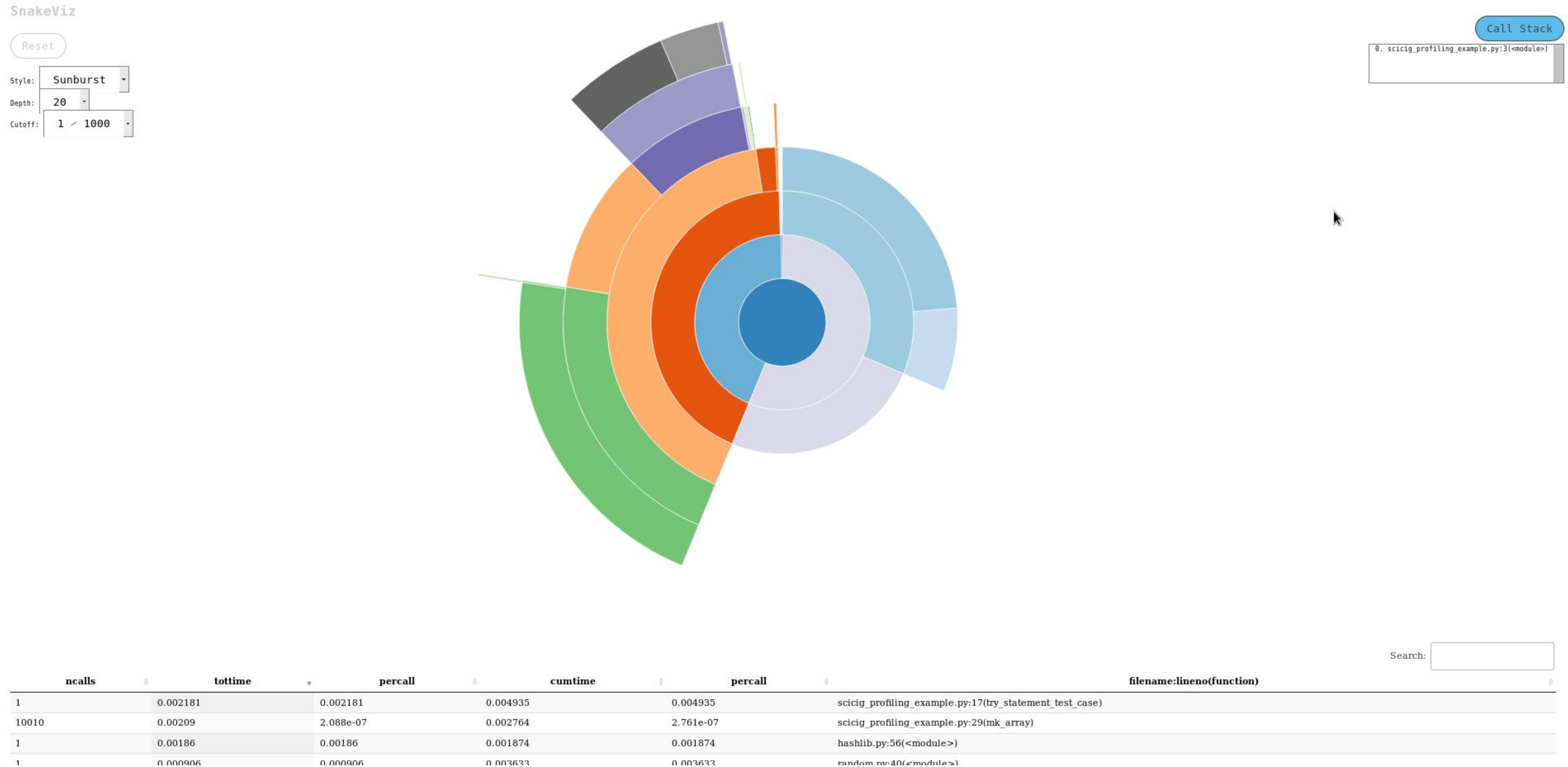
Python: - Anaconda Accelerate (Profiler)

This page on Anaconda Accelerate gives some good history and background to Numba/NumbaPro, which have/had an associated data profiler.

https://www.anaconda.com/blog/developer-blog/open-sourcing-anaconda-accelerate/

As far as I can tell, this data profiler is no longer support, but you can really do the same thing on your own with snakeviz.

https://jiffyclub.github.io/snakeviz/



Style: Sunburst

Depth: 20

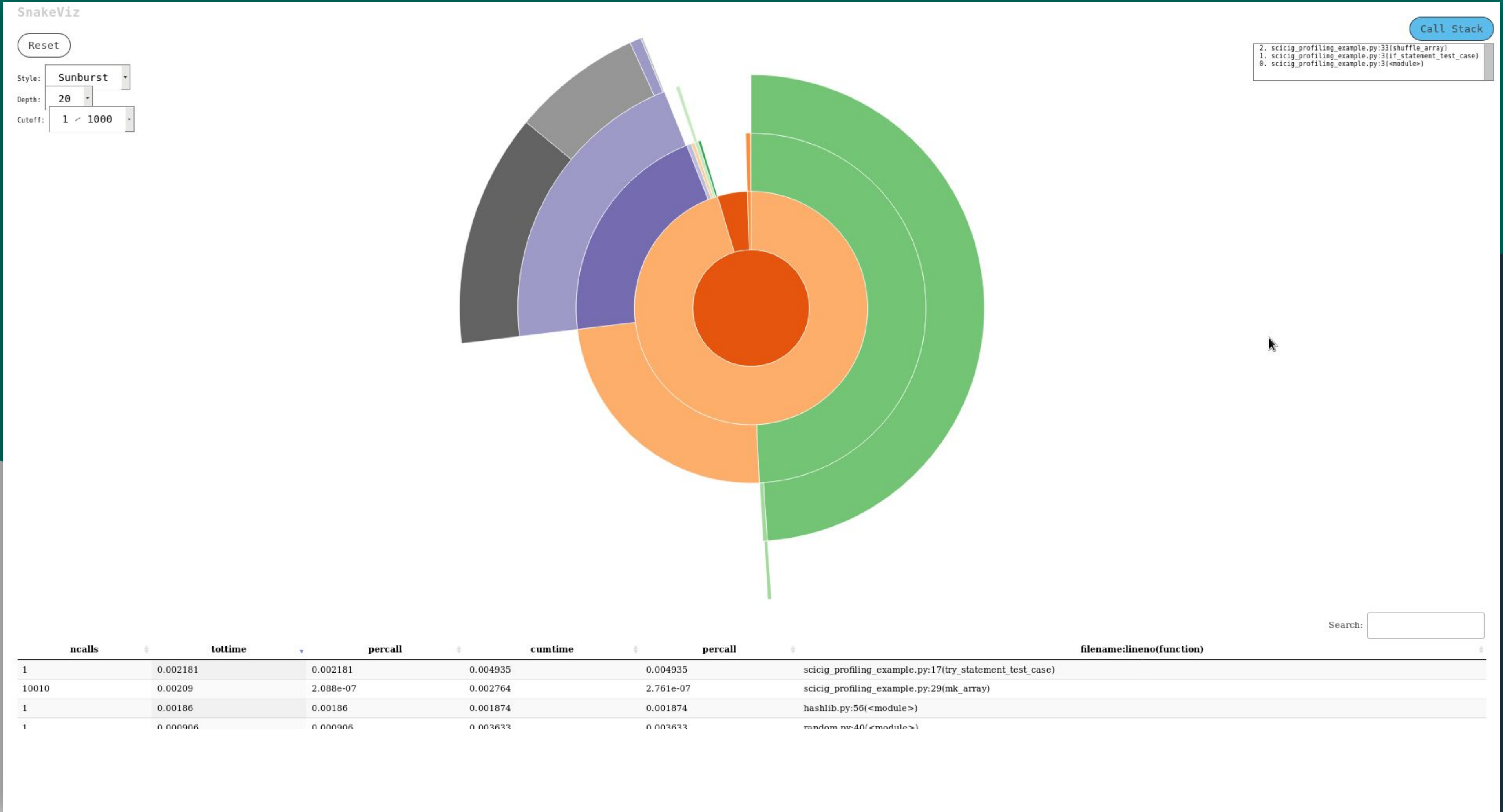
Cutoff: 1 ⁄ 1000

```
0. scicig_profiling_example.py:3(<module>)
```

Search:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.002181	0.002181	0.004935	0.004935	scicig_profiling_example.py:17(try_statement_test_case)
10010	0.00209	2.088e-07	0.002764	2.761e-07	scicig_profiling_example.py:29(mk_array)
1	0.00186	0.00186	0.001874	0.001874	hashlib.py:56(<module>)
1	0.000906	0.000906	0.003633	0.003633	random.py:40(<module>)

```
> python -m cProfile -o scicig_profiling.out ./scicig_profiling_example.py
> snakeviz -s -p 8888 timeit_test.out
```



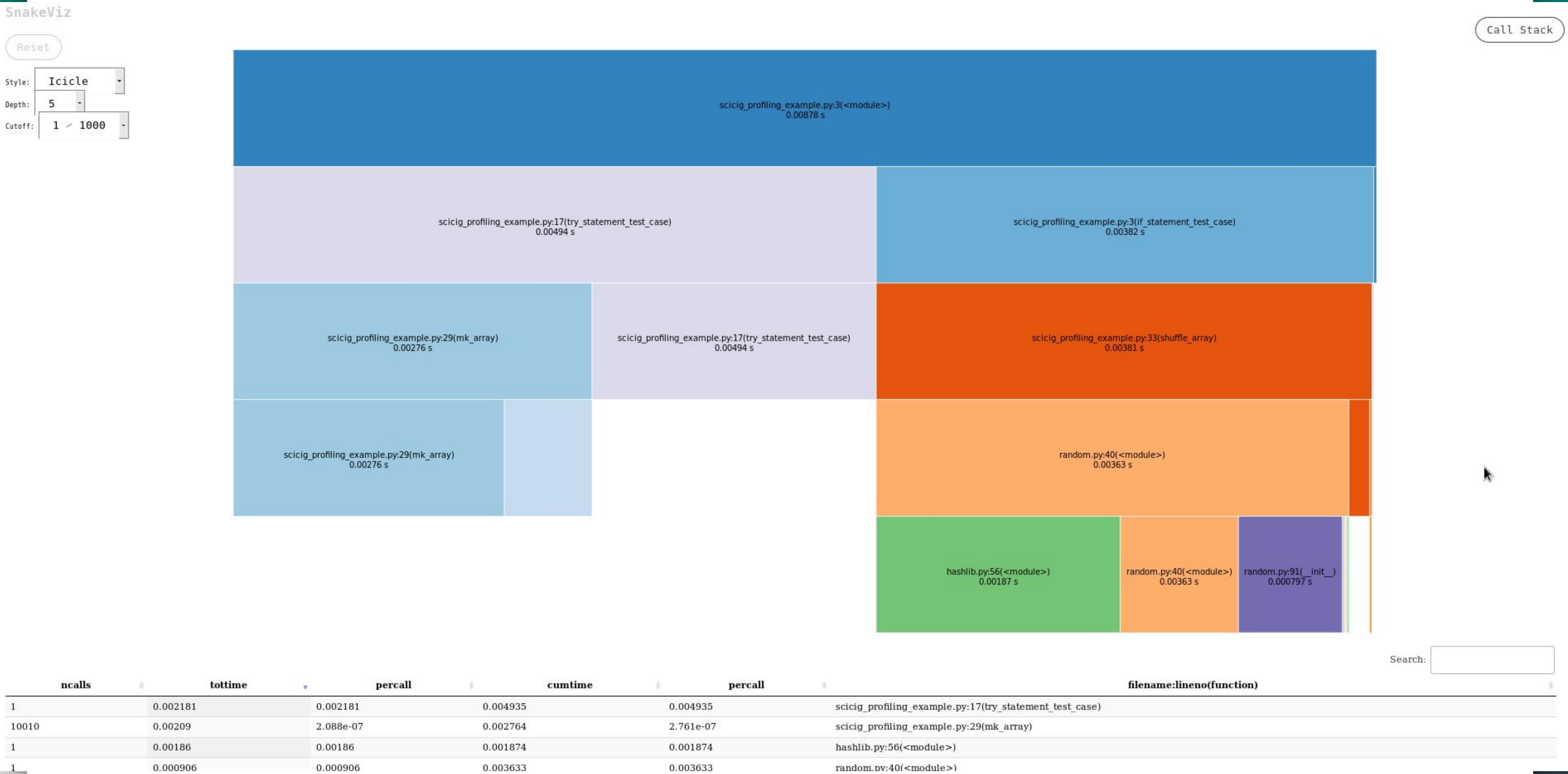
Style: Sunburst

Depth: 20

Cutoff: 1 ⁄ 1000

```
2. scicig_profiling_example.py:33(shuffle_array)
1. scicig_profiling_example.py:3(if_statement_test_case)
0. scicig_profiling_example.py:3(<module>)
```

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.002181	0.002181	0.004935	0.004935	scicig_profiling_example.py:17(try_statement_test_case)
10010	0.00209	2.088e-07	0.002764	2.761e-07	scicig_profiling_example.py:29(mk_array)
1	0.00186	0.00186	0.001874	0.001874	hashlib.py:56(<module>)
1	0.000906	0.000906	0.003633	0.003633	random.py:40(<module>)



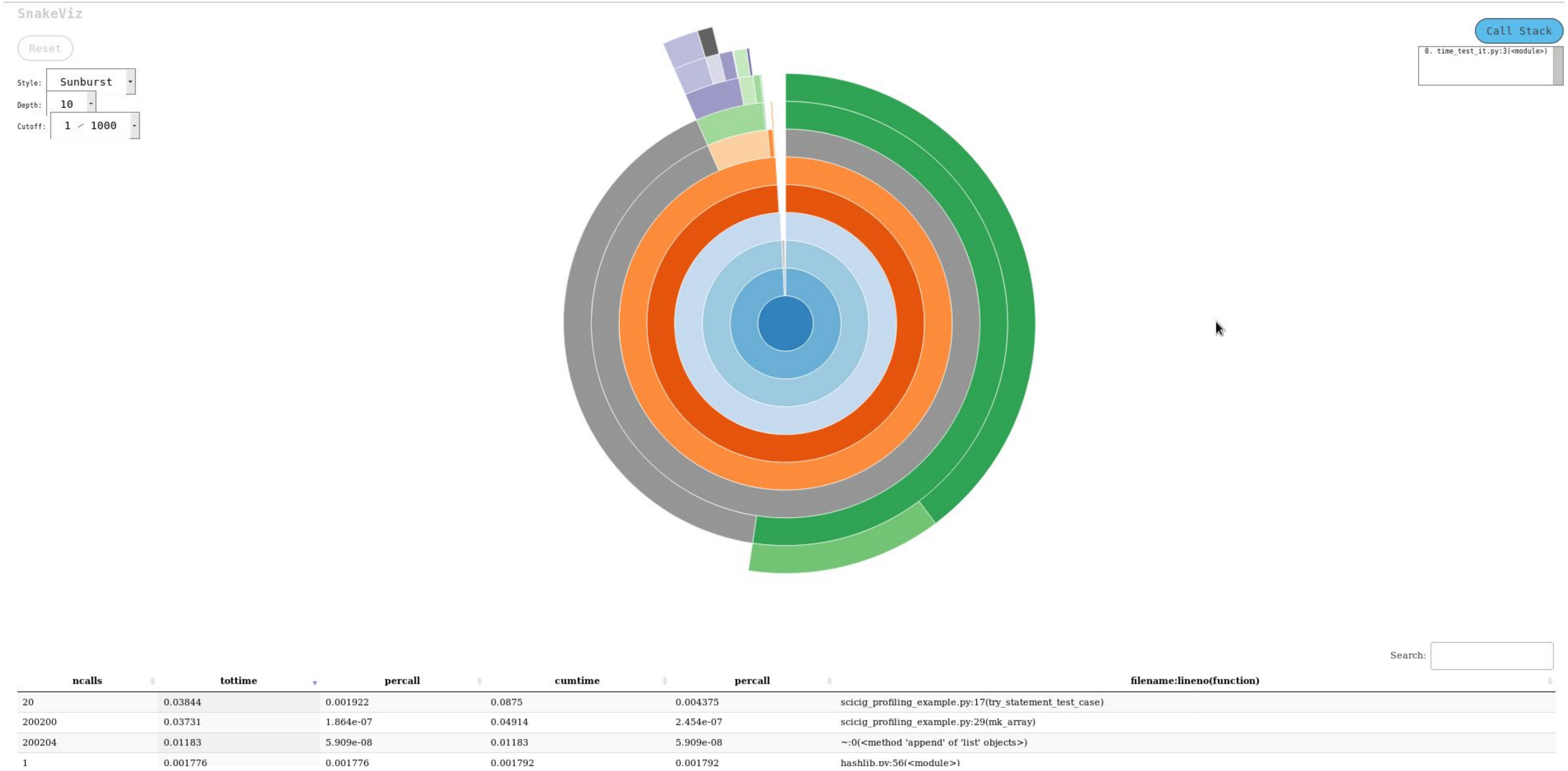
SnakeViz
Reset
Call Stack
scicig_profiling_example.py:3(<module>)
0.00878 s
scicig_profiling_example.py:17(try_statement_test_case)
0.00494 s
scicig_profiling_example.py:3(if_statement_test_case)
0.00382 s
scicig_profiling_example.py:29(mk_array)
0.00276 s
scicig_profiling_example.py:17(try_statement_test_case)
0.00494 s
scicig_profiling_example.py:33(shuffle_array)
0.00381 s
scicig_profiling_example.py:29(mk_array)
0.00276 s
random.py:40(<module>)
0.00363 s
hashlib.py:56(<module>)
0.00187 s
random.py:40(<module>)
0.00363 s
random.py:91(__init__)
0.000797 s
Search:

Style: Icicle

Depth: 5

Cutoff: 1 ⁄ 1000

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.002181	0.002181	0.004935	0.004935	scicig_profiling_example.py:17(try_statement_test_case)
10010	0.00209	2.088e-07	0.002764	2.761e-07	scicig_profiling_example.py:29(mk_array)
1	0.00186	0.00186	0.001874	0.001874	hashlib.py:56(<module>)
1	0.000906	0.000906	0.003633	0.003633	random.py:40(<module>)



Style: Sunburst

Depth: 10

Cutoff: 1 / 1000

0. time_test_it.py:3(<module>)

Search:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
20	0.03844	0.001922	0.0875	0.004375	scicig_profiling_example.py:17(try_statement_test_case)
200200	0.03731	1.864e-07	0.04914	2.454e-07	scicig_profiling_example.py:29(mk_array)
200204	0.01183	5.909e-08	0.01183	5.909e-08	~:0(<method 'append' of 'list' objects>)
1	0.001776	0.001776	0.001792	0.001792	hashlib.py:56(<module>)

```
> python -m cProfile -o timeit_test.out ./timeit_test.py
> snakeviz -s -p 8888 timeit_test.out
```

Great, so now what‽

Python: - Numba

Numba leverages the LLVM compiler library to optimize machine code at runtime, claiming executions times comparable to C and FORTRAN.

Home: https://numba.pydata.org/

Quick start: http://numba.pydata.org/numba-doc/latest/user/jit.html

Examples: https://numba.pydata.org/numba-examples/examples/physics/lennard_jones/results.html

Python: - Numba

```
from numba import jit
@jit(nopython=True, parallel=True)
def jitsum(x):
  sum_val = 0
  for x in xrange(x):
     sum_val = sum_val + x
  return sum_val
%timeit jitsum(1000000)
1000000 loops, best of 3: 217 ns per loop
```

```
def mysum(x):
  sum_val = 0
  for x in xrange(x):
     sum_val = sum_val + x
  return sum_val
%timeit mysum(1000000)
10 loops, best of 3: 28.5 ms per loop
```

Python: - Dask

The emphasis for Dask is making parallel computation in python easy to leverage

Home: http://dask.pydata.org/en/latest/

Examples: https://examples.dask.org/

Tutorial: https://github.com/dask/dask-tutorial

Python: - Dask

```
import dask.array as da

x = da.random.random((10000, 10000),
chunks=(1000, 1000))
%timeit da.random.random((10000, 10000),
chunks=(1000, 1000))
100 loops, best of 3: 2.42 ms per loop
```

```
import numpy as np

x = np.random.rand(10000,10000)
%timeit np.random.rand(10000,10000)
1 loop, best of 3: 824 ms per loop
```

https://towardsdatascience.com/how-i-learned-to-love-parallelized-applies-with-python-pandas-dask-and-numba-f06b0b367138

Perl:

> use Memoize;

“`Memoizing' a function makes it faster by trading space for time. It does this by caching the return values of the function in a table. If you call the function again with the same arguments, memoize jumps in and gives you the value out of the table, instead of letting the function compute the value all over again.”

- https://metacpan.org/pod/Memoize

Advice

- Profiling and optimization tools can be easy to use, but are prone to being finicky
- Make this easy for yourself
- Automate and alias anything and everything
- Keep everything in perspective: Saving a second over a million iterations vs saving 15min every two weeks

“Indeed, one of my major complaints about the computer field is that **whereas Newton could say, ‘If I have seen a little farther than others, it is because I have stood on the shoulders of giants,’ I am forced to say, ‘Today we stand on each other's feet.’”**

- Richard Wesley Hamming

https://en.wikiquote.org/wiki/Richard_Hamming