Hunting Performance in Python Code

twitter.com/sumercip

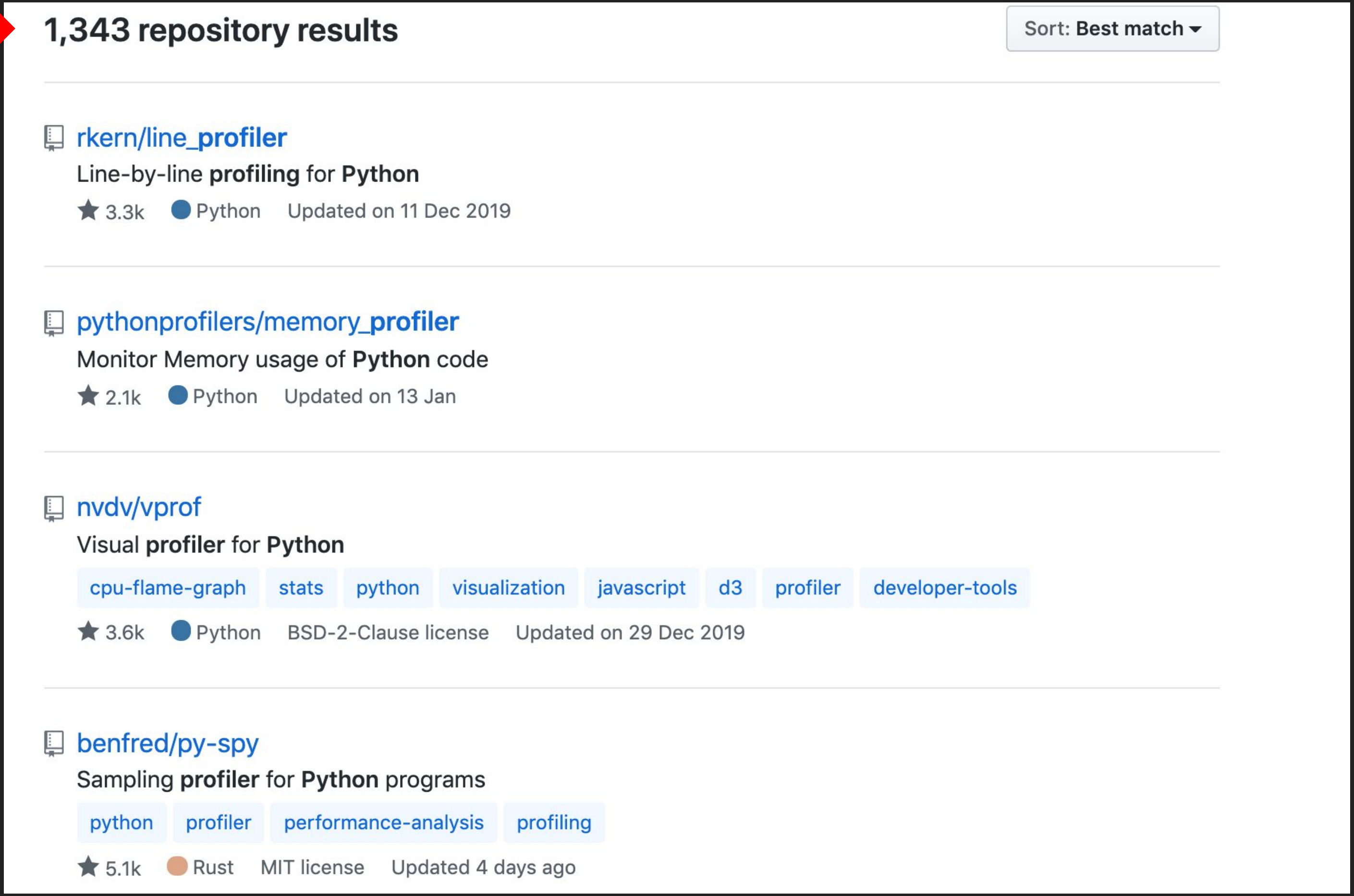
github.com/sumerc

sumercip.com

Outline

- Why?
- How?
- What?
- A slight peek over the ecosystem
- Some common rules for better performance
- Q/A

Performance = Measurement



1,343 repository results

Sort: Best match ▾

rkern/line_profiler
Line-by-line **profiling** for **Python**
★ 3.3k ● Python Updated on 11 Dec 2019

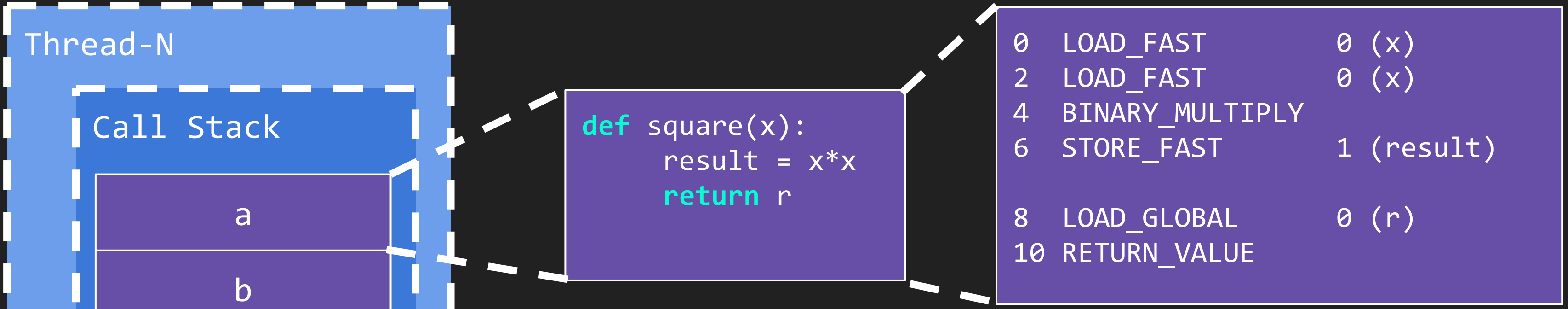
pythonprofilers/memory_profiler
Monitor Memory usage of **Python** code
★ 2.1k ● Python Updated on 13 Jan

nvdv/vprof
Visual **profiler** for **Python**
cpu-flame-graph stats python visualization javascript d3 profiler developer-tools
★ 3.6k ● Python BSD-2-Clause license Updated on 29 Dec 2019

benfred/py-spy
Sampling **profiler** for **Python** programs
python profiler performance-analysis profiling
★ 5.1k ● Rust MIT license Updated 4 days ago

Tracing vs Sampling

TRACING



```
def square(x):
      result = x*x
      return r
```

```
0  LOAD_FAST          0 (x)
2  LOAD_FAST          0 (x)
4  BINARY_MULTIPLY
6  STORE_FAST         1 (result)

8  LOAD_GLOBAL        0 (r)
10 RETURN_VALUE
```

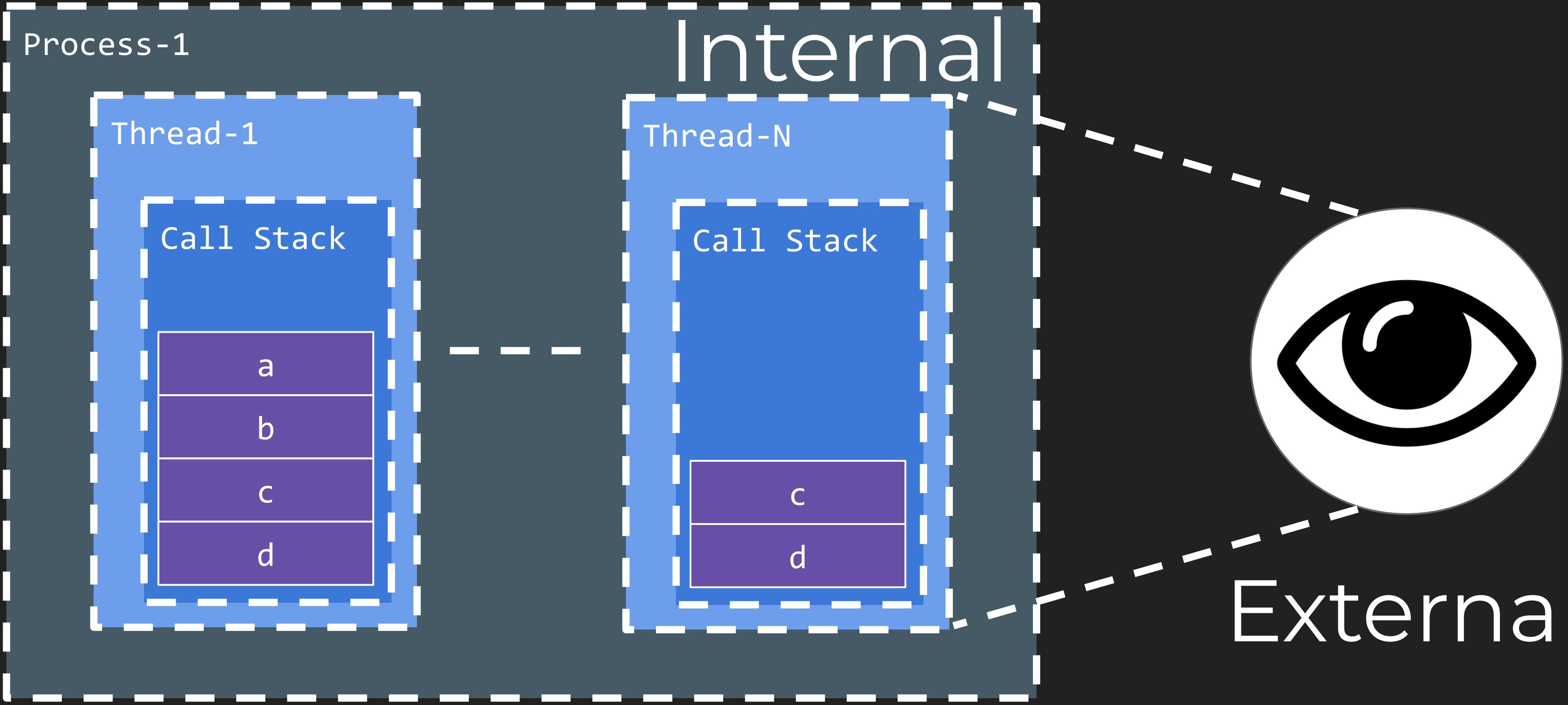
Function

Line

Opcode

(Python3.7 and up)

SAMPLING


Process-1
Internal
Thread-1
Call Stack
a
b
c
d
Thread-N
Call Stack
c
d
External

What does profilers measure?

Time?

Yes. But which time?

Wall

CPU

I/O (Wall - CPU)

More metrics?

Memory

tracemalloc, objgraph

App. specific metrics

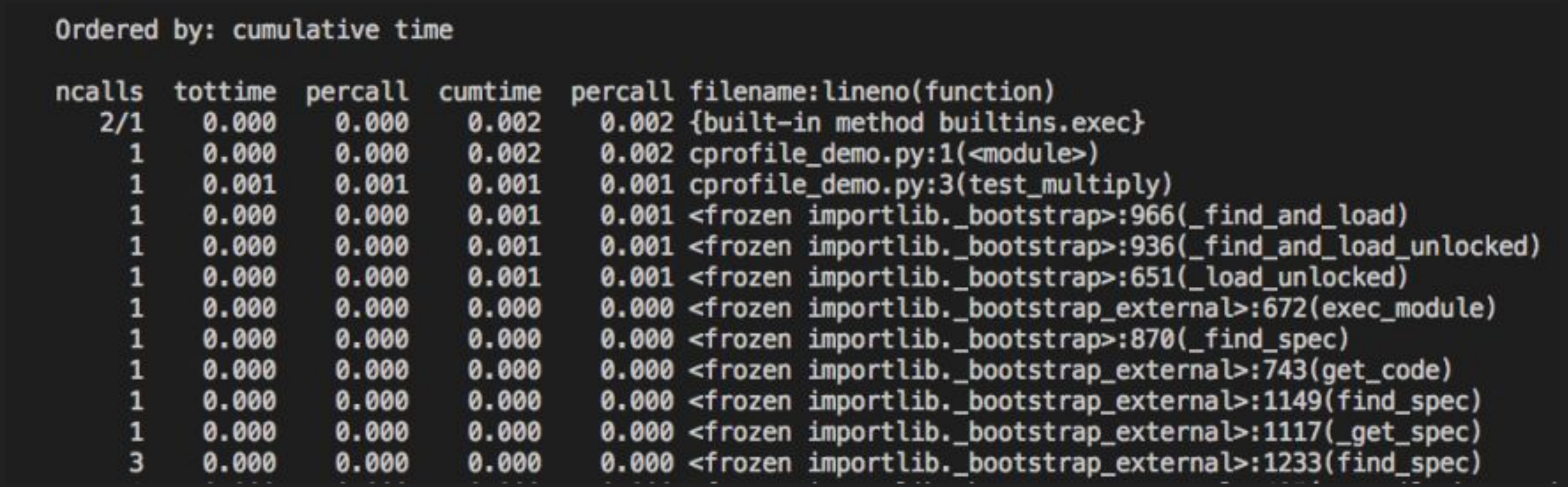
django-debug-toolbar

What does profilers output?

Outputs

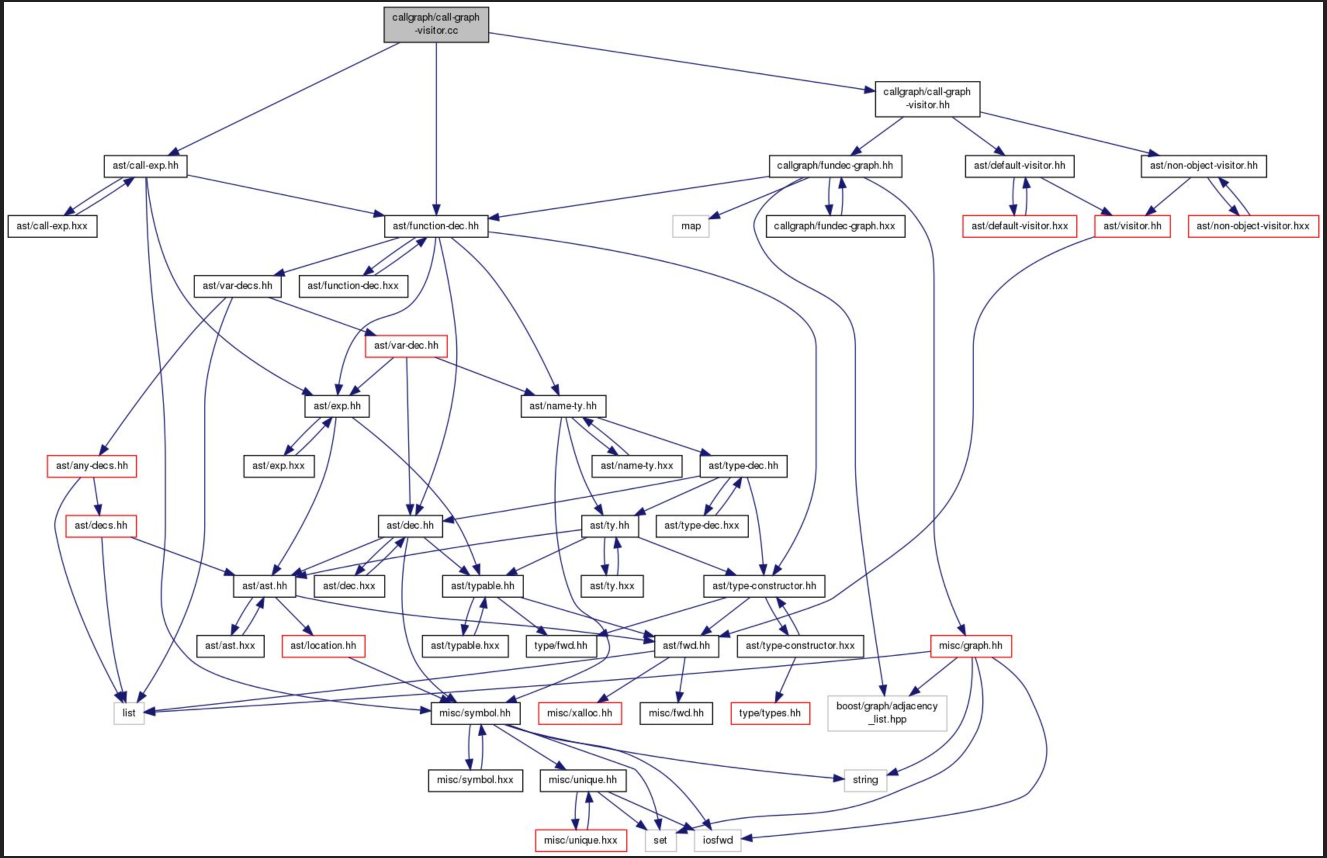
- Table
- Callgraph
- Flamegraph
- Custom formats

Table



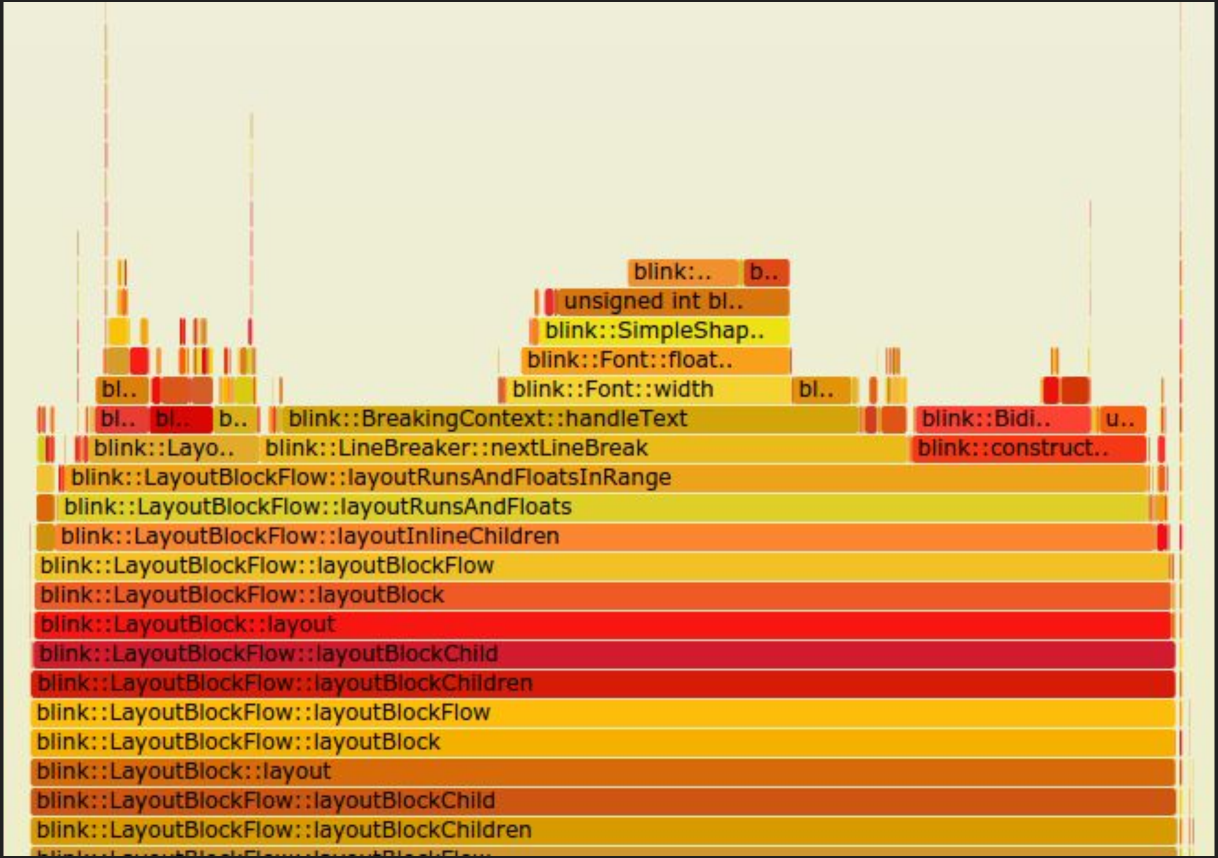
```
Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   2/1    0.000    0.000    0.002    0.002 {built-in method builtins.exec}
     1    0.000    0.000    0.002    0.002 cprofile_demo.py:1(<module>)
     1    0.001    0.001    0.001    0.001 cprofile_demo.py:3(test_multiply)
     1    0.000    0.000    0.001    0.001 <frozen importlib._bootstrap>:966(_find_and_load)
     1    0.000    0.000    0.001    0.001 <frozen importlib._bootstrap>:936(_find_and_load_unlocked)
     1    0.000    0.000    0.001    0.001 <frozen importlib._bootstrap>:651(_load_unlocked)
     1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:672(exec_module)
     1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:870(_find_spec)
     1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:743(get_code)
     1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:1149(find_spec)
     1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:1117(_get_spec)
     3    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:1233(find_spec)
```

Callgraph



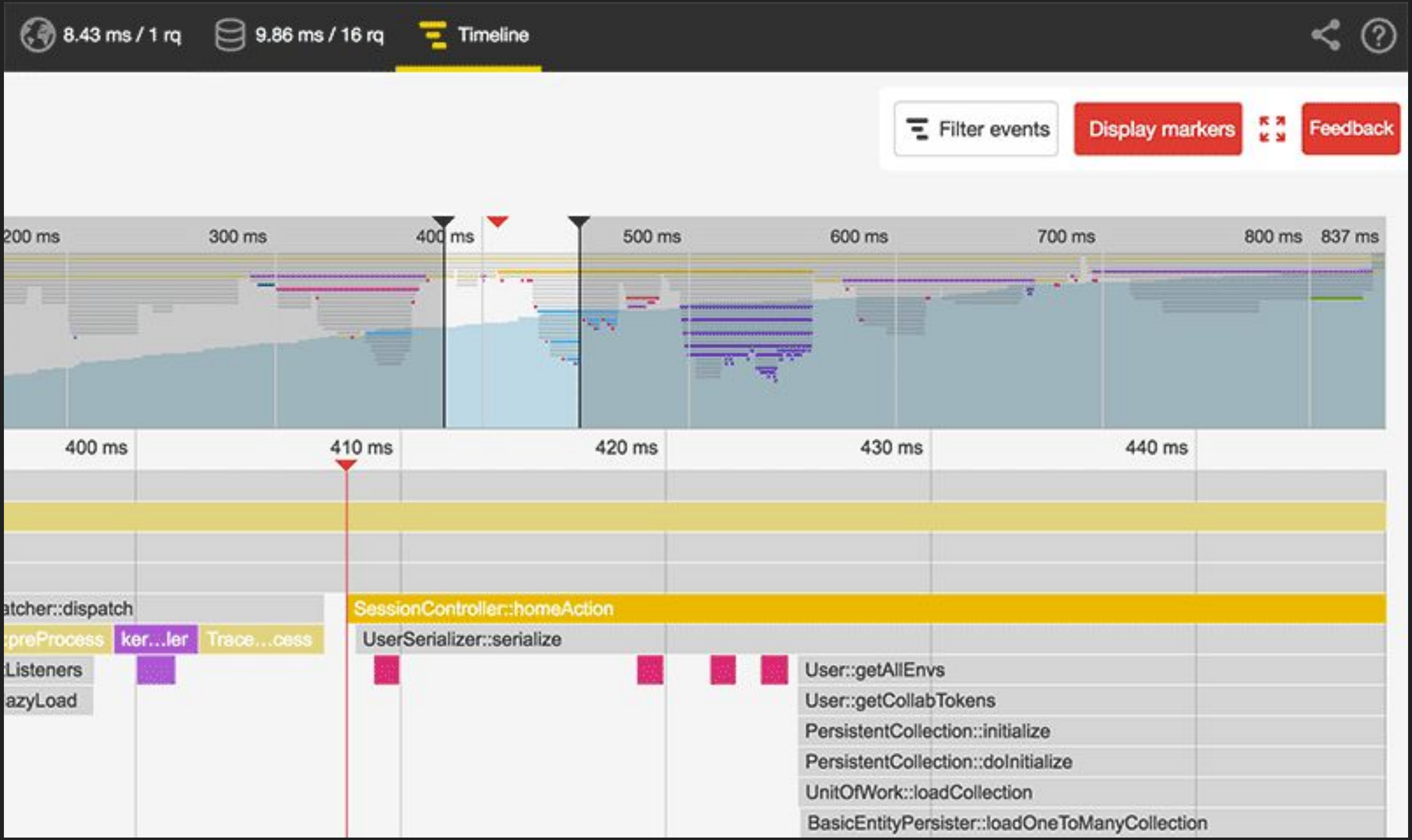
callgraph/call-graph -visitor.cc
callgraph/call-graph -visitor.hh
ast/call-exp.hh
callgraph/fundec-graph.hh
ast/default-visitor.hh
ast/non-object-visitor.hh
ast/call-exp.hxx
ast/function-dec.hh
map
callgraph/fundec-graph.hxx
ast/default-visitor.hxx
ast/visitor.hh
ast/non-object-visitor.hxx
ast/var-decs.hh
ast/function-dec.hxx
ast/var-dec.hh
ast/exp.hh
ast/name-ty.hh
ast/any-decs.hh
ast/exp.hxx
ast/name-ty.hxx
ast/type-dec.hh
ast/decs.hh
ast/dec.hh
ast/ty.hh
ast/type-dec.hxx
ast/ast.hh
ast/dec.hxx
ast/typable.hh
ast/ty.hxx
ast/type-constructor.hh
ast/ast.hxx
ast/location.hh
ast/typable.hxx
type/fwd.hh
ast/fwd.hh
ast/type-constructor.hxx
misc/graph.hh
list
misc/symbol.hh
misc/xalloc.hh
misc/fwd.hh
type/types.hh
boost/graph/adjacency _list.hpp
misc/symbol.hxx
misc/unique.hh
string
misc/unique.hxx
set
iosfwd

Flamegraph



blink::..
b..
unsigned int bl..
blink::SimpleShap..
blink::Font::float..
bl..
blink::Font::width
bl..
bl..
bl..
b..
blink::BreakingContext::handleText
blink::Bidi..
u..
blink::Layo..
blink::LineBreaker::nextLineBreak
blink::construct..
blink::LayoutBlockFlow::layoutRunsAndFloatsInRange
blink::LayoutBlockFlow::layoutRunsAndFloats
blink::LayoutBlockFlow::layoutInlineChildren
blink::LayoutBlockFlow::layoutBlockFlow
blink::LayoutBlockFlow::layoutBlock
blink::LayoutBlock::layout
blink::LayoutBlockFlow::layoutBlockChild
blink::LayoutBlockFlow::layoutBlockChildren
blink::LayoutBlockFlow::layoutBlockFlow
blink::LayoutBlockFlow::layoutBlock
blink::LayoutBlock::layout
blink::LayoutBlockFlow::layoutBlockChild
blink::LayoutBlockFlow::layoutBlockChildren

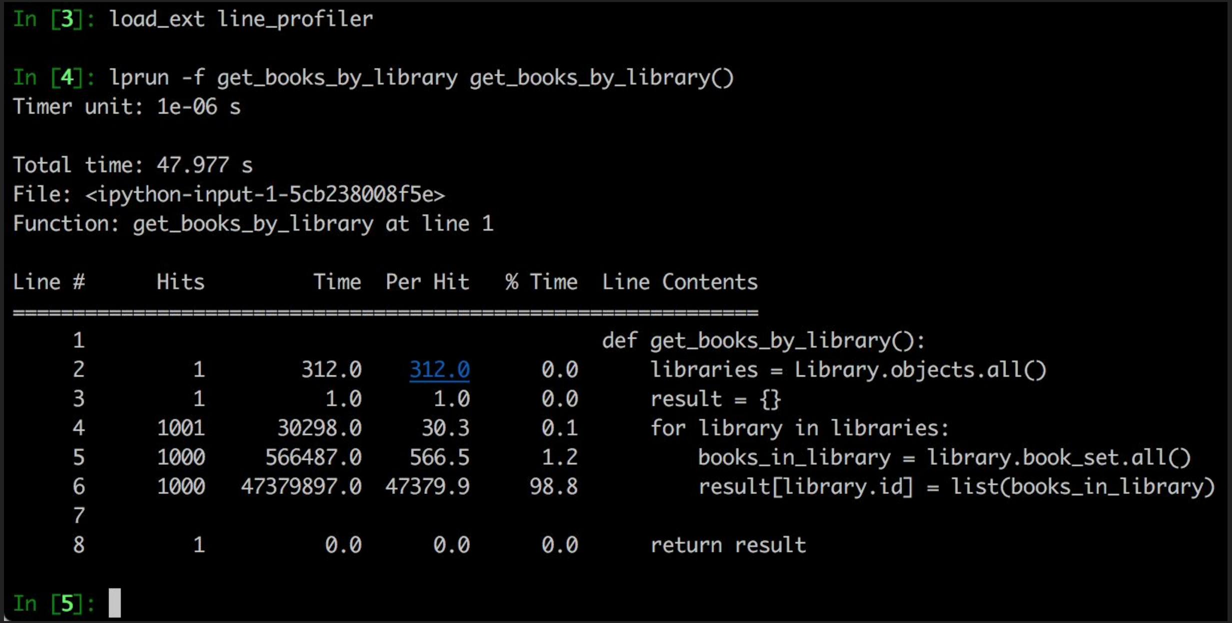
Custom


8.43 ms / 1 rq
9.86 ms / 16 rq
Timeline
Filter events
Display markers
Feedback
200 ms
300 ms
400 ms
500 ms
600 ms
700 ms
800 ms
837 ms
400 ms
410 ms
420 ms
430 ms
440 ms
atcher::dispatch
SessionController::homeAction
:preProcess
ker…ler
Trace…cess
UserSerializer::serialize
Listeners
azyLoad
User::getAllEnvs
User::getCollabTokens
PersistentCollection::initialize
PersistentCollection::doInitialize
UnitOfWork::loadCollection
BasicEntityPersister::loadOneToManyCollection

Examples

line_profiler (line tracing)

- Measures per-line wall time.
- Can inspect its output from cmdline.

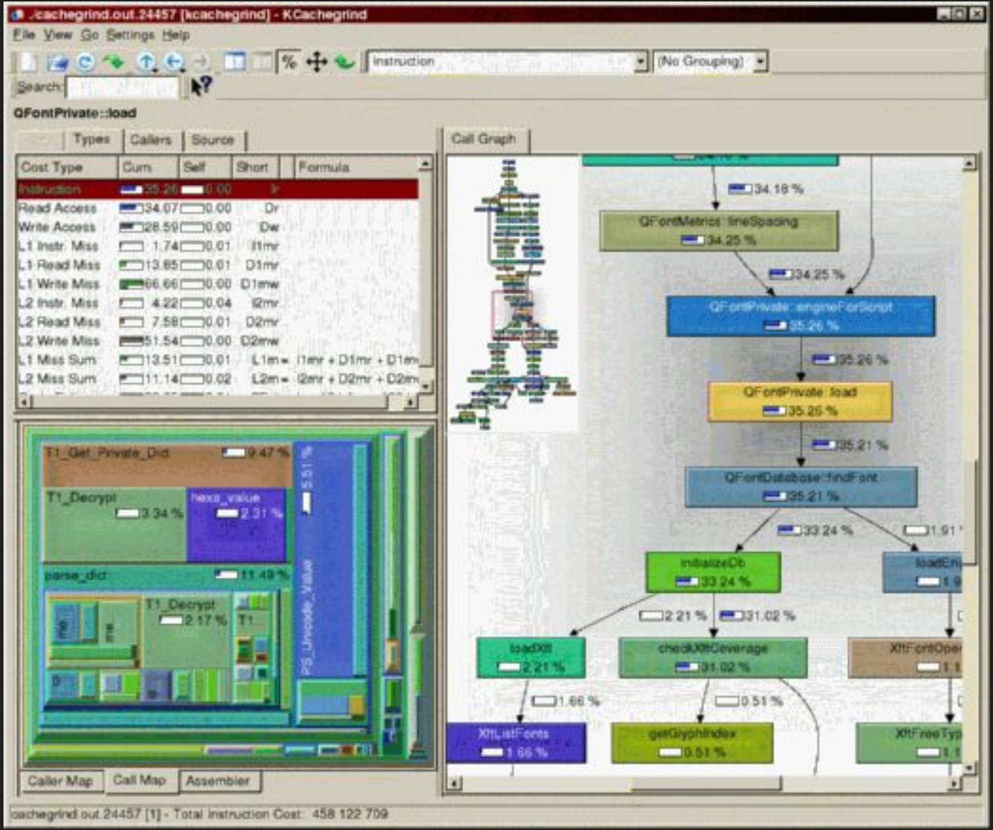


```
In [3]: load_ext line_profiler

In [4]: lprun -f get_books_by_library get_books_by_library()
Timer unit: 1e-06 s

Total time: 47.977 s
File: <ipython-input-1-5cb238008f5e>
Function: get_books_by_library at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def get_books_by_library():
     2         1        312.0    312.0      0.0      libraries = Library.objects.all()
     3         1          1.0      1.0      0.0      result = {}
     4      1001      30298.0     30.3      0.1      for library in libraries:
     5      1000     566487.0    566.5      1.2          books_in_library = library.book_set.all()
     6      1000   47379897.0  47379.9     98.8          result[library.id] = list(books_in_library)
     7
     8         1          0.0      0.0      0.0      return result

In [5]: 
```

Yappi (function tracing)

- Measures per-function wall/CPU time.
- Can profile multithreaded/async applications and show per-thread traces.
- Supports profiling asyncio and gevent(new) applications.
- Can use KCacheGrind to visualize its output as a callgraph.

Yappi (function tracing)



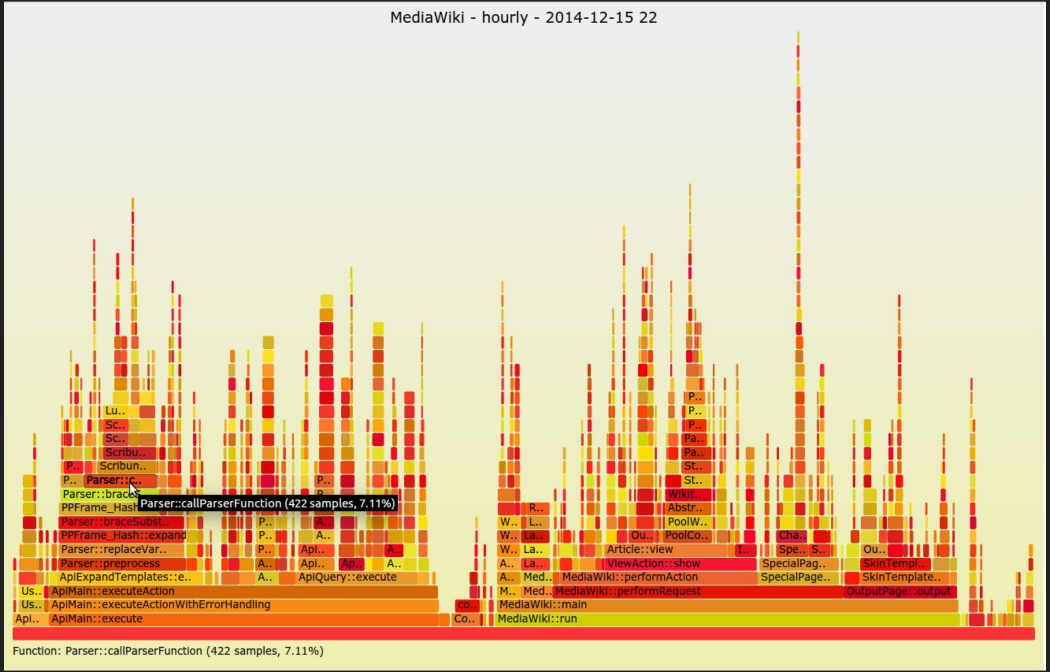
Cost Type	Cum	Self	Short	Formula
Instruction	35.26	0.00	Ir	
Read Access	34.07	0.00	Dr	
Write Access	28.59	0.00	Dw	
L1 Instr. Miss	1.74	0.01	I1mr	
L1 Read Miss	13.85	0.01	D1mr	
L1 Write Miss	66.66	0.00	D1mw	
L2 Instr. Miss	4.22	0.04	I2mr	
L2 Read Miss	7.58	0.01	D2mr	
L2 Write Miss	51.54	0.00	D2mw	
L1 Miss Sum	13.51	0.01	L1m =	I1mr + D1mr + D1m
L2 Miss Sum	11.14	0.02	L2m =	I2mr + D2mr + D2m

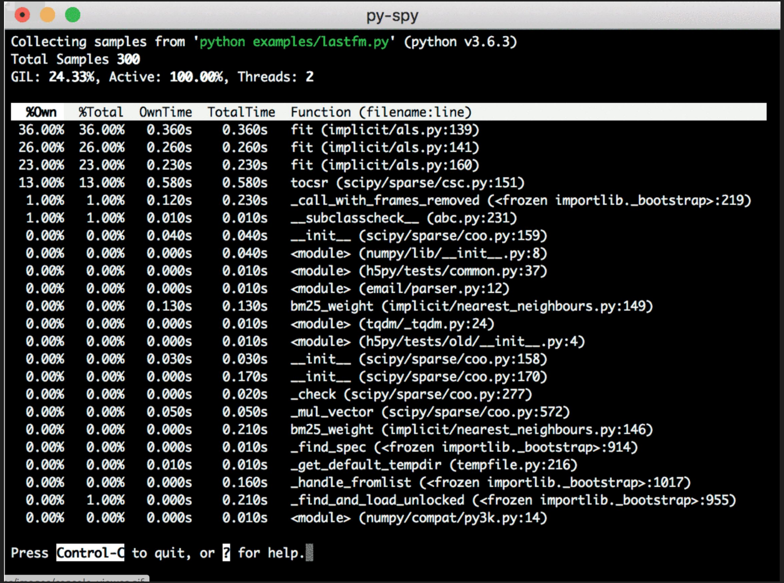
cachegrind.out.24457 [1] - Total Instruction Cost: 458 122 709

py-spy (external sampling)

- Measures per-function CPU time.
- Minimal overhead.(~%10-%15)
- top like CLI, can output to FlameGraph or SpeedScope formats.
- Show GIL contention whenever possible.

py-spy (external sampling)





```
Collecting samples from 'python examples/lastfm.py' (python v3.6.3)
Total Samples 300
GIL: 24.33%, Active: 100.00%, Threads: 2

  %Own   %Total  OwnTime  TotalTime  Function (filename:line)
 36.00%  36.00%   0.360s    0.360s   fit (implicit/als.py:139)
 26.00%  26.00%   0.260s    0.260s   fit (implicit/als.py:141)
 23.00%  23.00%   0.230s    0.230s   fit (implicit/als.py:160)
 13.00%  13.00%   0.580s    0.580s   tocsr (scipy/sparse/csc.py:151)
  1.00%   1.00%   0.120s    0.230s   _call_with_frames_removed (<frozen importlib._bootstrap>:219)
  1.00%   1.00%   0.010s    0.010s   __subclasscheck__ (abc.py:231)
  0.00%   0.00%   0.040s    0.040s   __init__ (scipy/sparse/coo.py:159)
  0.00%   0.00%   0.000s    0.040s   <module> (numpy/lib/__init__.py:8)
  0.00%   0.00%   0.000s    0.010s   <module> (h5py/tests/common.py:37)
  0.00%   0.00%   0.000s    0.010s   <module> (email/parser.py:12)
  0.00%   0.00%   0.130s    0.130s   bm25_weight (implicit/nearest_neighbours.py:149)
  0.00%   0.00%   0.000s    0.010s   <module> (tqdm/_tqdm.py:24)
  0.00%   0.00%   0.000s    0.010s   <module> (h5py/tests/old/__init__.py:4)
  0.00%   0.00%   0.030s    0.030s   __init__ (scipy/sparse/coo.py:158)
  0.00%   0.00%   0.000s    0.170s   __init__ (scipy/sparse/coo.py:170)
  0.00%   0.00%   0.000s    0.020s   _check (scipy/sparse/coo.py:277)
  0.00%   0.00%   0.050s    0.050s   _mul_vector (scipy/sparse/coo.py:572)
  0.00%   0.00%   0.000s    0.210s   bm25_weight (implicit/nearest_neighbours.py:146)
  0.00%   0.00%   0.000s    0.010s   _find_spec (<frozen importlib._bootstrap>:914)
  0.00%   0.00%   0.010s    0.010s   _get_default_tempdir (tempfile.py:216)
  0.00%   0.00%   0.000s    0.160s   _handle_fromlist (<frozen importlib._bootstrap>:1017)
  0.00%   1.00%   0.000s    0.210s   _find_and_load_unlocked (<frozen importlib._bootstrap>:955)
  0.00%   0.00%   0.000s    0.010s   <module> (numpy/compat/py3k.py:14)

Press Control-C to quit, or ? for help.
```

tracemalloc (memory profiler)

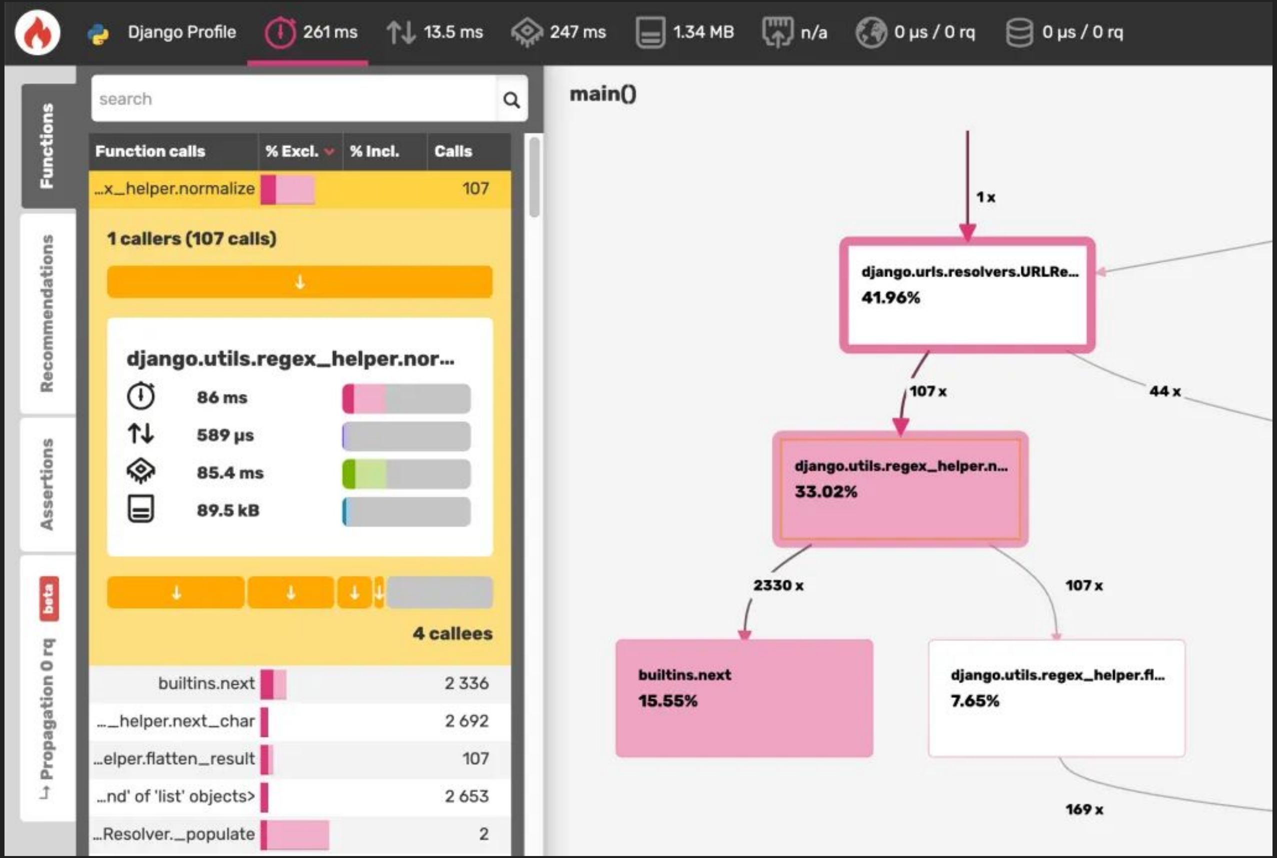
- Included in the stdlib. as of 3.4. (There is a backport for 2.x)
- Traces all malloc/realloc/free calls and saves a traceback along with the allocation.

```
Example output:

tm.py:15: size=4656 B, count=1, average=4656 B
tm.py:14: size=64 B, count=1, average=64 B
tm.py:14: size=40 B, count=1, average=40 B
```

Blackfire **(on-demand tracing)**

- Measures per-function wall/CPU time, memory and some app. specific metrics.
- Enabled only on-demand.
- Profiles microservice requests.
- Has a web application to show its output as callgraph and FlameGraph with time(*Timeline*).

Blackfire (on-demand tracing)



Django Profile · 261 ms · 13.5 ms · 247 ms · 1.34 MB · n/a · 0 µs / 0 rq · 0 µs / 0 rq

Function calls	% Excl.	% Incl.	Calls
...x_helper.normalize			107

1 callers (107 calls)

django.utils.regex_helper.nor...

- 86 ms
- 589 µs
- 85.4 ms
- 89.5 kB

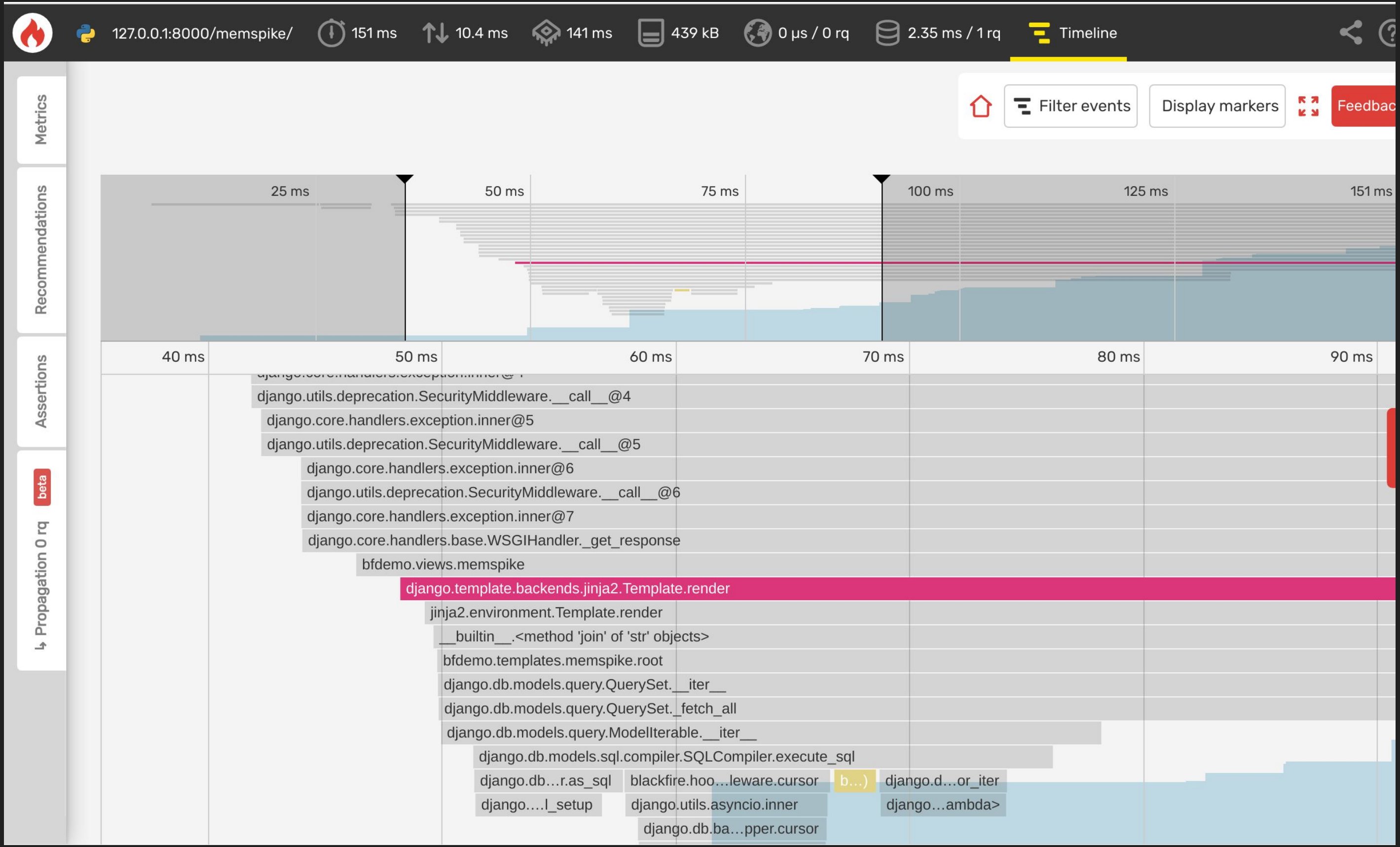
4 callees

builtins.next	2 336
..._helper.next_char	2 692
...elper.flatten_result	107
...nd' of 'list' objects>	2 653
...Resolver._populate	2

main()

1 x
django.urls.resolvers.URLRe...
41.96%
107 x
44 x
django.utils.regex_helper.n...
33.02%
2330 x
107 x
builtins.next
15.55%
django.utils.regex_helper.fl...
7.65%
169 x

Blackfire (on-demand tracing)



```
django.utils.deprecation.SecurityMiddleware.__call__@4
django.core.handlers.exception.inner@5
django.utils.deprecation.SecurityMiddleware.__call__@5
django.core.handlers.exception.inner@6
django.utils.deprecation.SecurityMiddleware.__call__@6
django.core.handlers.exception.inner@7
django.core.handlers.base.WSGIHandler._get_response
bfdemo.views.memspike
django.template.backends.jinja2.Template.render
jinja2.environment.Template.render
__builtin__.<method 'join' of 'str' objects>
bfdemo.templates.memspike.root
django.db.models.query.QuerySet.__iter__
django.db.models.query.QuerySet._fetch_all
django.db.models.query.ModelIterable.__iter__
django.db.models.sql.compiler.SQLCompiler.execute_sql
django.db…r.as_sql   blackfire.hoo…leware.cursor   b…)   django.d…or_iter
django.…l_setup   django.utils.asyncio.inner   django…ambda>
django.db.ba…pper.cursor
```

Common Rules

Common Rules

- Always **measure**.
- Focus first on **Architecture/Design/Algorithms**.
- Know your **data** and how you access it(e.g: asymptotic complexity).
- Check if **standard library** has a function for your case.
- Avoid micro optimizations.
- If you really need performance, then there are tools. **Cython**, **Numba** or directly write code in C.

Thank you!

twitter.com/sumercip
github.com/sumerc
sumercip.com