Flame Graphs for MySQL DBAs

Valerii Kravchuk, Principal Support Engineer, MariaDB

vkravchuk@gmail.com

Who am I and What Do I Do?

Valerii (aka **Valeriy**) **Kravchuk**:

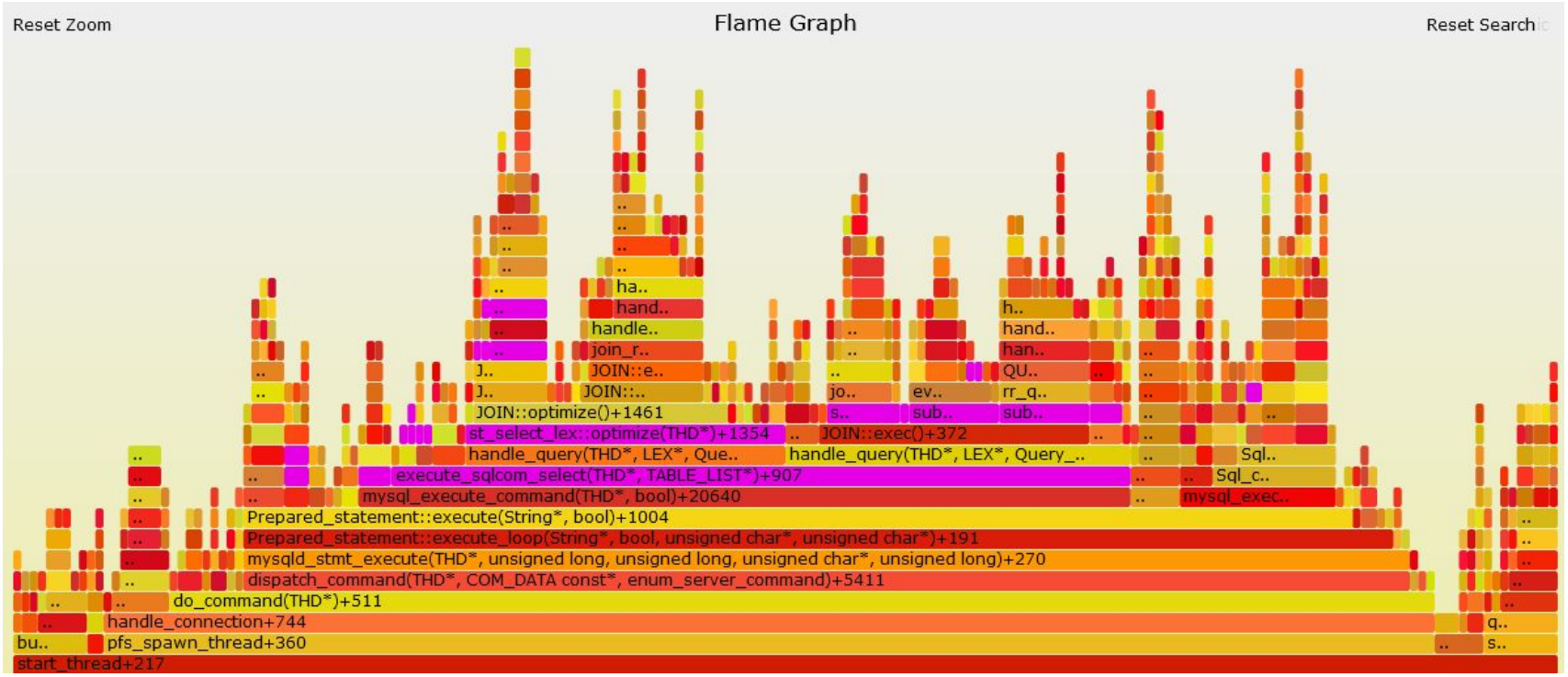
- MySQL Support Engineer in MySQL AB, Sun and Oracle, 2005-2012
- Principal Support Engineer in Percona, 2012-2016
- Principal Support Engineer in MariaDB Corporation since March 2016
- **http://mysqlentomologist.blogspot.com** - my blog about MySQL and MariaDB (including some **HowTo**s, used to be mostly MySQL bugs marketing). See my posts with **flame graphs** used to make some point.
- **https://www.facebook.com/valerii.kravchuk** - my Facebook page
- **http://bugs.mysql.com** - used to be my personal playground
- **@mysqlbugs #bugoftheday** - links to interesting MySQL bugs, few er week
- **MySQL Community Contributor of the Year 2019**
- I speak about MySQL and MariaDB in public. Some slides from previous talks are here and there…
- "I solve problems", "I drink and I know things"

Disclaimers

- Since September, 2012 I act as an Independent Consultant providing services to different companies
- All views, ideas, conclusions, statements and approaches in my presentations and blog posts are mine and may not be shared by any of my previous, current and future employees, customers and partners
- All examples are either based on public information or are truly fictional and has nothing to do with any real persons or companies. Any similarities are pure coincidence :)
- The information presented is true to the best of my knowledge

Flame Graphs: what are they and how they help

- *Flame graphs* are a visualization of profiled software, allowing the most frequent code-paths to be identified quickly and accurately
- Consider this example (PS 5.7.33, **sysbench** read-write, **bpftrace**):



Problem when profiling - overview of the data

- Let's run typical profiling session with **perf** while MySQL runs:

```
openxs@ao756:~$ sudo perf record -a -g -F99 -- sleep 30
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1,144 MB perf.data (1684 samples) ]
openxs@ao756:~$ sudo perf report > perf.out
```

- Here is the output in **perf.out** (small font is in purpose):

```
...
        36.54%    0.00%  mysqld                libpthread-2.31.so           [.] start_thread
        |
        ---start_thread
                  |
                  |--32.59%--pfs_spawn_thread
                  |          |
                  |          --32.55%--handle_connection
                  |                    |
                  |                    --32.45%--do_command
                  |                              |
                  |                              |--30.96%--dispatch_command
                  |                              |          |
                  |                              |          --29.39%--mysqld_stmt_execute
                  |                              |                    |
                  |                              |                    --29.14%--Prepared_statement::execute_loop
                  |                              |                              |
                  |                              |                              --29.01%--Prepared_statement::execute
                  |                              |                                        |
                  |                              |                                        --27.80%--mysql_execute_command
                  |                              |                                                  |
                  |                              |                                                  |--19.09%--execute_sqlcom_select
                  |                              |                                                  |          |
                  |                              |                                                  |          |--18.20%--handle_query
                  |                              |                                                  |          |          |
                  |                              |                                                  |          |          |--9.37%--JOIN::exec
                  |                              |                                                  |          |          |          |
                  |                              |                                                  |          |          |          |--7.65%--sub_select
...
openxs@ao756:~$ ls -l perf.out
-rw-rw-r-- 1 openxs openxs 1109381 кві 25 15:55 perf.out
```

Raw profiling data are just timestamps and stacks

- Let's check raw **perf** data, hardly useful as is:

```
openxs@ao756:~$ openxs@ao756:~$ sudo perf script | more
...
mysqld 143863 [001] 105802.967446:  4744028   cycles:
        557a3c9d08bc insert_events_statements_history+0xac
(/usr/sbin/mysqld
)
        557a3c9c658b pfs_end_statement_v1+0x14ab (/usr/sbin/mysqld)
        557a3c6ec8b7 dispatch_command+0x557 (/usr/sbin/mysqld)
        557a3c6ee79f do_command+0x1ff (/usr/sbin/mysqld)
        557a3c7aecd8 handle_connection+0x2e8 (/usr/sbin/mysqld)
        557a3c9bf2c8 pfs_spawn_thread+0x168 (/usr/sbin/mysqld)
        7f2fea56d609 start_thread+0xd9
(/usr/lib/x86_64-linux-gnu/libpthread
-2.31.so)
...
```

- We still have to summarize them somehow for better overview!

Profiling - challenges and solutions...

- Profiling is basically measuring frequency and duration of function calls, or any resource usage
- For complex software like MySQL or MariaDB servers **perf** (or any other profilers) produces too large data sets to study efficiently
- The answer is filtering (with **grep**), summarizing (with **awk** etc, see how **pt-pmp** does this for **gdb** backtraces, some 120 lines of code) or … visualisation as Heat Maps or Flame Graphs (or some GUI)
- If you care, Windows Performance Analyzer (WPA) also supports flame graphs, and I had to check them :)

Flame Graphs - use free tools by Brendan Gregg

- **http://www.brendangregg.com/flamegraphs.html**
- Flame graphs produced by these tools are a visualization (as **.svg** file to be checked in browser) of profiled software, allowing the most frequent code-paths to be identified quickly and accurately.
- The x-axis shows the stack profile population, sorted *alphabetically* (it is not the passage of time), and the y-axis shows stack depth. Each rectangle represents a stack frame. The wider a frame is, the more often it was present in the stacks.
- **CPU Flame Graphs** ← profiling by sampling at a fixed rate. Check this post.
- **Memory Flame Graphs** ← tracing **malloc()**, **free()**, **brk()**, **mmap()**, **page_fault**
- **Off-CPU Flame Graphs** ← tracing file I/O, block I/O or scheduler
- More (Hot-Cold, Differential, **pt-pmp**-based etc),
- **https://github.com/brendangregg/FlameGraph** + **perf** + ... or **bcc** tools like **offcputime.py**

flamegraph.pl - basic options

```
openxs@ao756:~/git/FlameGraph$ ./flamegraph.pl --help
USAGE: ./flamegraph.pl [options] infile > outfile.svg

    --title TEXT     # change title text
    --subtitle TEXT  # second level title (optional)
    --width NUM      # width of image (default 1200)
    --height NUM     # height of each frame (default 16)
    --minwidth NUM   # omit smaller functions (default 0.1 pixels)
    --fonttype FONT  # font type (default "Verdana")
    --fontsize NUM   # font size (default 12)
    --countname TEXT # count type label (default "samples")
    --nametype TEXT  # name type label (default "Function:")
    --colors PALETTE # set color palette. choices are: hot (default), mem, ...
    --bgcolors COLOR # set background colors. gradient choices are yellow
...
    --hash           # colors are keyed by function name hash
    --cp             # use consistent palette (palette.map)
    --reverse        # generate stack-reversed flame graph
    --inverted       # icicle graph
    --flamechart     # produce a flame chart (sort by time, do not merge ...)
    --negate         # switch differential hues (blue<->red)
    --notes TEXT     # add notes comment in SVG (for debugging)
...
```

flamegraph.pl - expected input format

- Flame graphs can be generated from any profile data that contains “stack traces”. This can be abused to show anything...
- Check comments in the source code for format details:

```
   ...
# The input is stack frames and sample counts formatted as single
# lines.  Each frame in the stack is semicolon separated, with a
# space and count at the end of the line.  These can be generated
# for Linux perf script output using stackcollapse-perf.pl, for
# DTrace using stackcollapse.pl, and for other tools
# using the other stackcollapse programs.  Example input:
#
#  swapper;start_kernel;rest_init;cpu_idle;default_idle;nati... 1
#
# An optional extra column of counts can be provided to generate a
# differential flame graph of the counts, colored red for more,
# and blue for less.  This can be useful when using flame graphs for
# non-regression testing.  See the header comment in the
# difffolded.pl program for instructions.
...
```

Flame Graphs - tools to process stack traces

- Different stack output formats are supported by the tools, including **gdb**, **perf** and **bpftrace**:

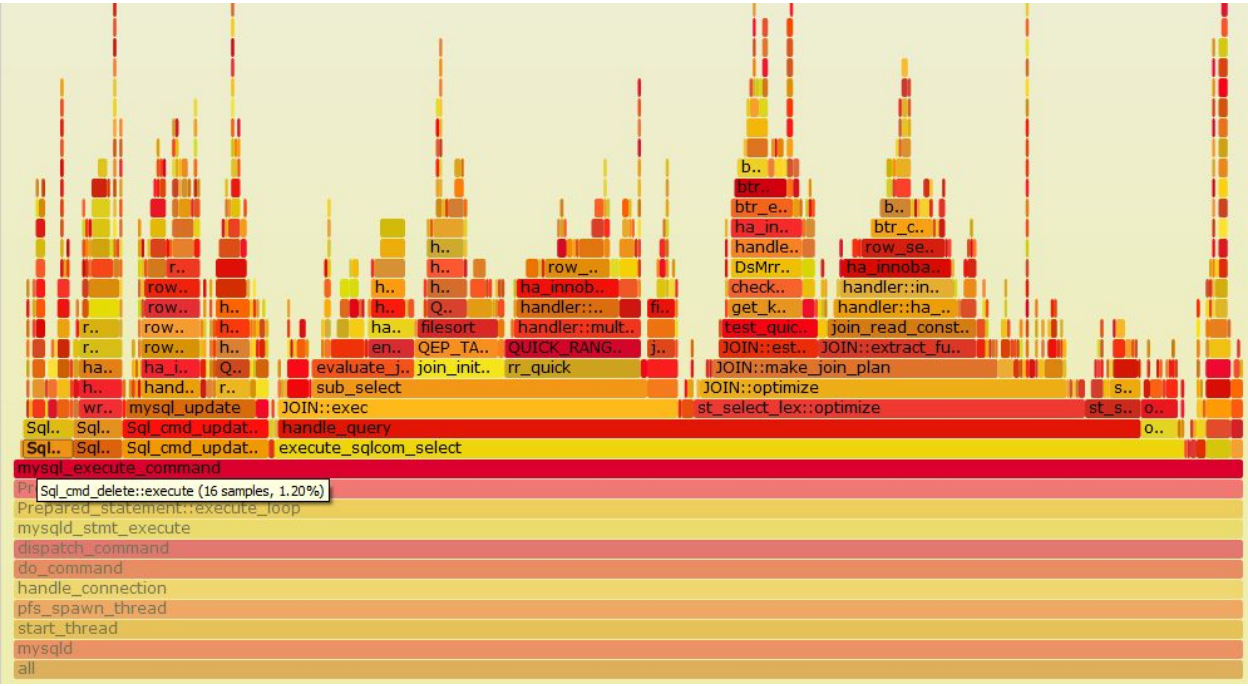
```
openxs@ao756:~/git/FlameGraph$ ls *.pl
aix-perf.pl                 stackcollapse-instruments.pl
difffolded.pl               stackcollapse-java-exceptions.pl
files.pl                    stackcollapse-jstack.pl
flamegraph.pl               stackcollapse-perf.pl
pkgsplit-perf.pl            stackcollapse.pl
range-perf.pl               stackcollapse-pmc.pl
stackcollapse-aix.pl        stackcollapse-recursive.pl
stackcollapse-bpftrace.pl   stackcollapse-stap.pl
stackcollapse-elfutils.pl   stackcollapse-vsprof.pl
stackcollapse-gdb.pl        stackcollapse-vtune.pl
stackcollapse-go.pl
```

- USAGE notes and sample command lines are presented in **.pl** files as comments

CPU Flame Graph - simple example

- Created based on these steps (while **sysbench oltp_read_write** was running):

```
openxs@ao756:~/git/FlameGraph$ sudo perf record -F 99 -a -g -- sleep 20
openxs@ao756:~/git/FlameGraph$ perf script | ./stackcollapse-perf.pl > /tmp/perf-folded.out
openxs@ao756:~/git/FlameGraph$ ./flamegraph.pl --width=1000 /tmp/perf-folded.out > /tmp/mysqld_sysbench_read_write.svg
```



Custom CPU Flame Graph - hot mutex waits

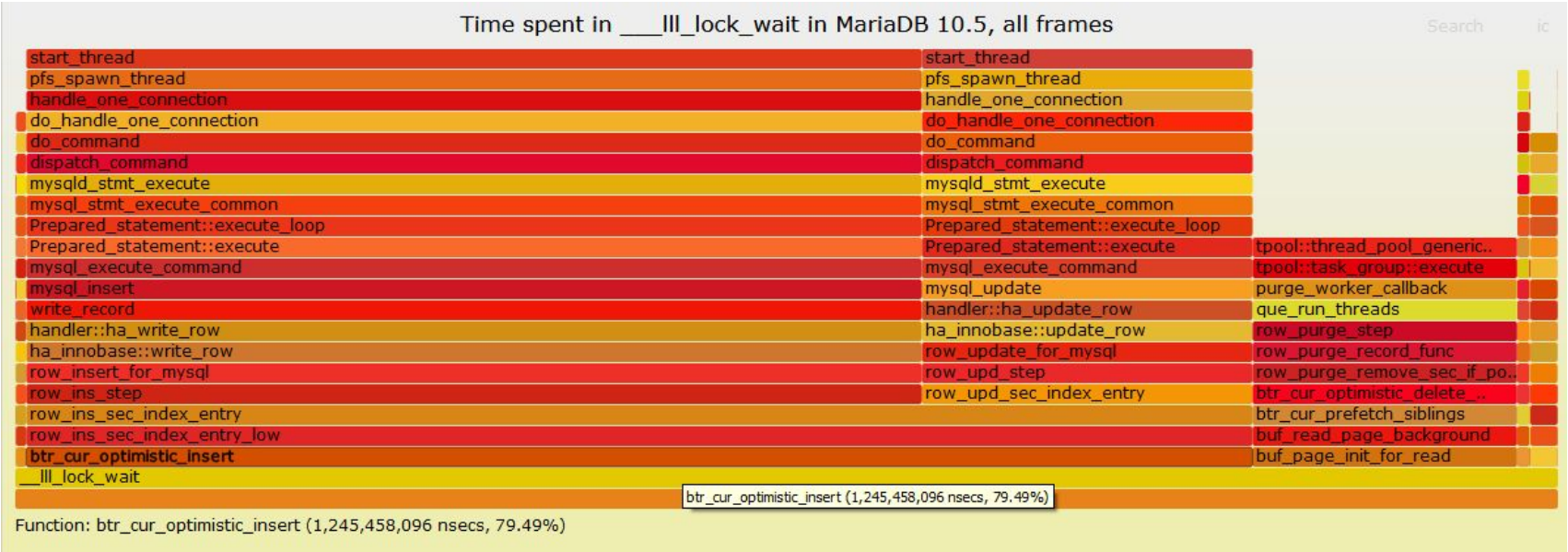
- In some cases you may want to collapse stacks yourself. Check this blog post for the details, but the idea was get “clean” frames from **bpftrace** (no address, arguments etc), for better summarizing, and remove “garbage” output:

```
[openxs@fc31 ~]$ time sudo ./lll_lock_wait2.bt 60 2>/dev/null | awk '
BEGIN { s = ""; }
/^@futexstack\[\]/ { s = ""; }
/^@futexstack/ { s = ""; }
/^\t/ { if (index($2, "(") > 0) {targ = substr($2, 1, index($2, "(") - 1)}
else {targ = substr($2, 1, index($2, "+") - 1)} ; if (s != "") { s = s ";"
targ } else { s = targ } }
/^]/ { print $2, s }
' > /tmp/collapsed_lll_lock_v2_raw.txt

[openxs@fc31 ~]$ cat /tmp/collapsed_lll_lock_v2_raw.txt | awk '{ if
(length($2) > 0) {print $2, $1} }' |
/mnt/home/openxs/git/FlameGraph/flamegraph.pl --title="Time spent in
___lll_lock_wait in MariaDB 10.5, all frames" --countname=nsecs >
~/Documents/lll_lock_v2_2.svg
```

Flame Graphs - what paths lead to mutex waits

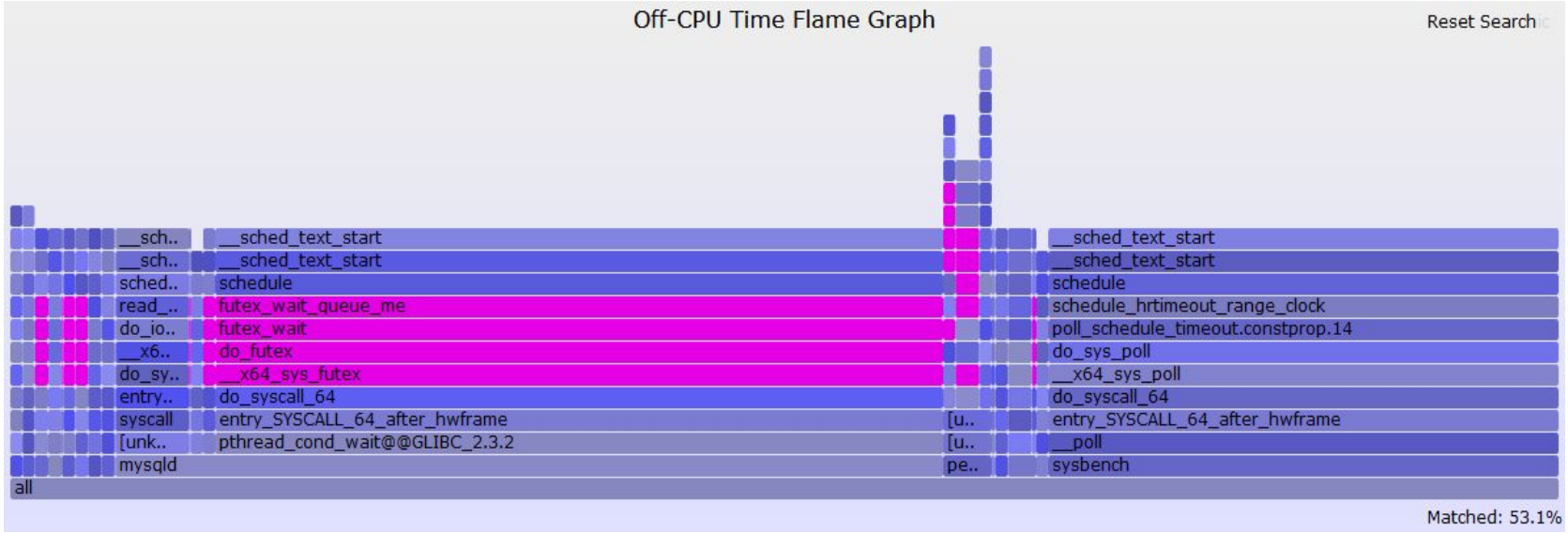
- We ended up with the following result for the **sysbench oltp_read_write** test running inserts into 5 tables from 32 threads on 4 cores for 6:



Off-CPU Flame Graph - simple example

- Created based on these steps (while **oltp_update_index.lua** was running):

```
[openxs@fc29 FlameGraph]$ sudo /usr/share/bcc/tools/offcputime -df 60 > /tmp/out.stacks
WARNING: 459 stack traces lost and could not be displayed.
[openxs@fc29 FlameGraph]$ ./flamegraph.pl --color=io --title="Off-CPU Time Flame Graph" --countname=us < /tmp/out.stacks > ~/Documents/out.svg
```

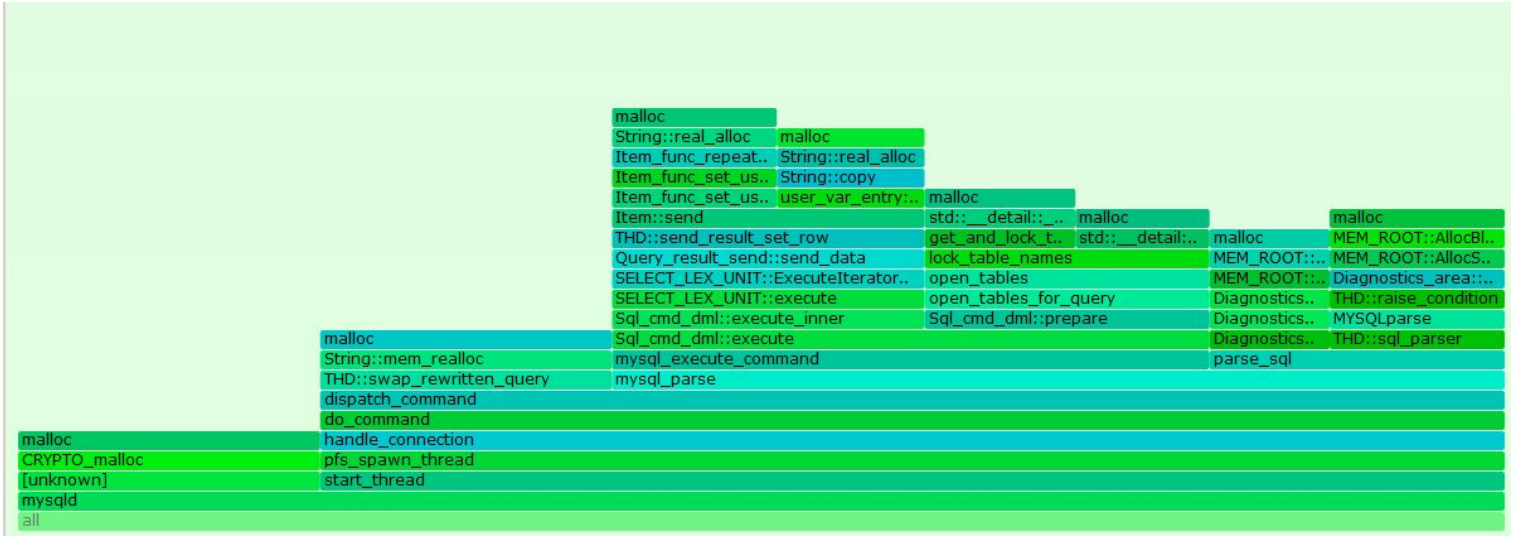


- I've searched for "futex" and related frames are highlighted

Memory Flame Graph - simple example

- Created based on **malloc()** calls tracing with **perf**, not the best ideea. See this blog post for more details:

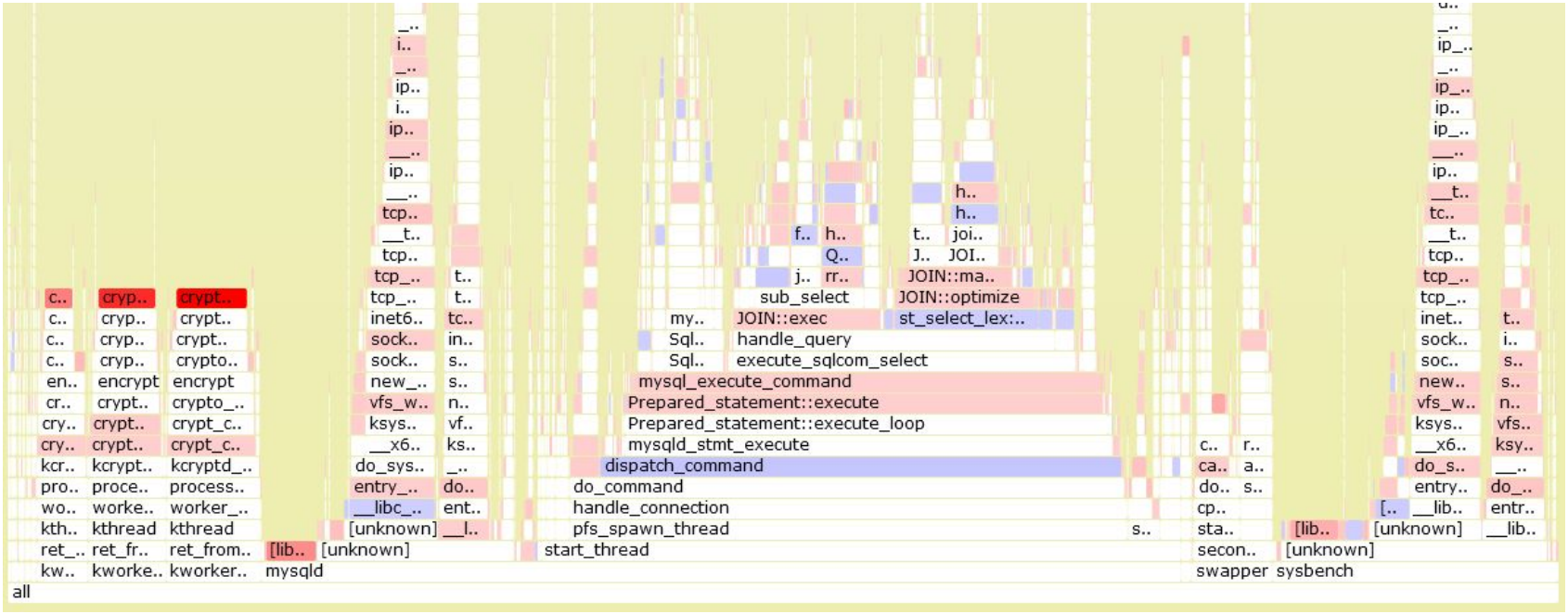
```
openxs@ao756:~$ sudo perf probe -x /lib/x86_64-linux-gnu/libc.so.6 'malloc size=%di:s64'
openxs@ao756:~$ sudo perf record -e 'probe_libc:malloc' -aRg sleep 10
openxs@ao756:~$ sudo perf script > out.stack
openxs@ao756:~$ git/FlameGraph/stackcollapse-perf.pl < out.stack | git/FlameGraph/flamegraph.pl --color=mem --title='malloc( Fale Graph' --countname="calls" -- > malloc.svg
```



Differential Flame Graph - simple CPU example

- Check this page for more details and types of them
- I've tried the same test as for CPU graph, but with 12 threads instead of 4:

```
openxs@ao756:~/git/FlameGraph$ sudo perf script | ./stackcollapse-perf.pl
> /tmp/perf-folded2.out
openxs@ao756:~/git/FlameGraph$ ./difffolded.pl /tmp/perf-folded.out
/tmp/perf-folded2.out | ./flamegraph.pl > /tmp/diff.svg
```



Flame Graphs - more examples, Q&A

- MySQL bug reports based on flame graphs (**Mark Callaghan**):
  - **Bug #102238** - "**log_writer uses too much CPU on small servers**". 8.0.22
  - **Bug #102037** - "**CPU overhead from inlists much larger in 8.0.22**".
- MariaDB bug reports based on flame graphs:
  - **MDEV-23475** - "**InnoDB performance regression for write-heavy workloads**"
  - **MDEV-19399** - "**do not call slow my_timer_init() several times**"
  - Google for **site:jira.mariadb.org flame graph**
- See also (from my collection):
  - https://www.percona.com/blog/2019/11/20/profiling-software-using-perf-and-flame-graphs/
  - https://www.percona.com/blog/2020/01/15/using-flame-graphs-to-process-outputs-from-pt-pmp
  - https://github.com/pingcap/tidb/pull/12986 - PR for TiDB (PingCap)
  - https://randomascii.wordpress.com/2013/03/26/summarizing-xperf-cpu-usage-with-flame-graphs/ - WPA/Windows
- **Questions and Answers?**