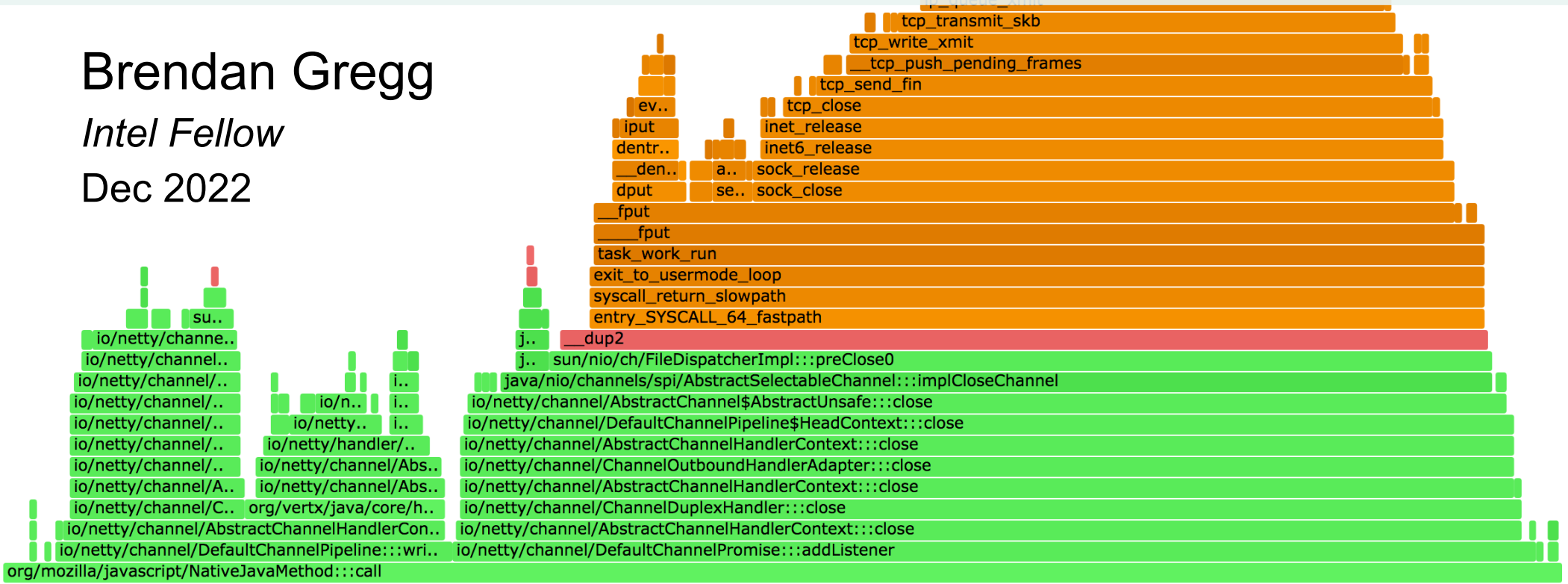
YOW! 2022

Visualizing Performance
The Developer's Guide to **Flame Graphs**

Brendan Gregg

Intel Fellow

Dec 2022

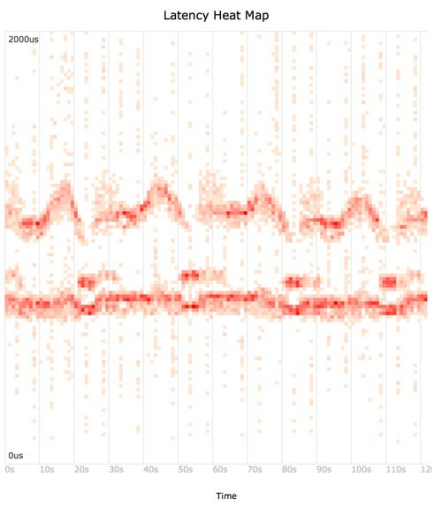


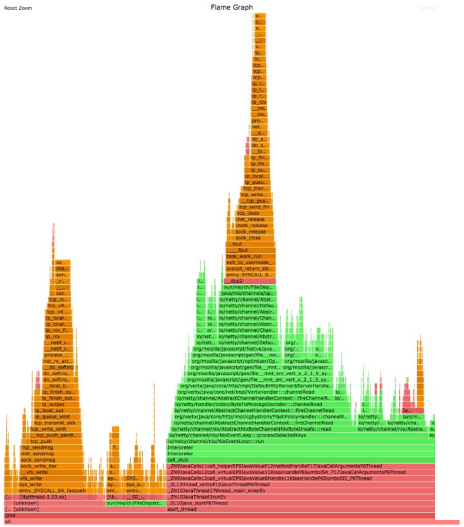
Statement from the heart

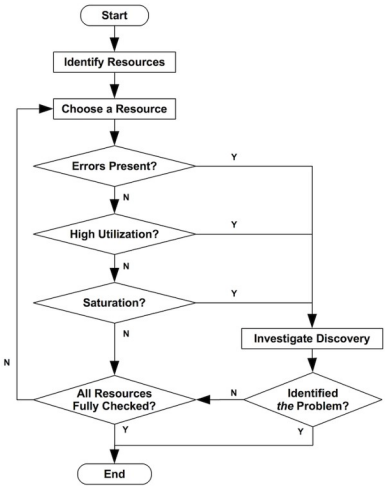
I’d like to begin by acknowledging the Traditional Owners of this land and pay my respects to Elders past and present.

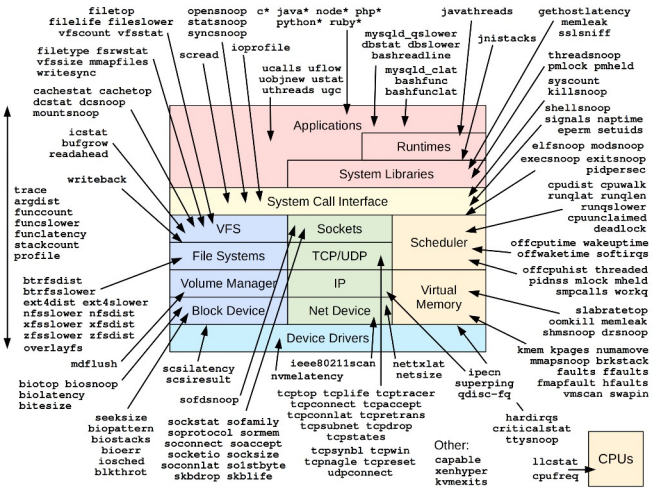
My Dream

To Completely Understand
the Performance of Everything







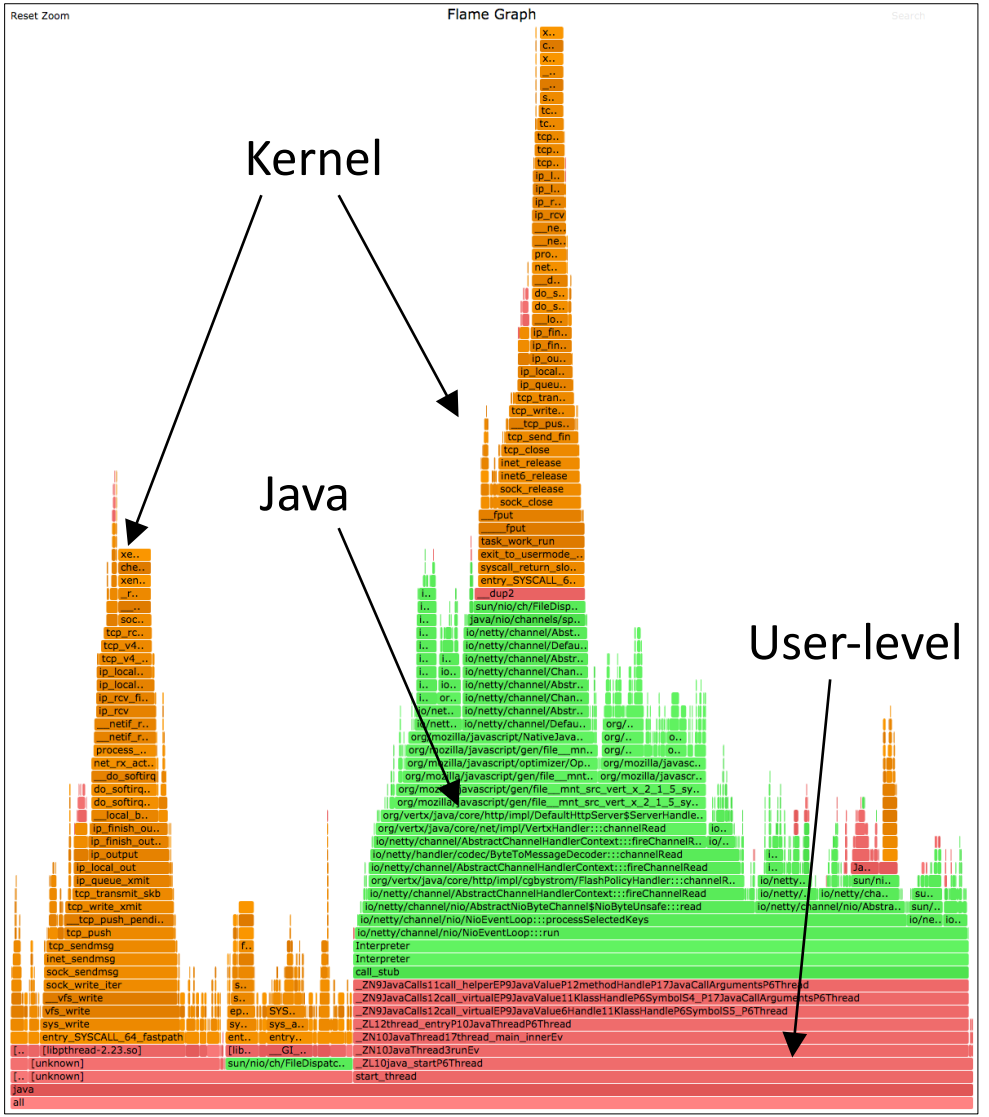


Flame Graphs

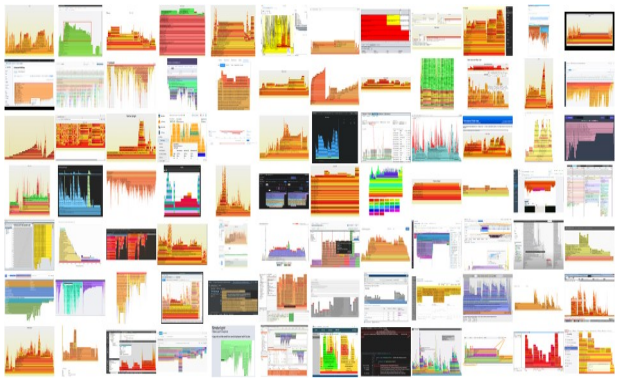
A visualization of software

Can also visualize CPU and other resource usage

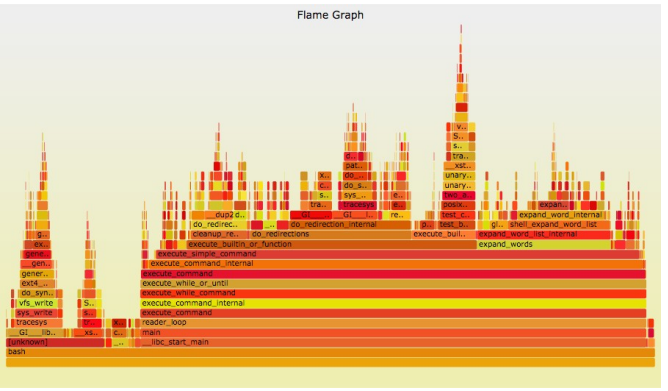
Now a staple in performance engineering



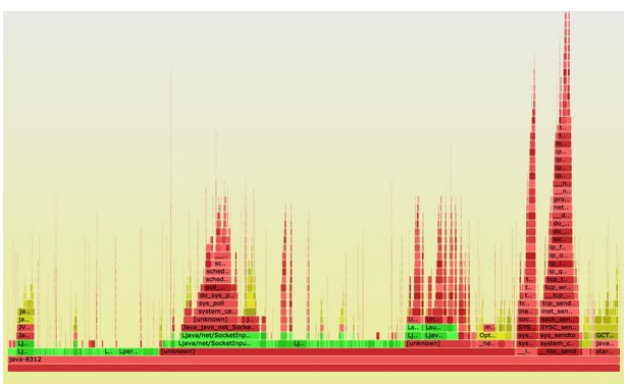
Agenda



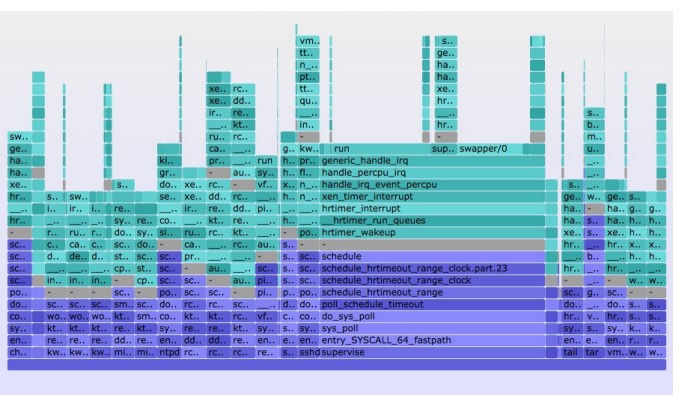
1. Implementations



2. CPU Flame graphs



3. Stacks & Symbols



4. Advanced flame graphs

Take Aways

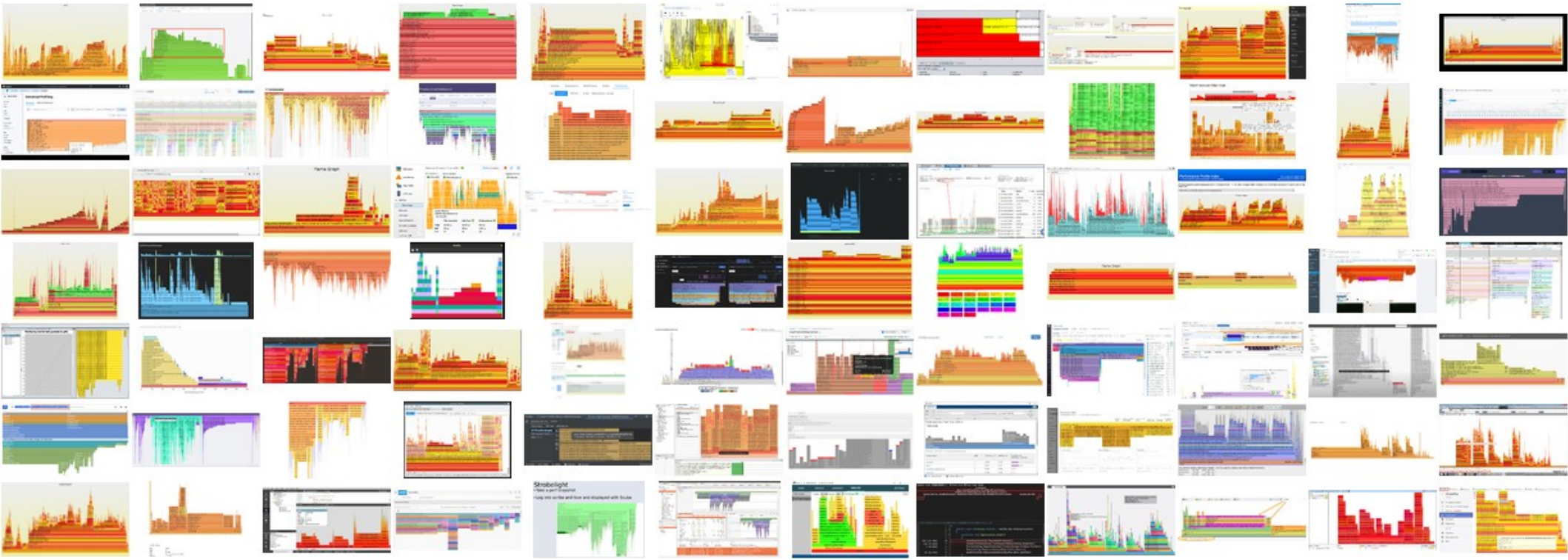
1. Interpret CPU flame graphs
2. Understand runtime challenges
3. Why eBPF for advanced flame graphs

A new tool to lower your cost, latency, and carbon

Slides online:
https://www.brendangregg.com/Slides/YOW2022_flame_graphs.pdf

1. IMPLEMENTATIONS

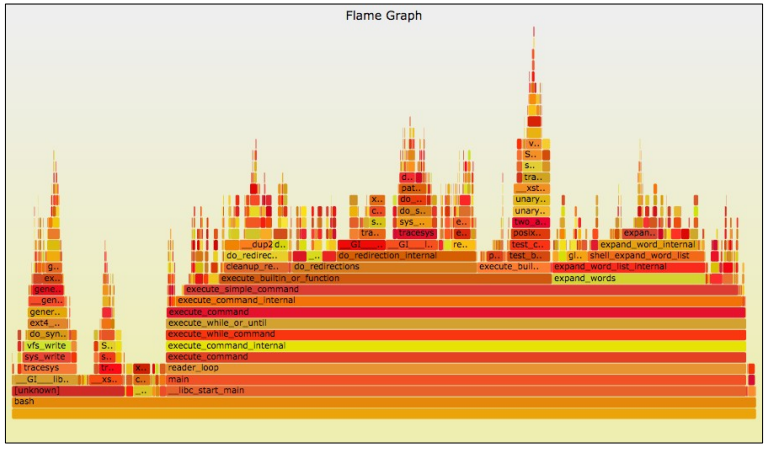
Quick Tour of Some Examples

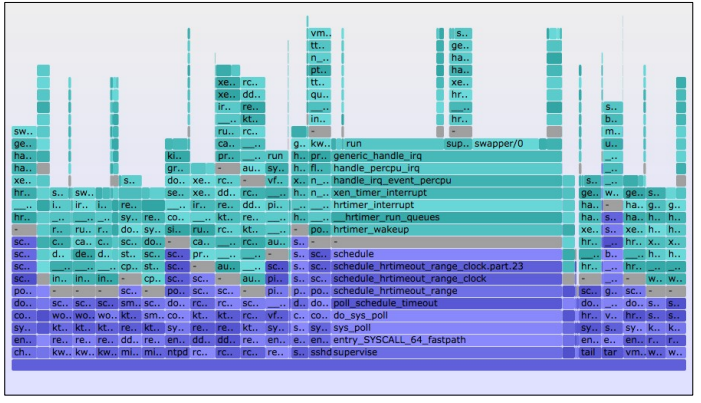


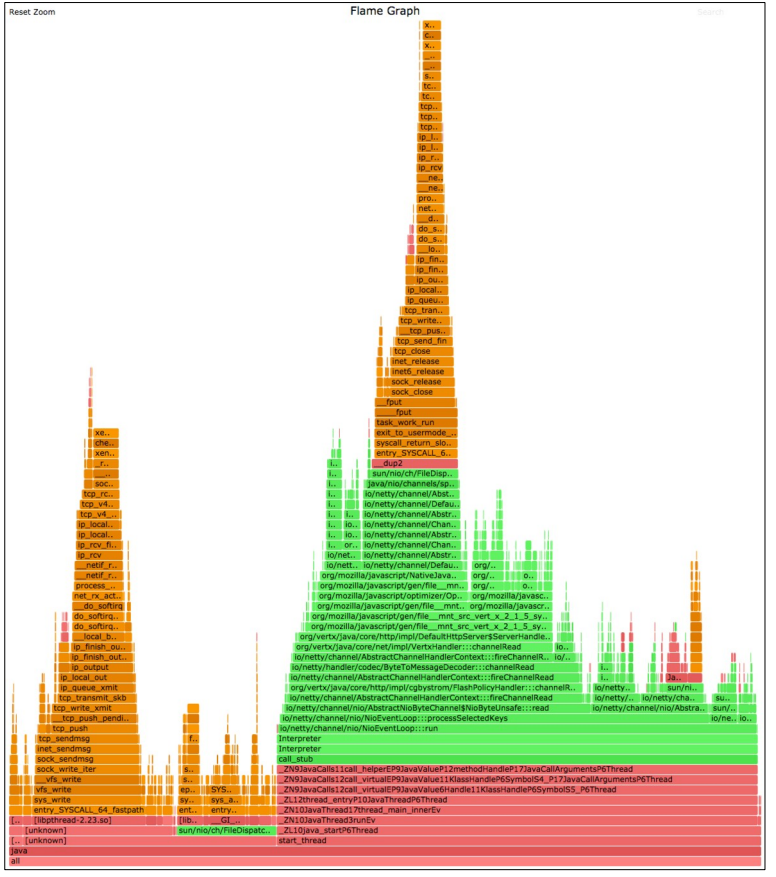
More examples in later “bonus slides” section.

(Note: This is not an an endorsement of any company/product or sponsored by anyone.)

My original flamegraph.pl (2011; using Perl/SVG/JavaScript)

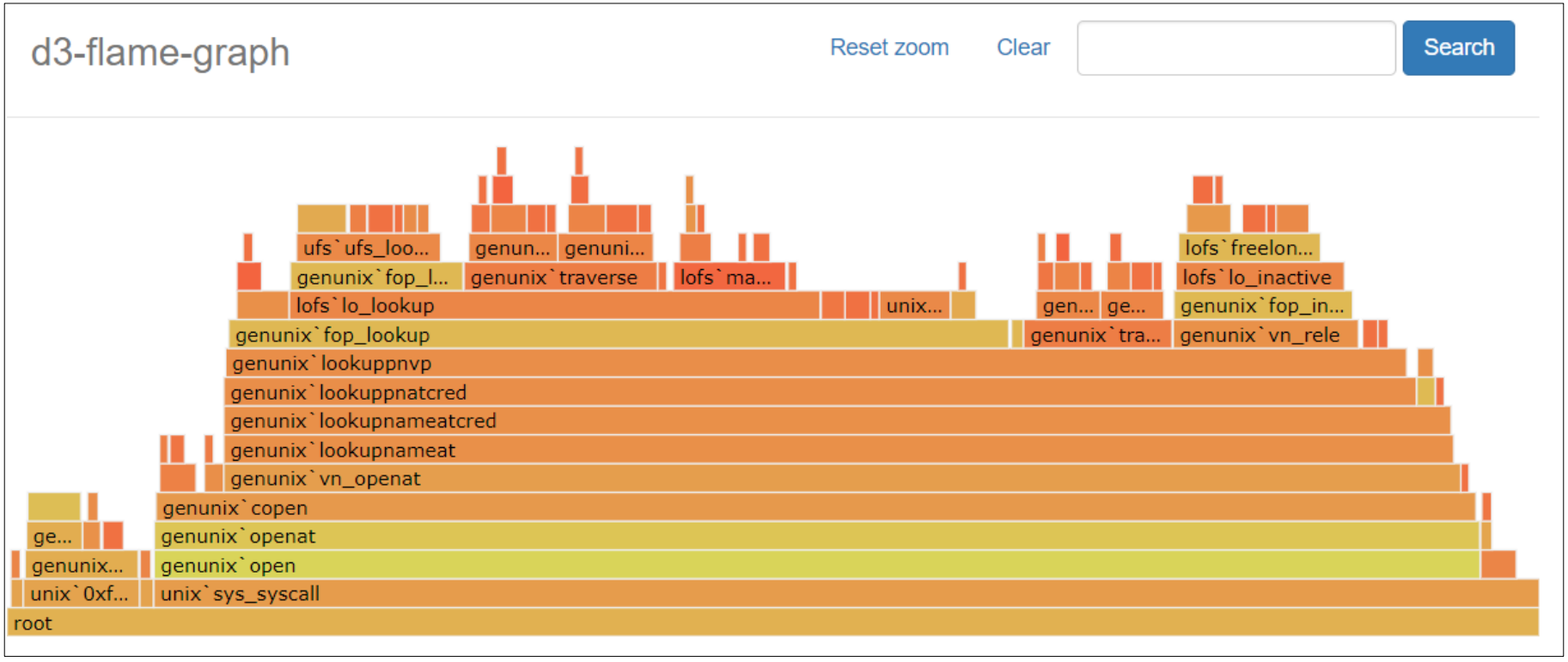






https://github.com/brendangregg/FlameGraph

Martin Spier d3-flame-graph (my colleague at Netflix; 2015; D3)

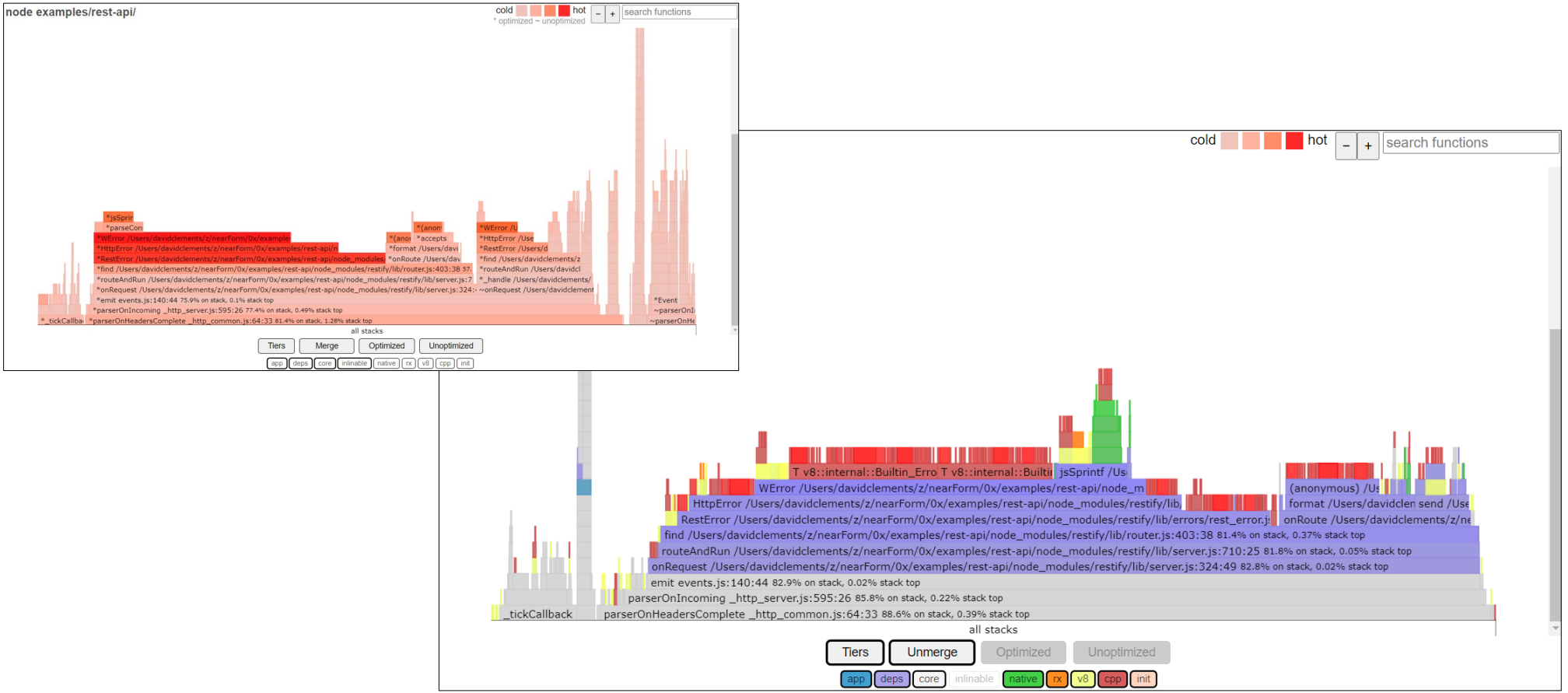


Source: https://github.com/spiermar/d3-flame-graph https://martinspier.io/

Facebook: Strobelight (2014)

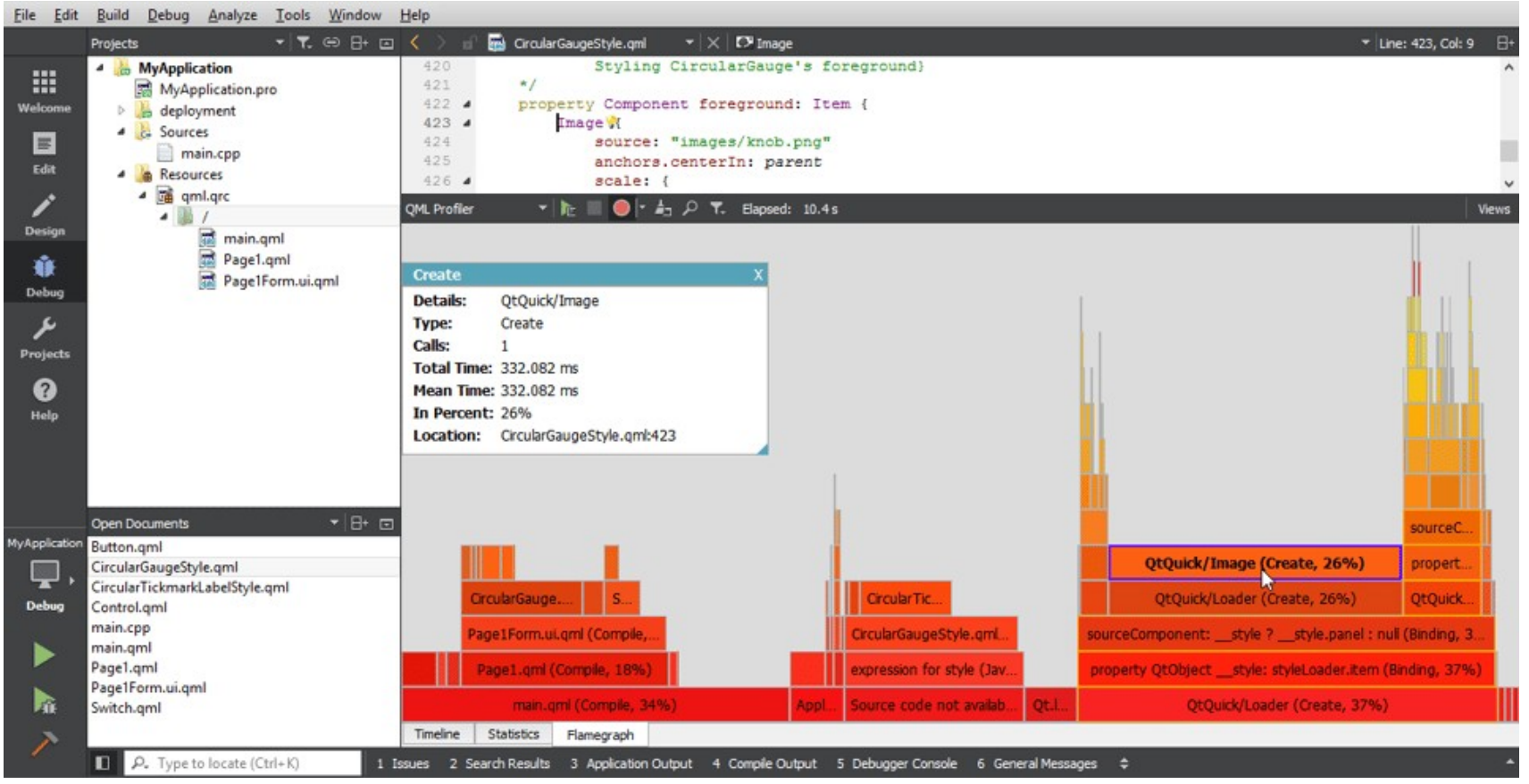
Source: https://tracingsummit.org/ts/2014/files/TracingSummit2014-Tracing-at-Facebook-Scale.pdf

Node.js: 0x (2016)



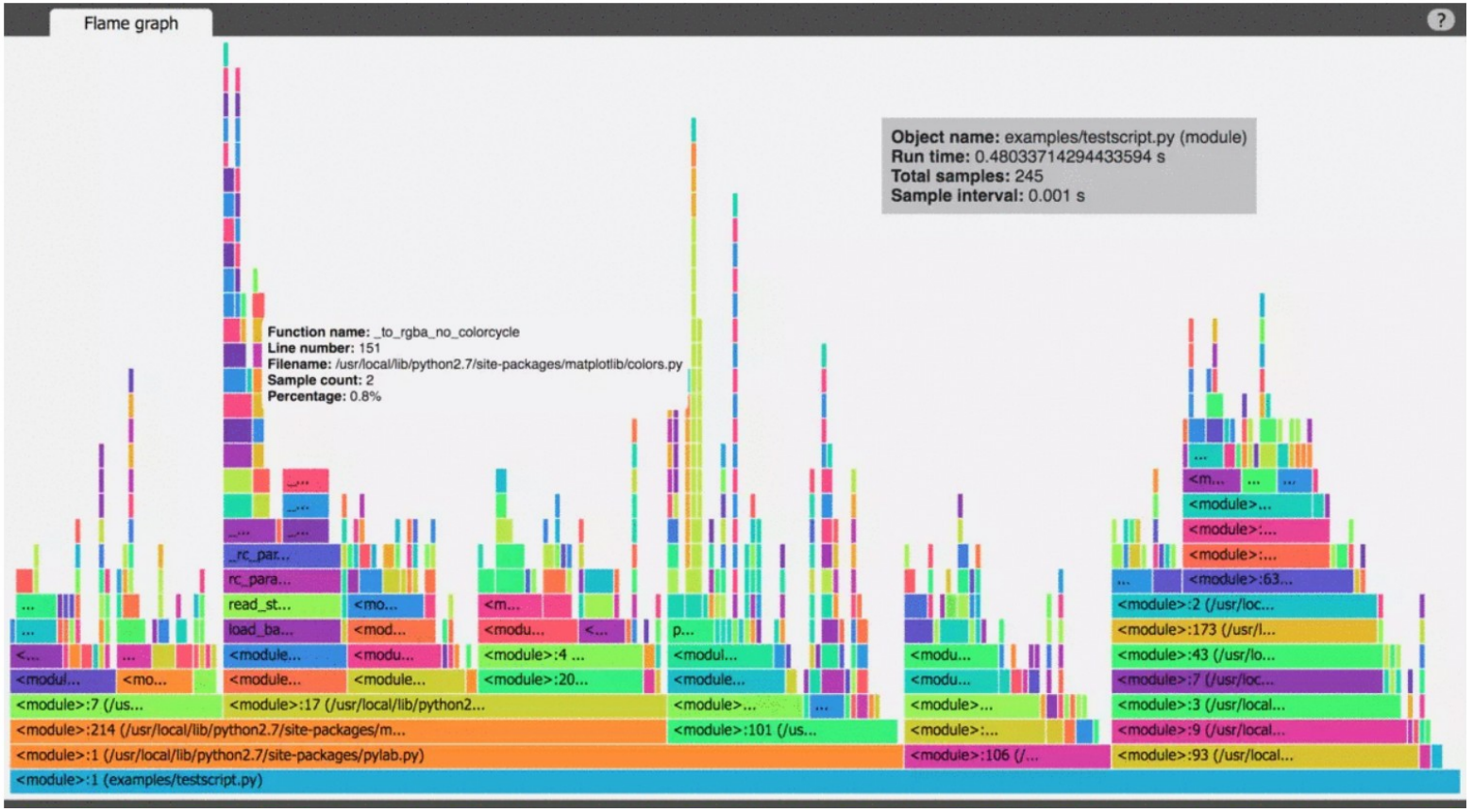
Source: https://github.com/davidmarkclements/0x (David Mark Clements)

Qt: Creator (2016)



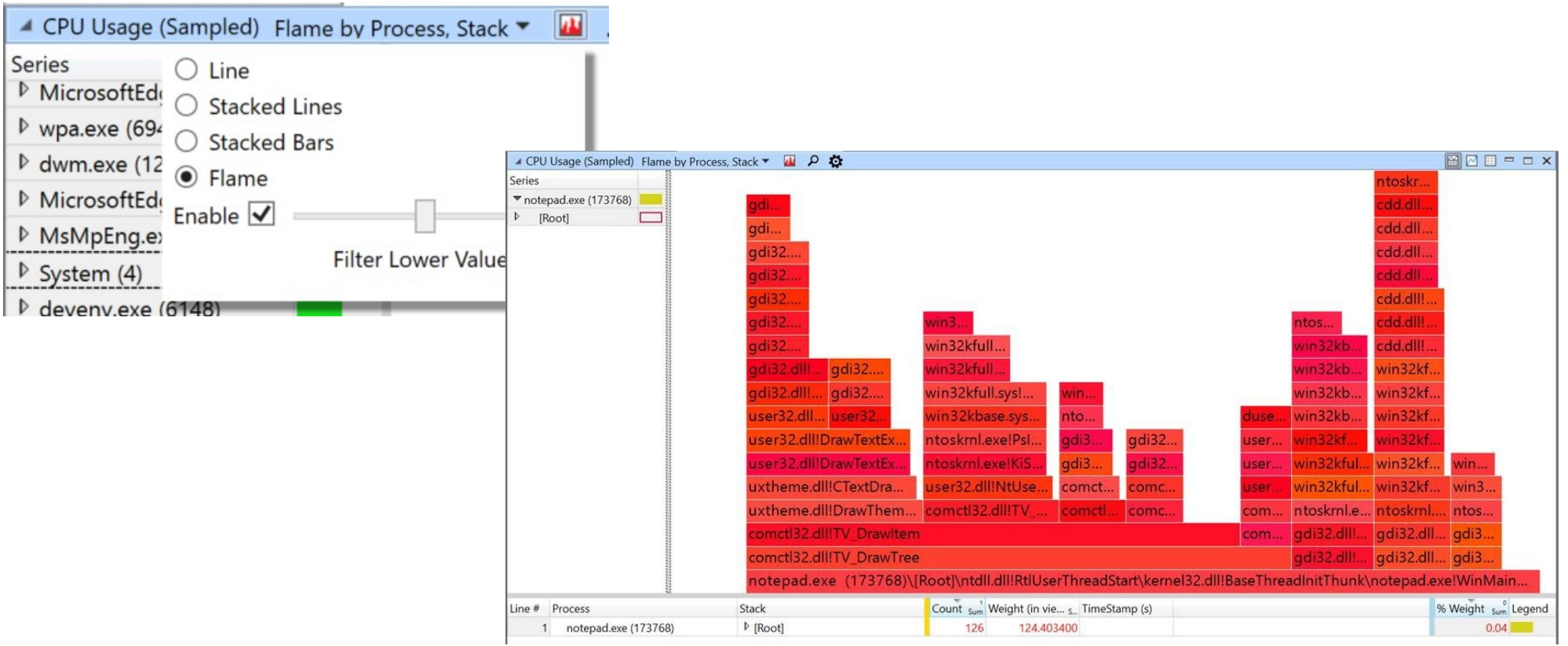
Source: https://www.qt.io/blog/2016/05/11/qt-creator-4-0-0-released

Python: vprof (2016)



Source: https://github.com/nvdv/vprof (Nick Volynets)

Microsoft: WPA / ETW (2016)

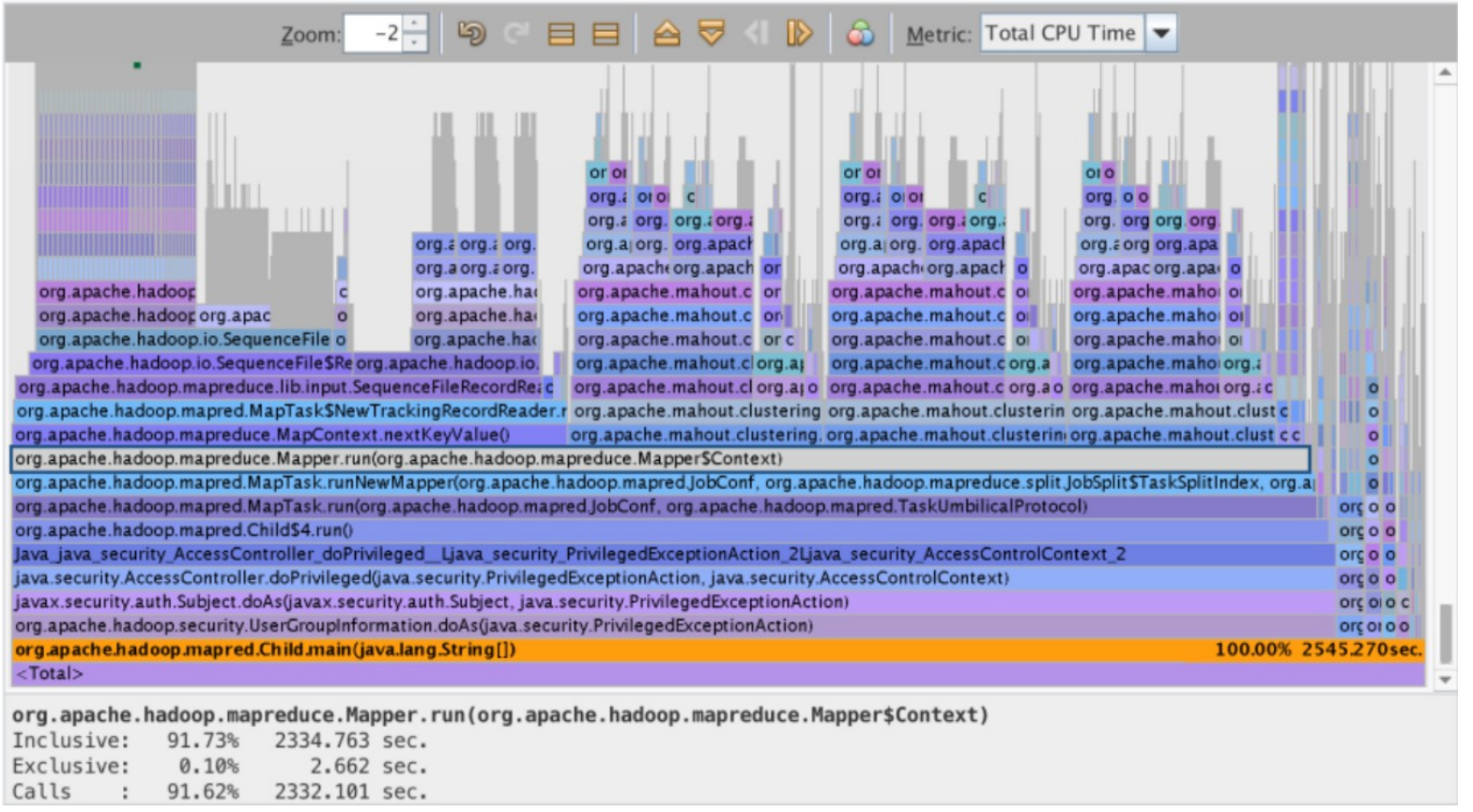


Source: https://learn.microsoft.com/en-us/windows-hardware/test/wpt/graphs#flame_graphs

LinkedIn: ODP (2017)

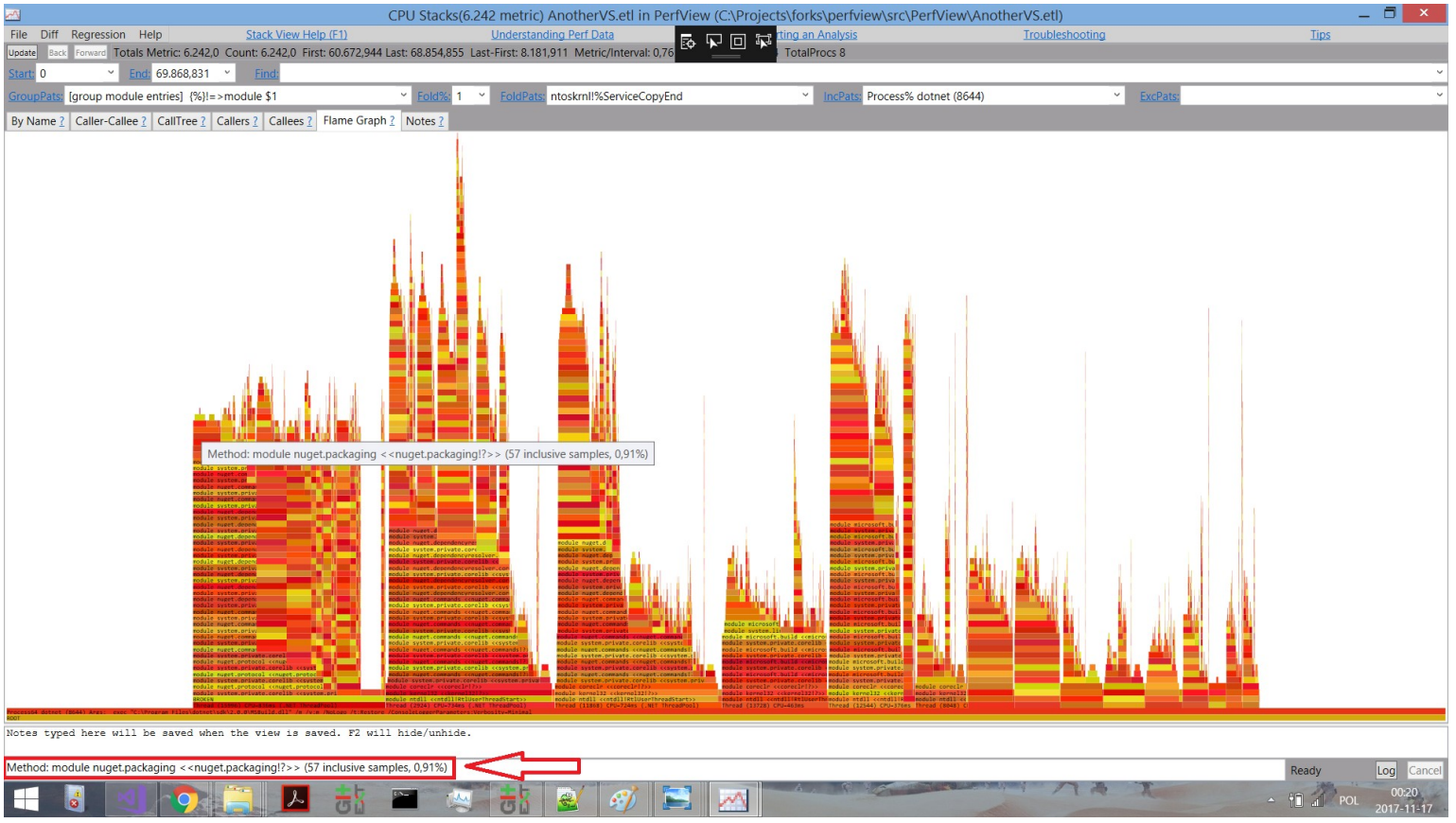
Source: https://engineering.linkedin.com/blog/2017/01/odp--an-infrastructure-for-on-demand-service-profiling

Oracle: Developer Studio Performance Analyzer (2017)



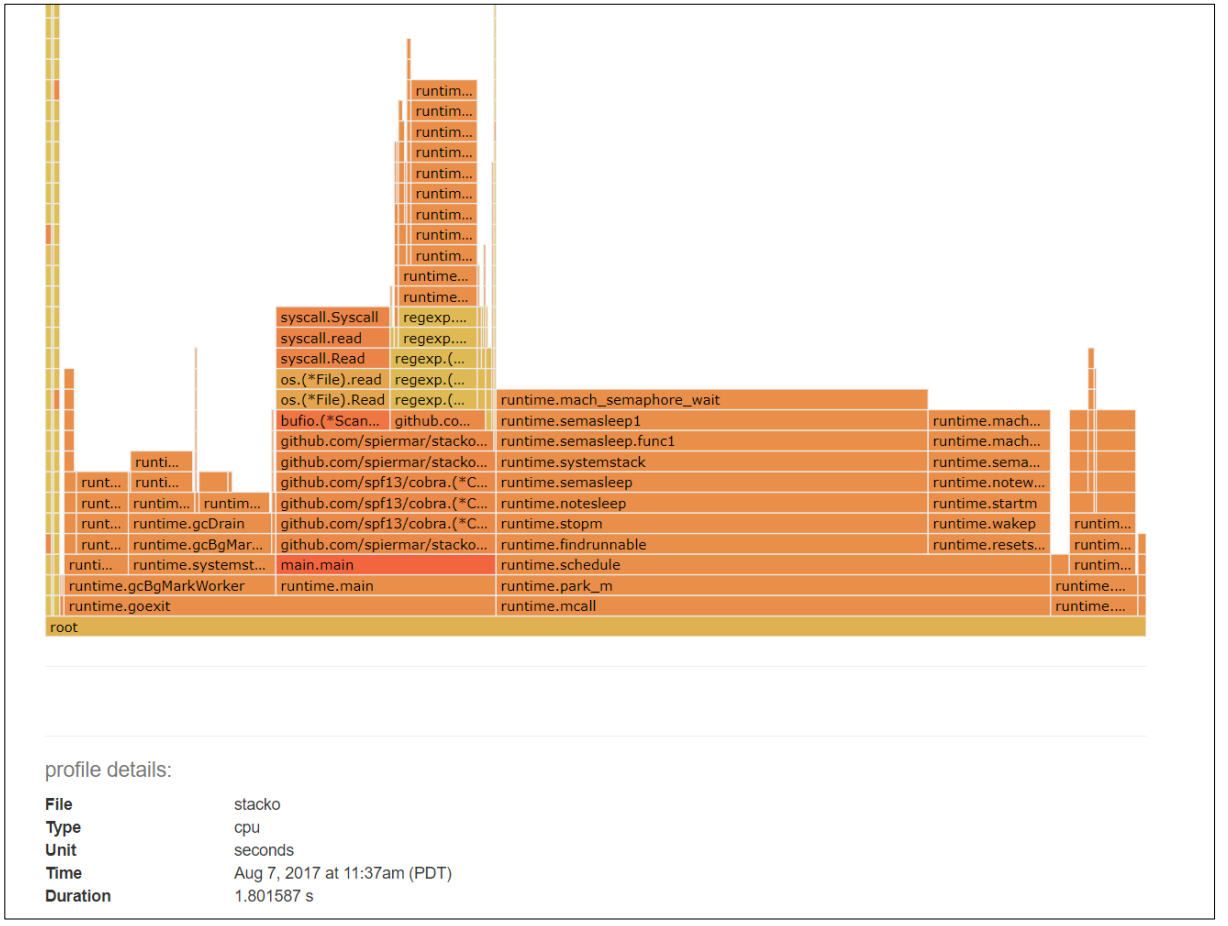
Source: https://www.oracle.com/technetwork/server-storage/solarisstudio/documentation/o11-151-perf-analyzer-brief-1405338.pdf

Windows: PerfView (2017)



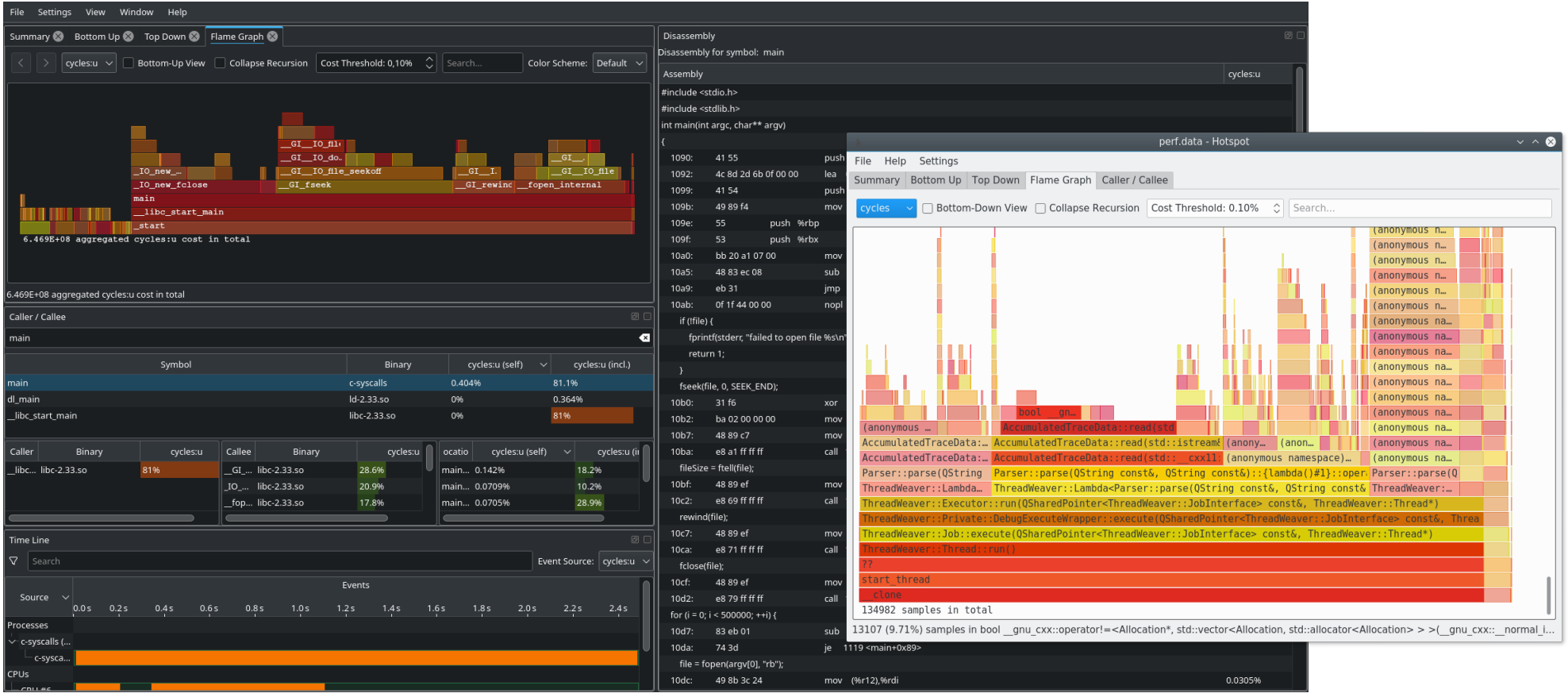
Source: https://github.com/Microsoft/perfview/pull/440 (Adam Sitnik)

Google: pprof (2017)



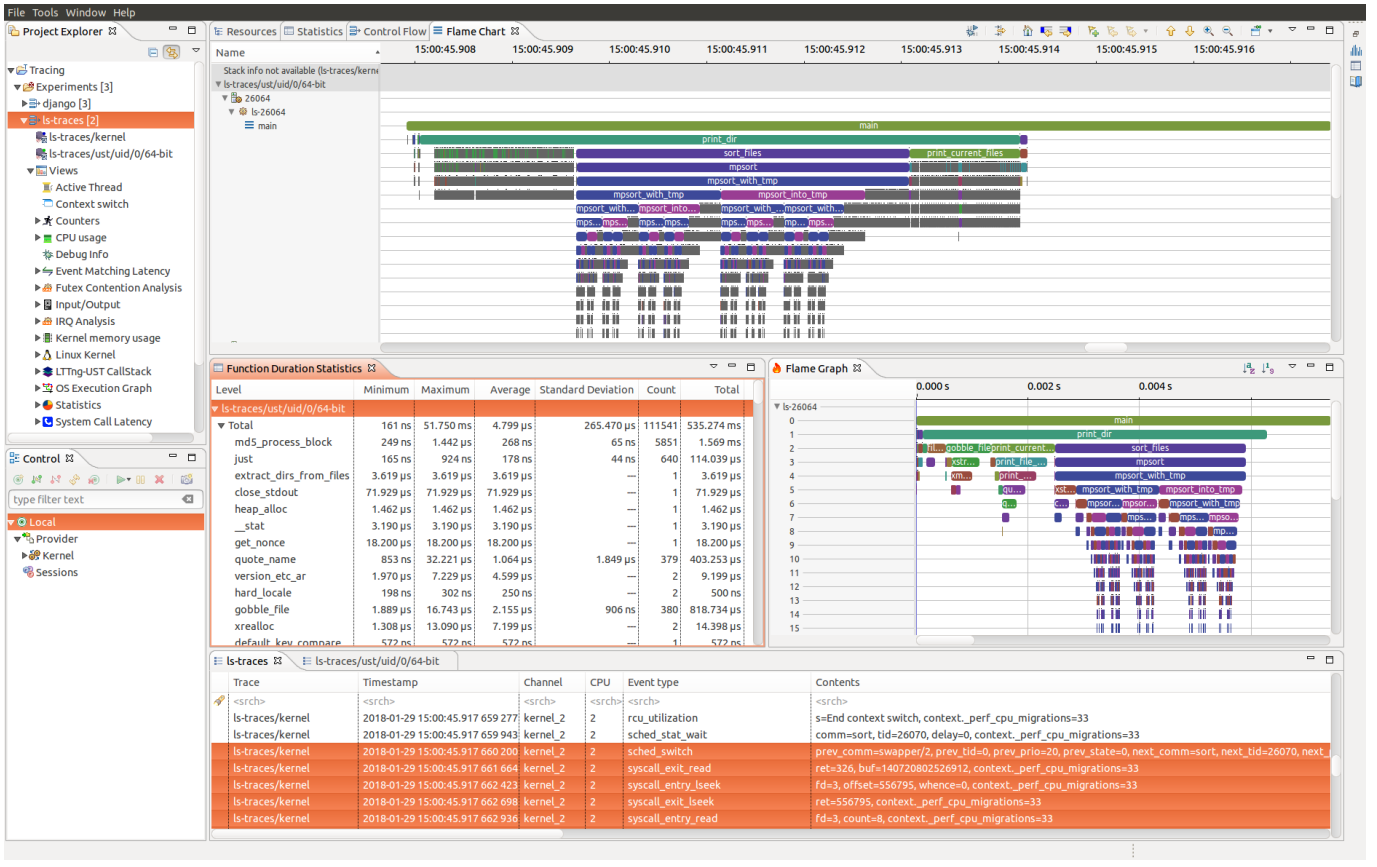
Source: https://github.com/google/pprof/pull/188 (Martin Spier)

Linux: hotspot (2017)



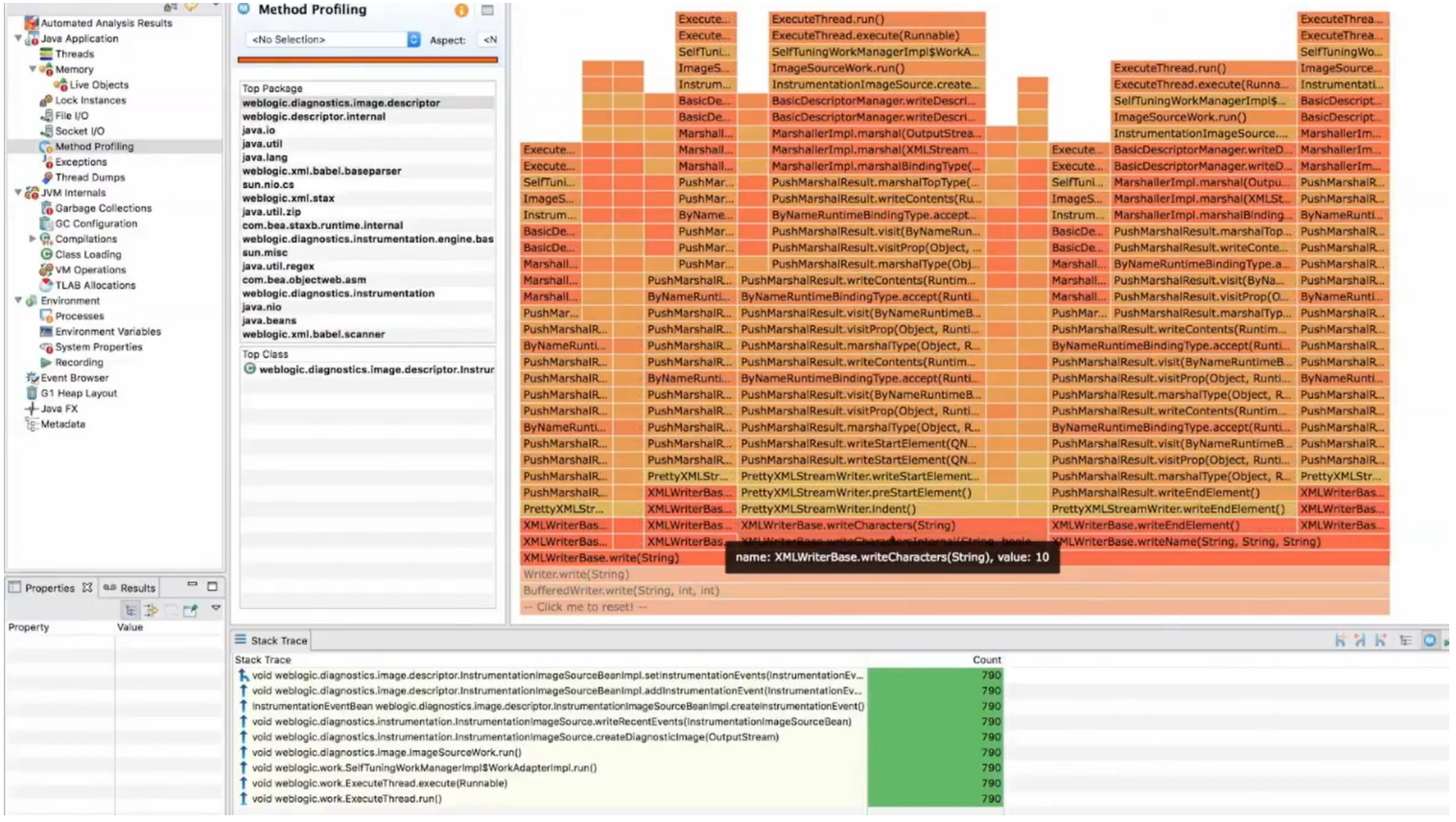
Source: https://github.com/KDAB/hotspot (Milian Wolff)

Eclipse Foundation: TraceCompass (2018)



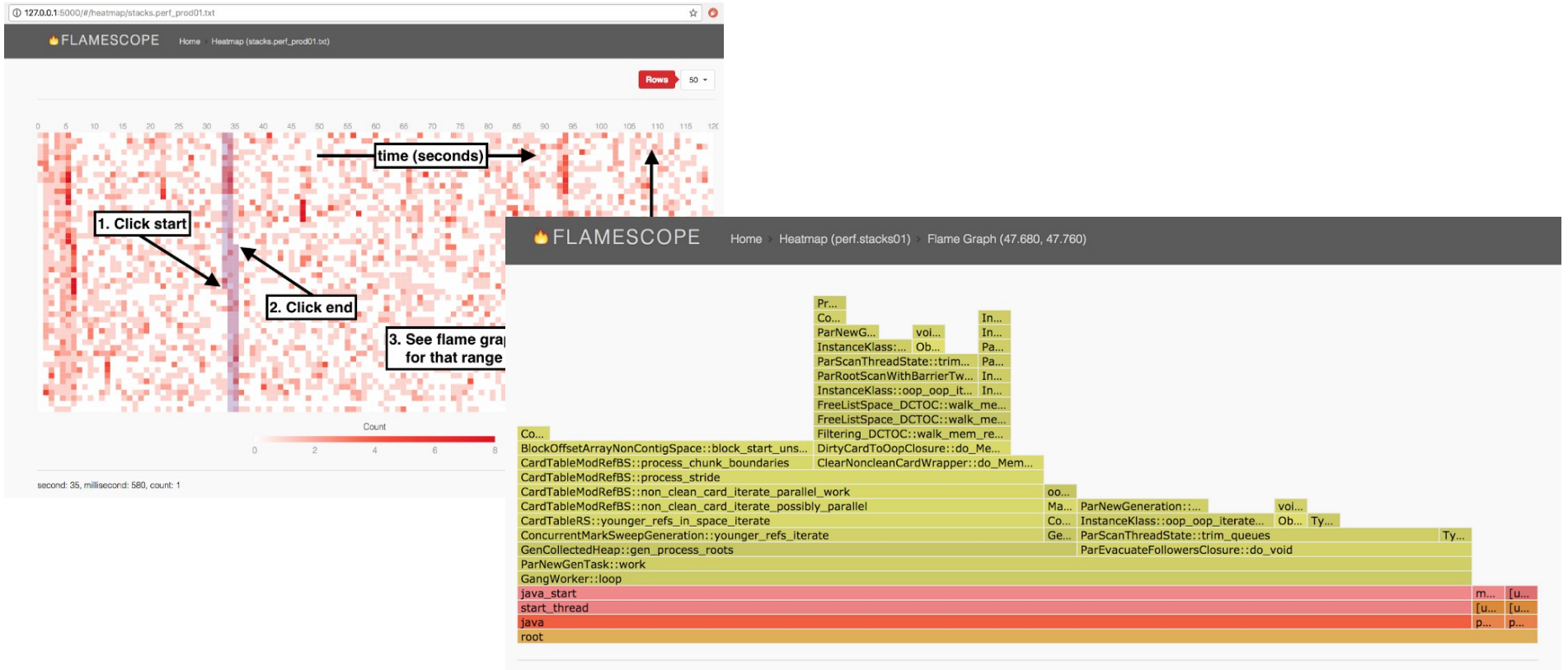
Source: https://www.eclipse.org/tracecompass/index.html

Java: Java Mission Control (2018)



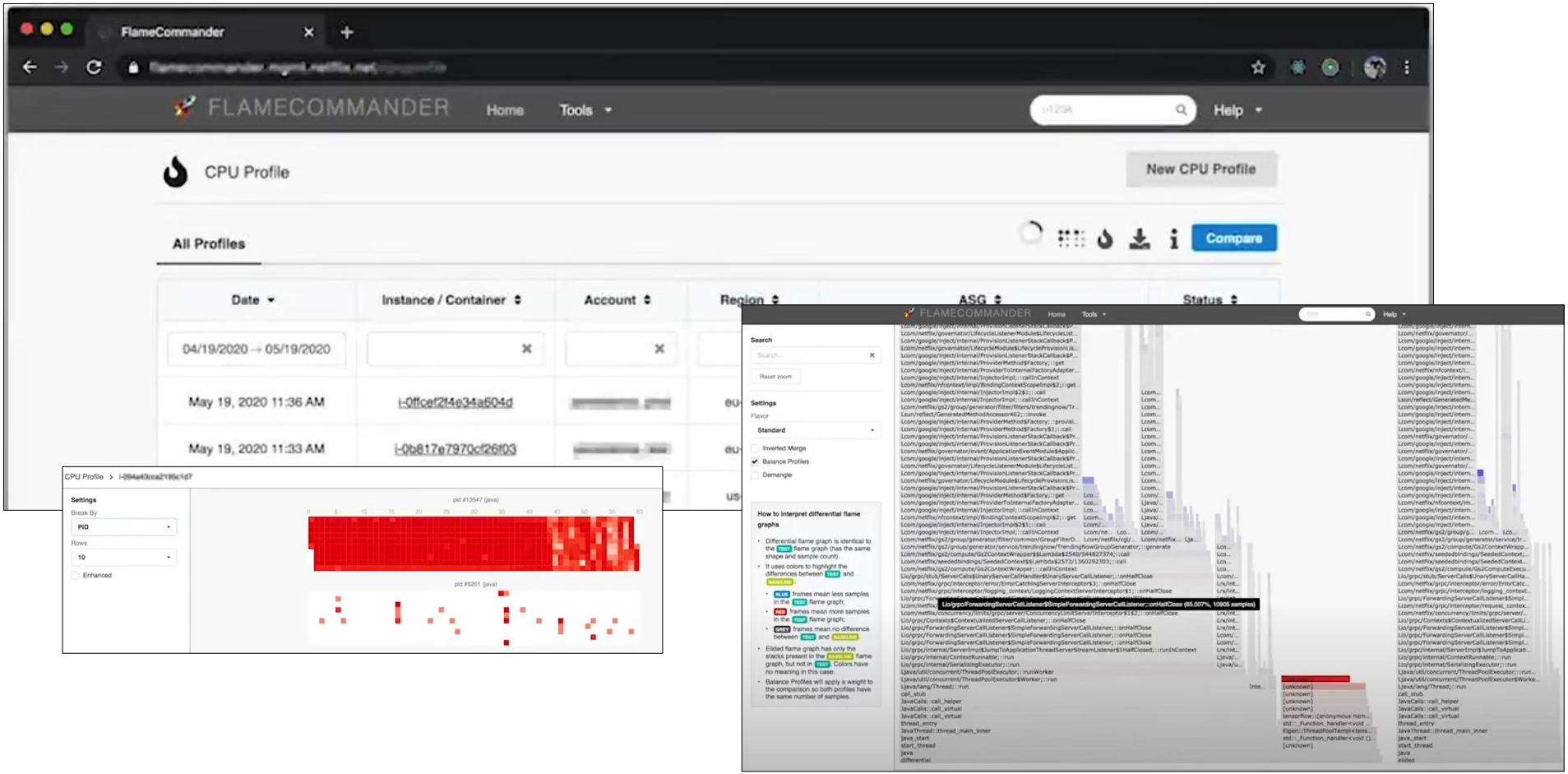
Source: https://github.com/thegreystone/jmc-flame-view (Marcus Hirt)

Netflix: FlameScope (2018)



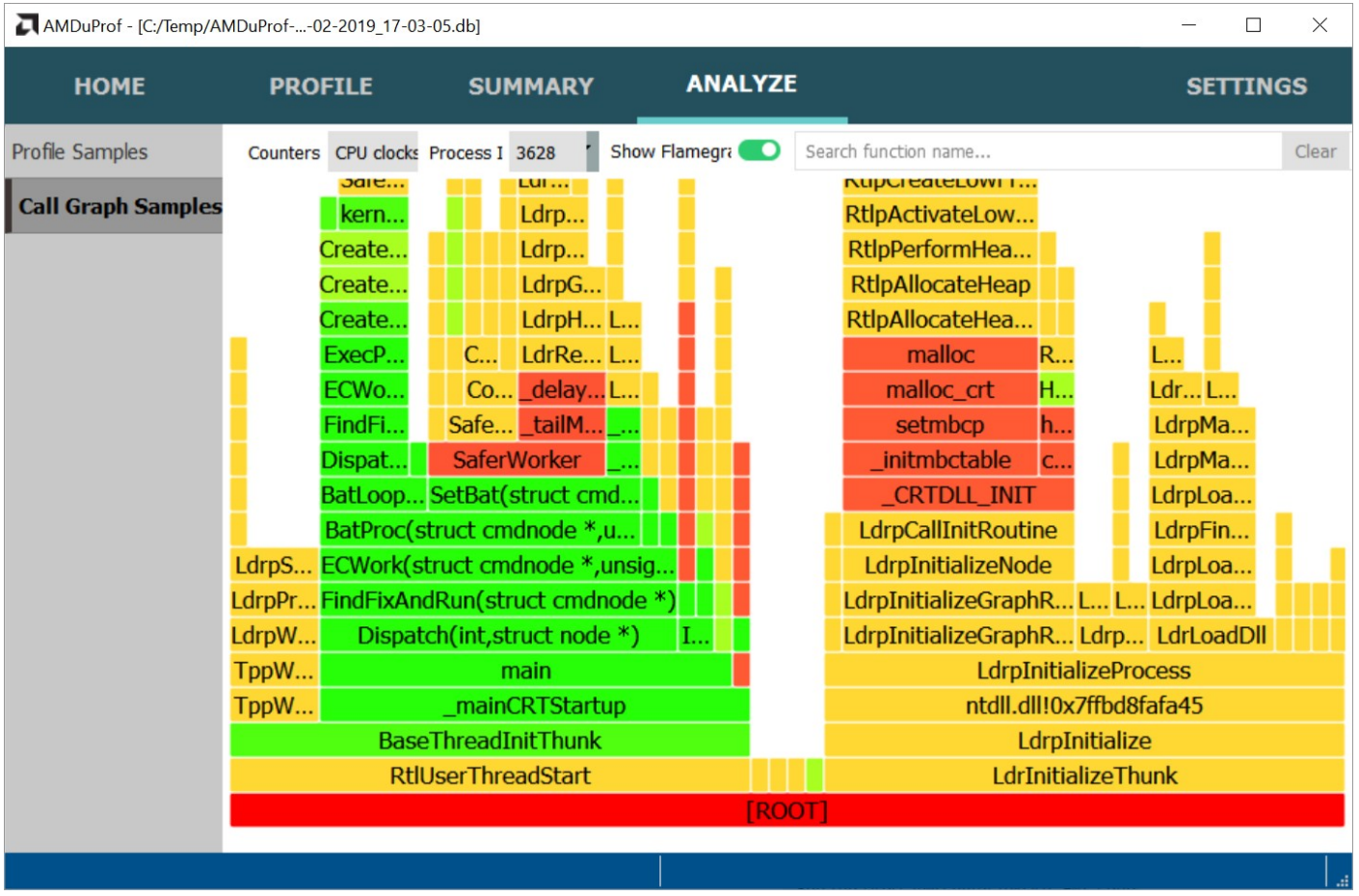
Source: https://netflixtechblog.com/netflix-flamescope-a57ca19d47bb (Brendan Gregg, Martin Spier)

Netflix: FlameCommander (2019)



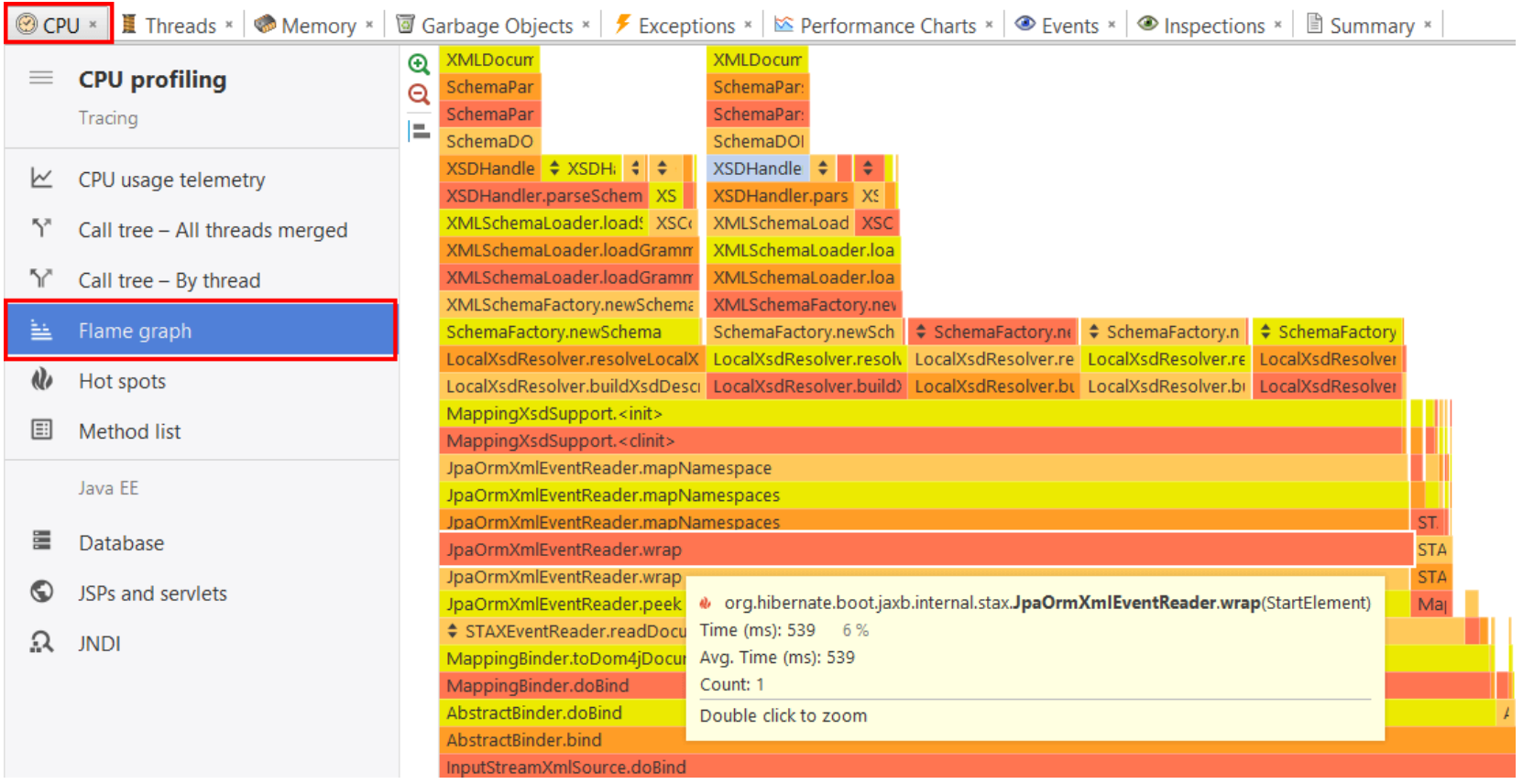
Source: https://www.youtube.com/watch?v=L58GrWcrD00 (Martin Spier, Jason Koch, Susie Xia, Brendan Gregg)

AMD: uProf (2019)



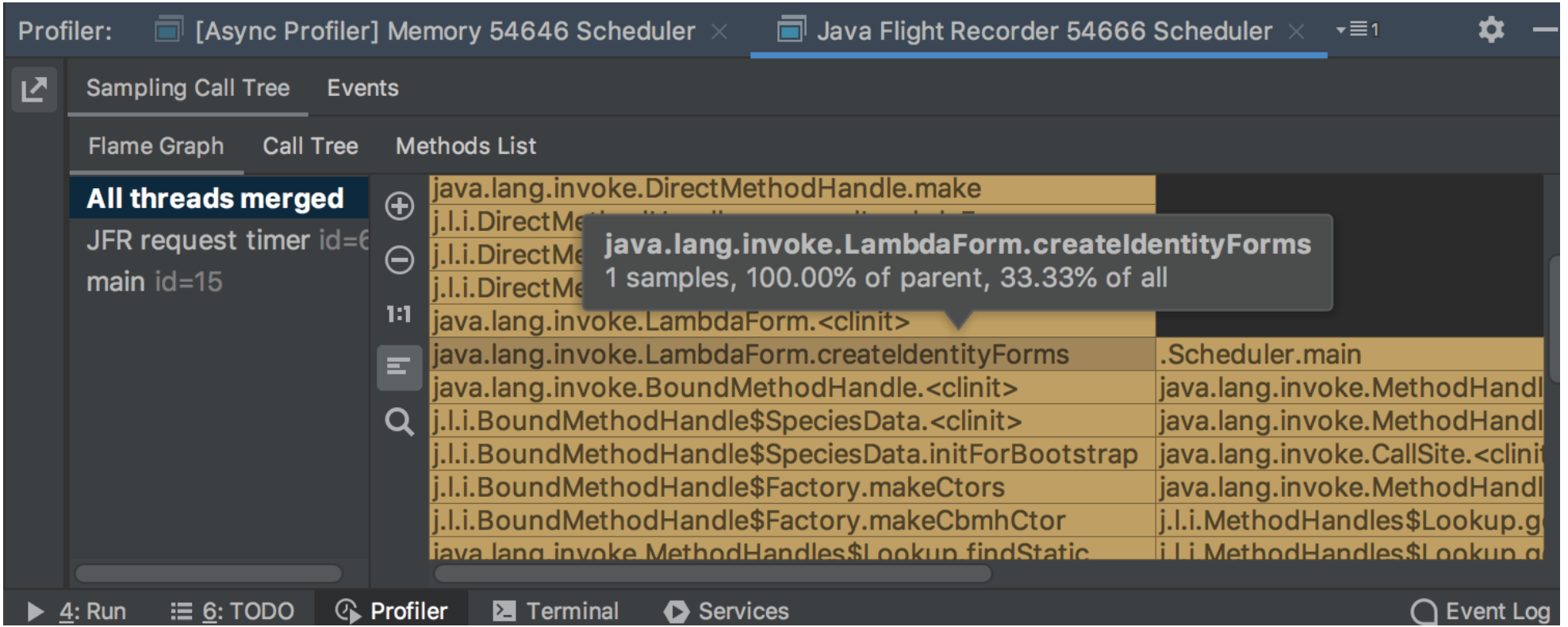
Source: https://developer.amd.com/amd-uprof/?sf215410082=1

Java: YourKit (2019)



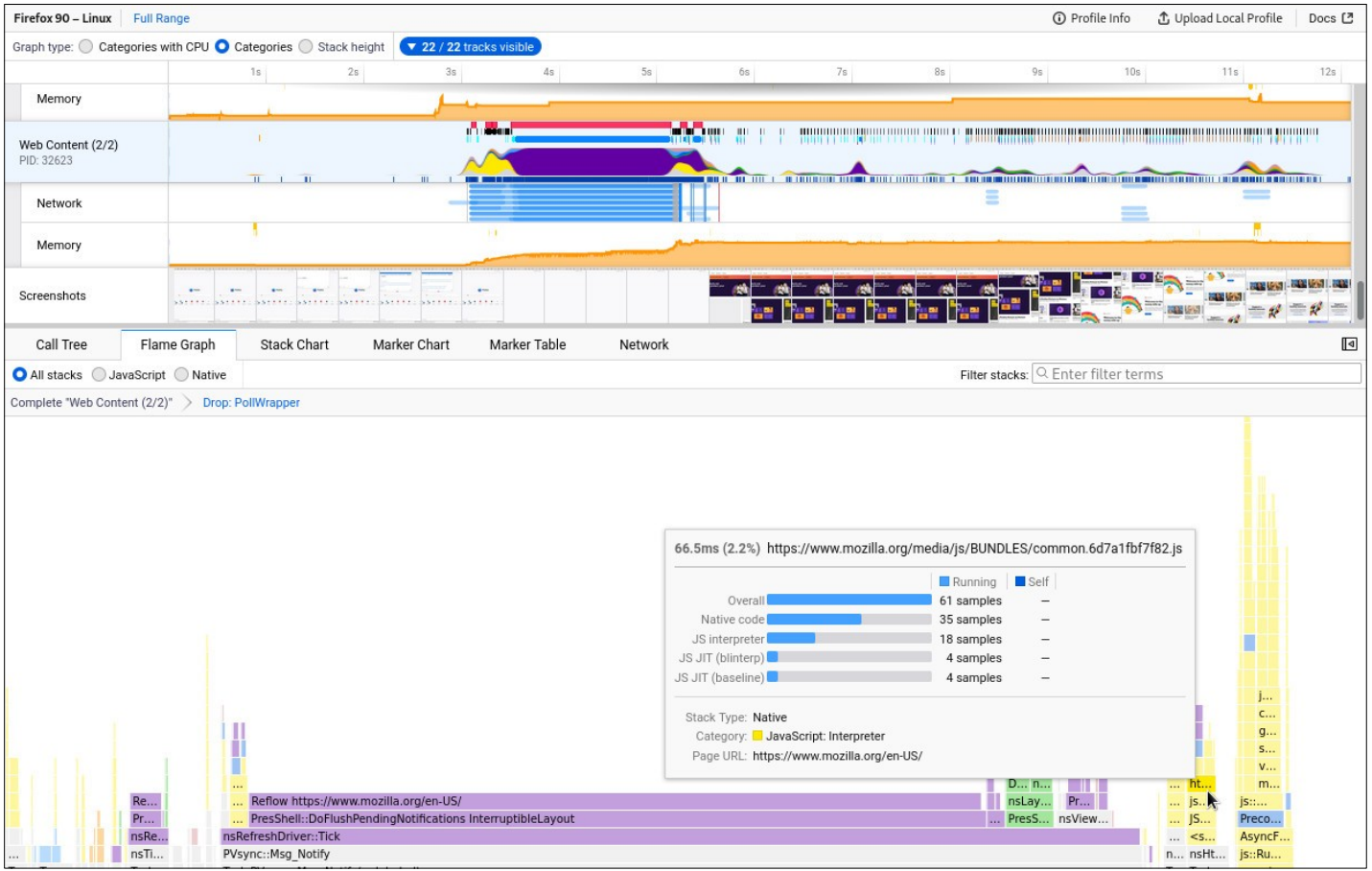
Source: https://www.yourkit.com/docs/java/help/cpu_flame_graph.jsp

Java: IntelliJ IDEA (2019)



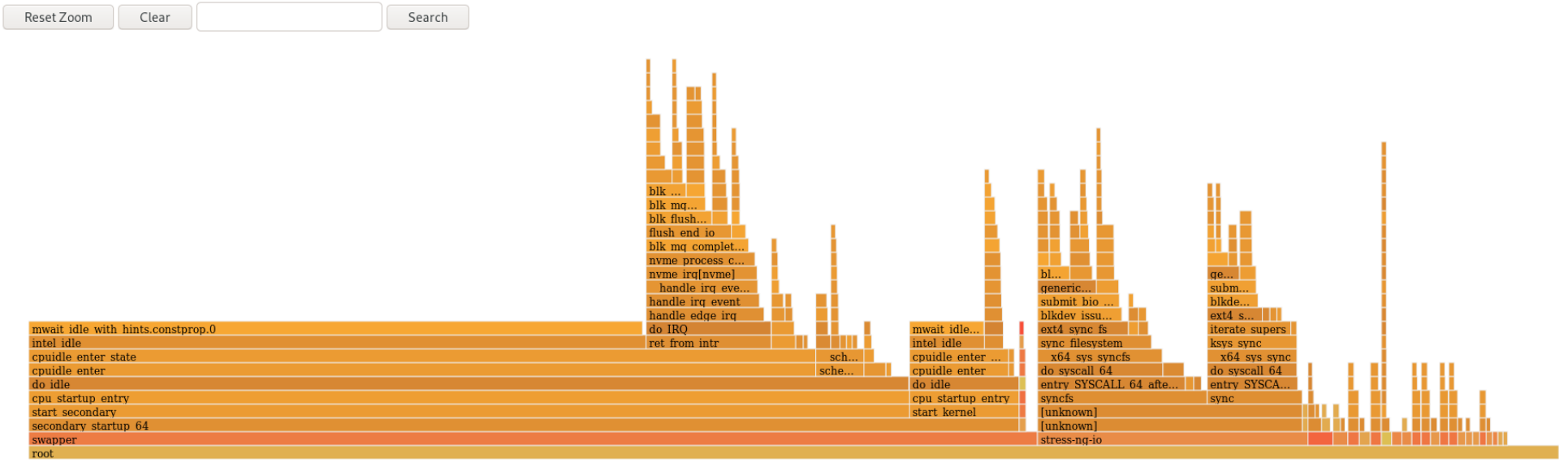
Source: https://blog.jetbrains.com/idea/2019/06/intellij-idea-2019-2-eap-4-profiling-tools-structural-search-preview-and-more/

Firefox: Profiler (2019)



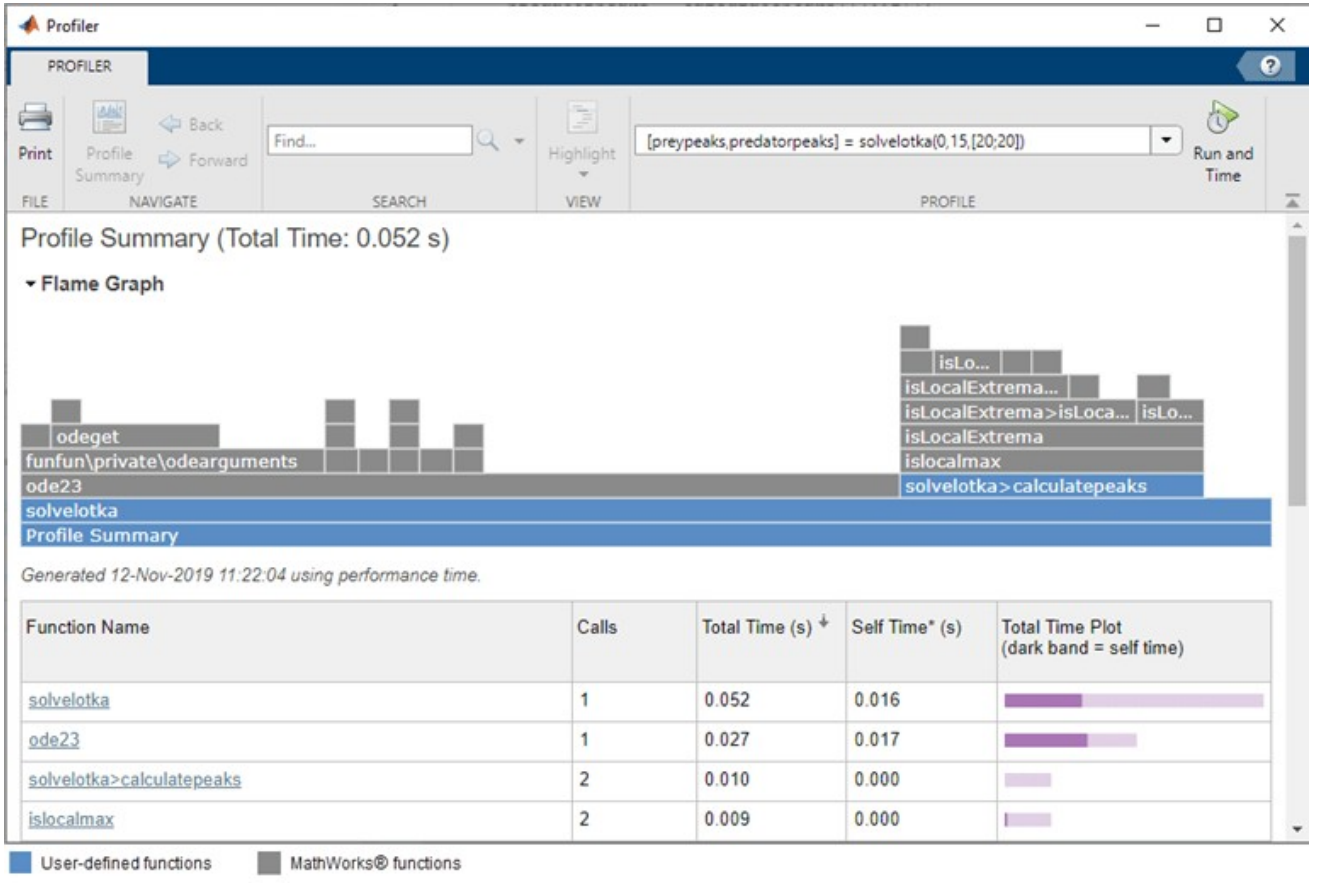
Source: https://profiler.firefox.com

Linux: perf script flamegraph (2020)



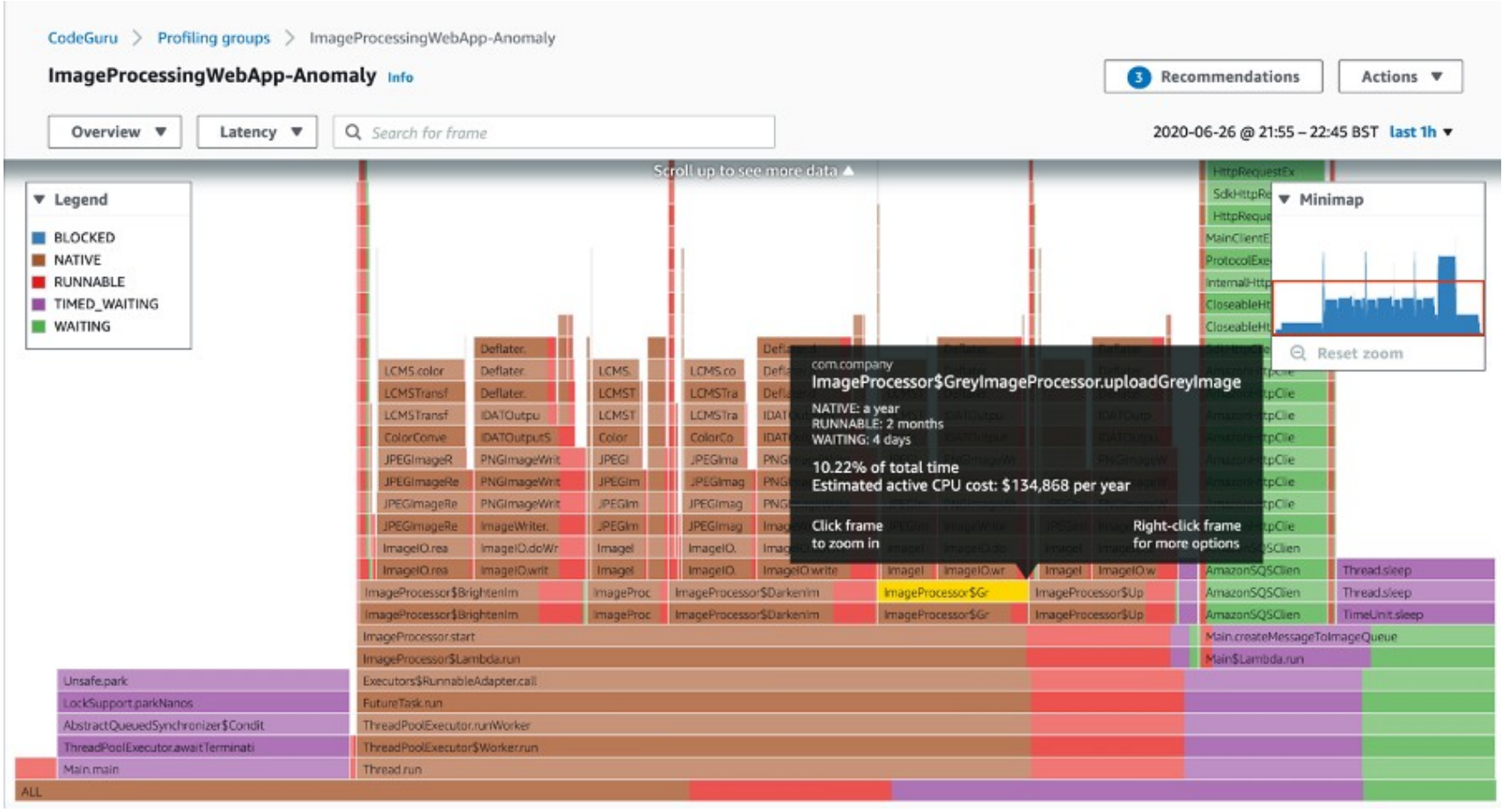
Source: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/monitoring_and_managing_system_status_and_performance/getting-started-with-flamegraphs_monitoring-and-managing-system-status-and-performance (Andreas Gerstmayr)

MathWorks: MATLAB Profiler (2020)



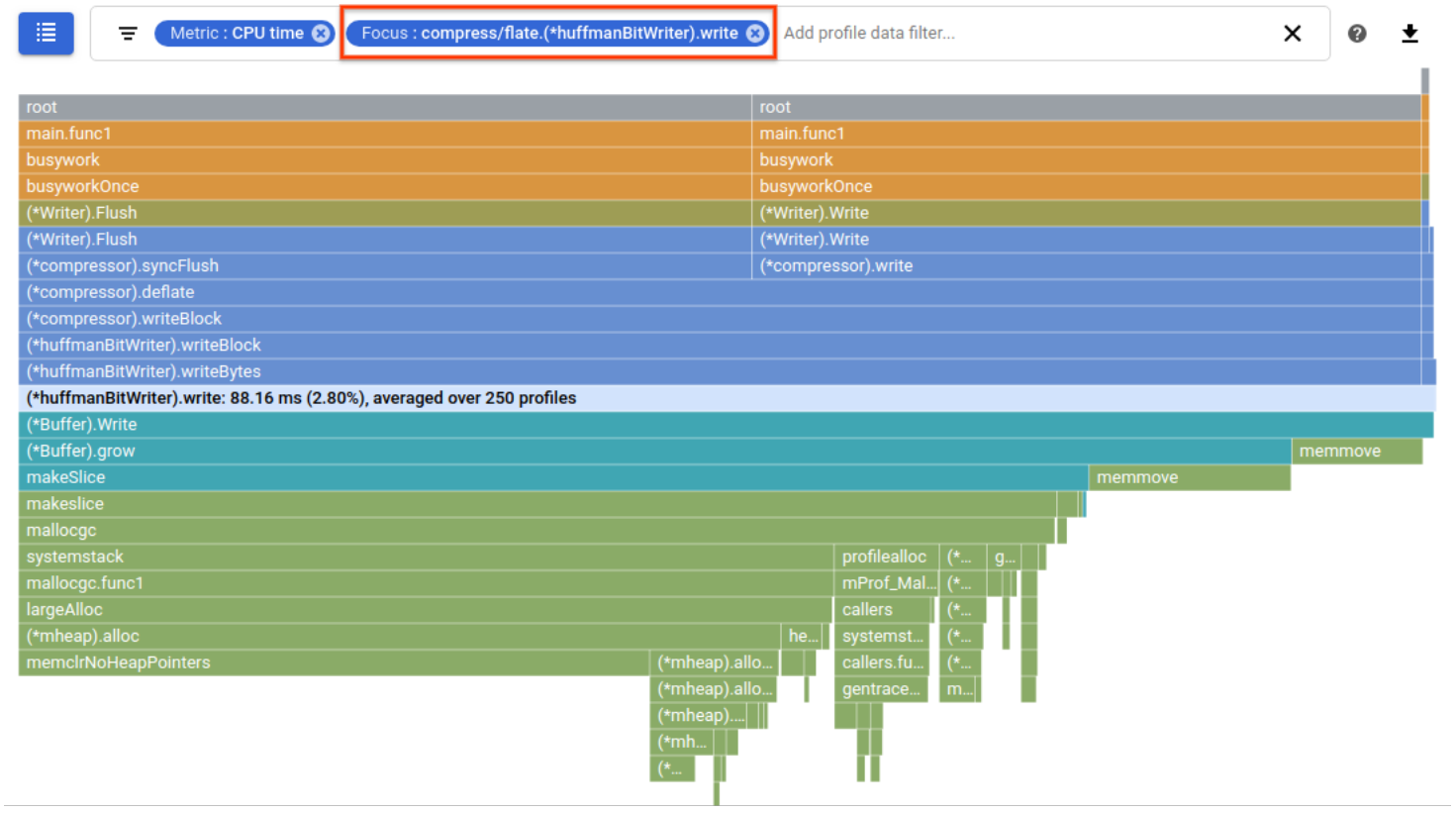
Source: https://www.mathworks.com/help/matlab/matlab_prog/profiling-for-improving-performance.html

AWS: CodeGuru (2020)



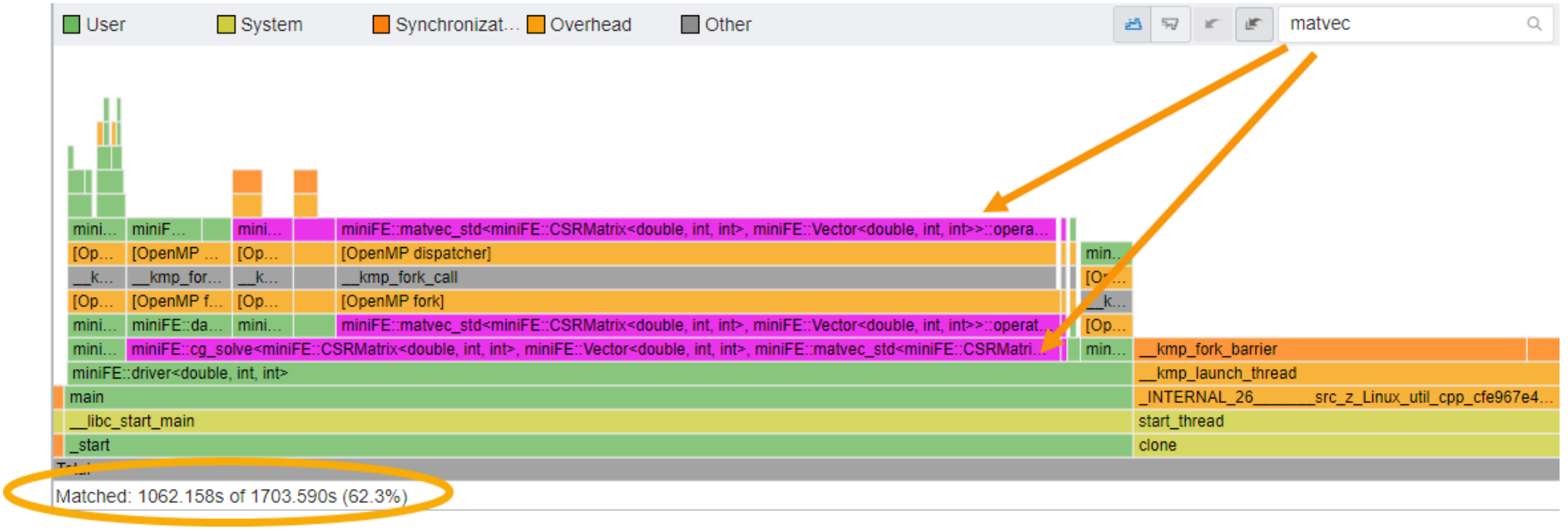
Source: https://aws.amazon.com/codeguru/features/

Google: Cloud Profiler (2020)



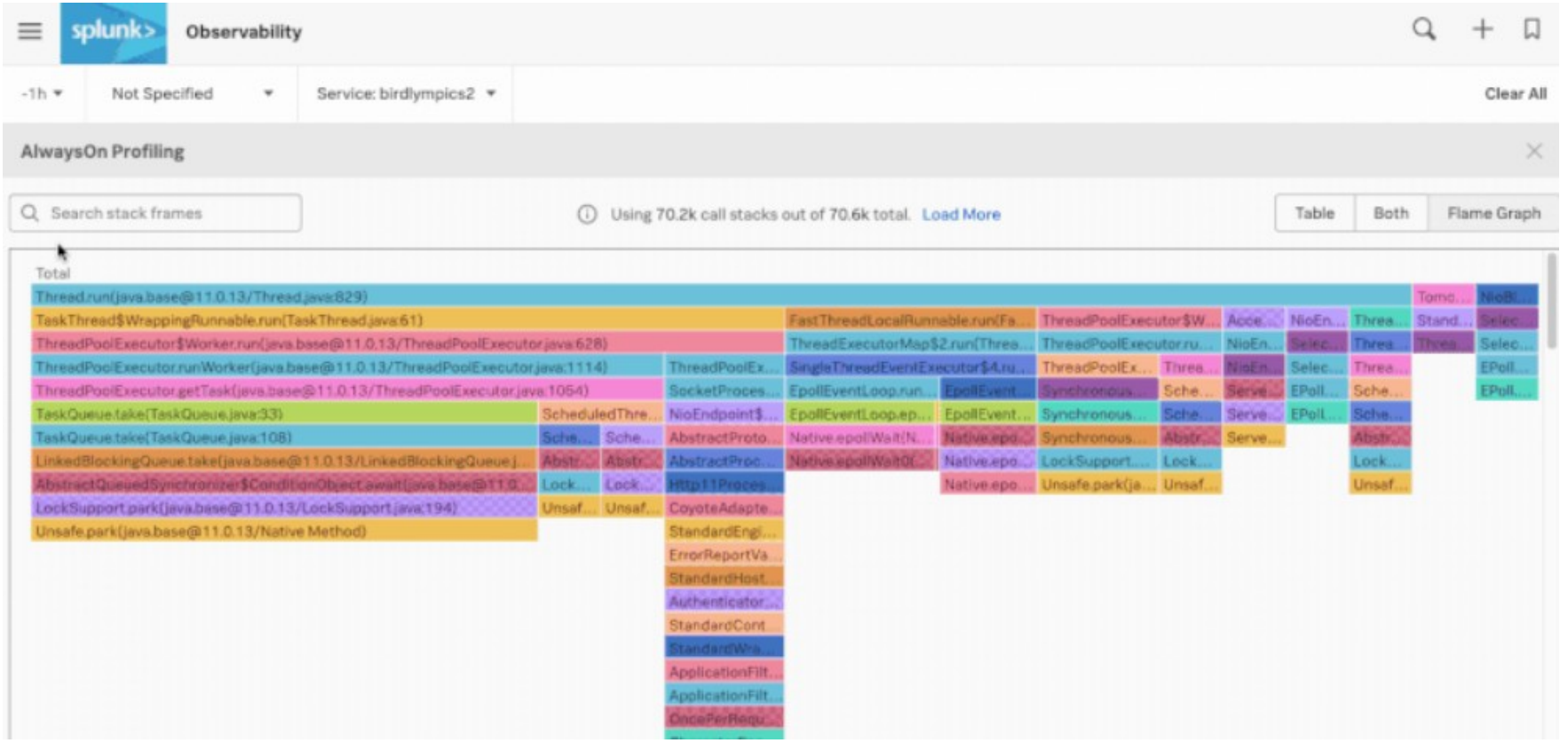
Source: https://cloud.google.com/profiler/docs/focusing-profiles

Intel: vTune (2021)



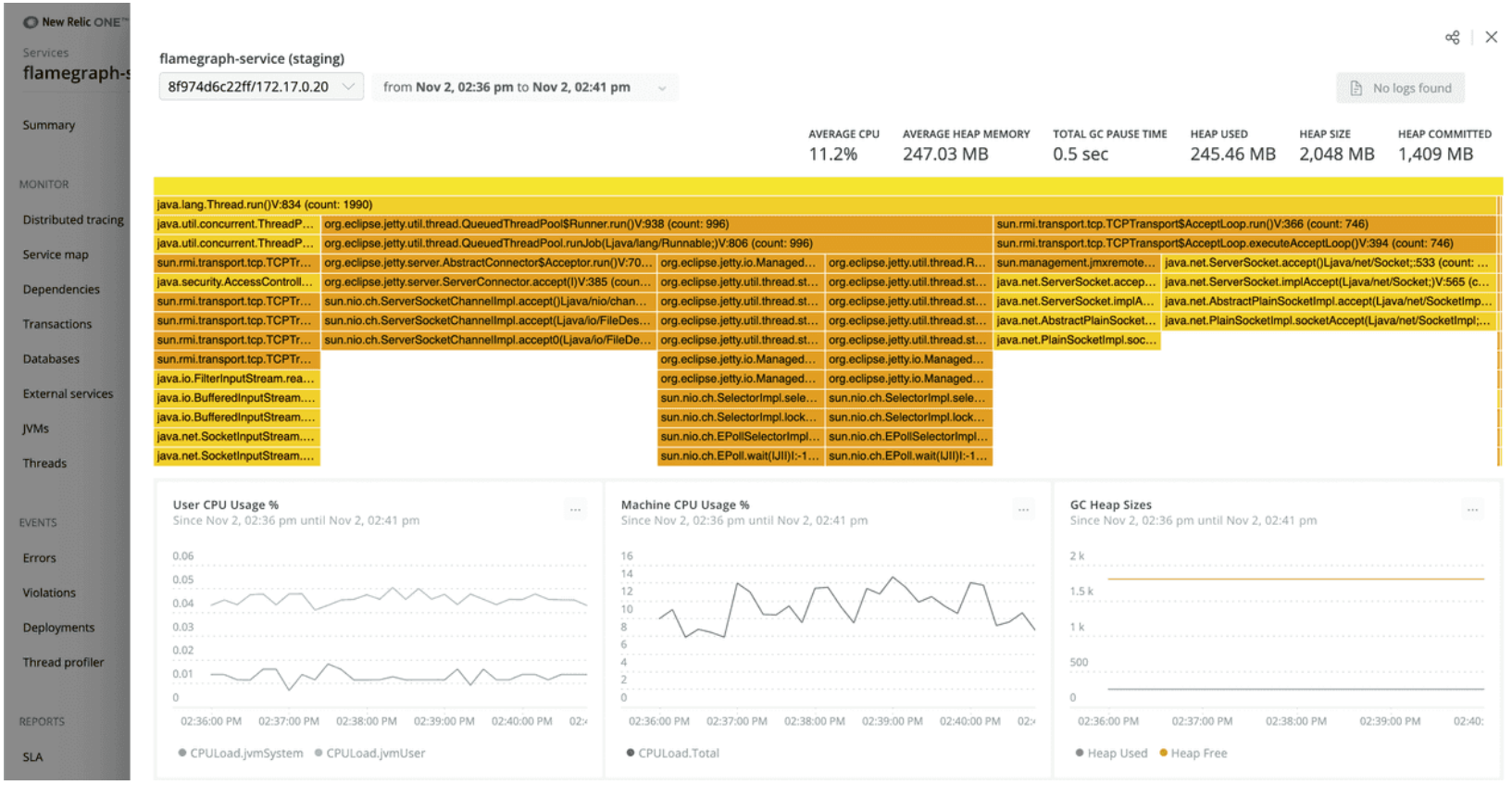
Source: https://www.intel.com/content/www/us/en/develop/documentation/vtune-help/top/reference/user-interface-reference/window-flame-graph.html

Splunk: AlwaysOn Profiling flame graph (2021)



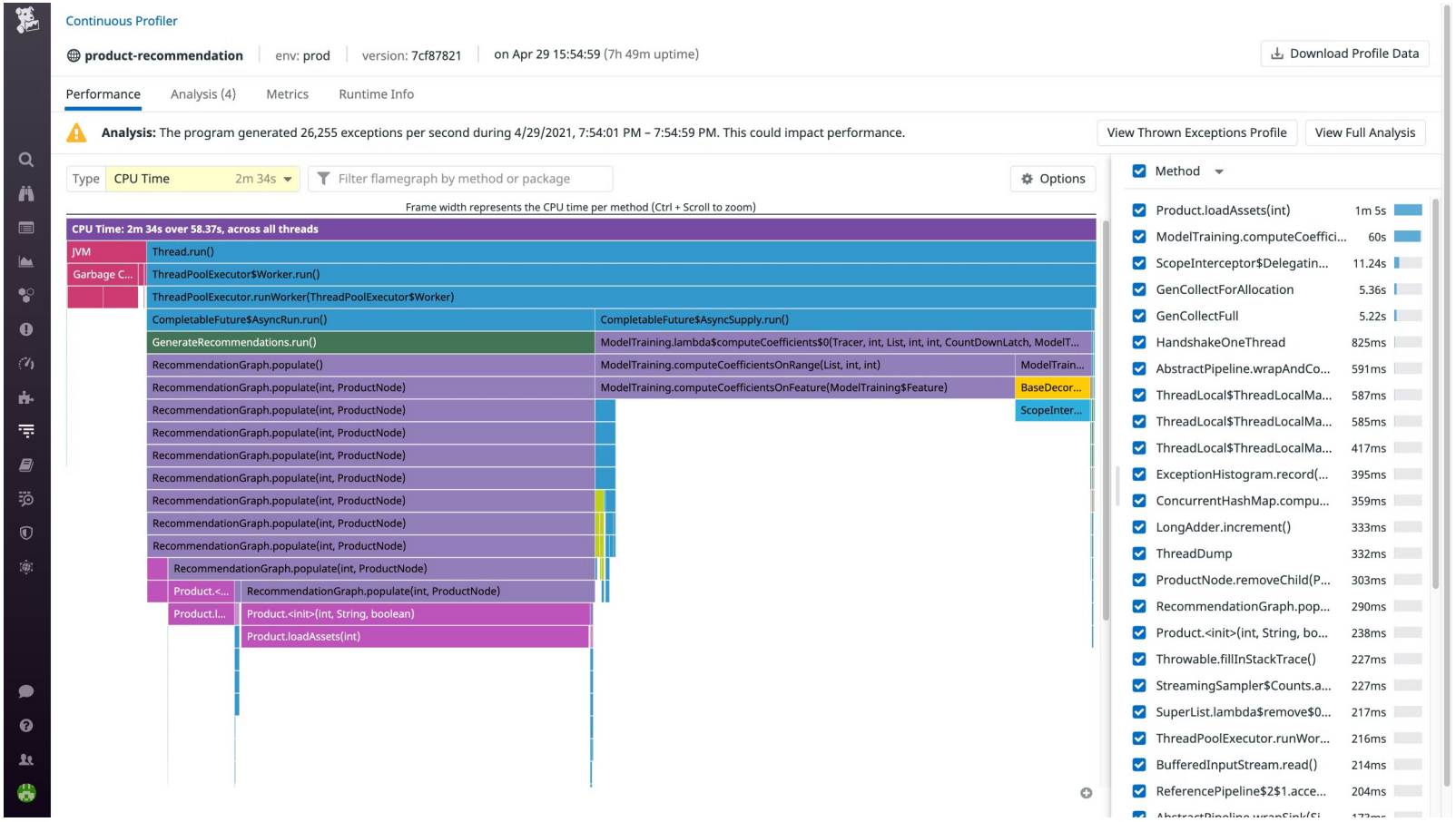
Source: https://docs.splunk.com/Observability/apm/profiling/using-the-flamegraph.html

New Relic: flame graphs (2021)



Source: https://docs.newrelic.com/whats-new/2021/07/whats-new-july-8-realtime-profiling-java/

DataDog: profiling flame graph (2021)

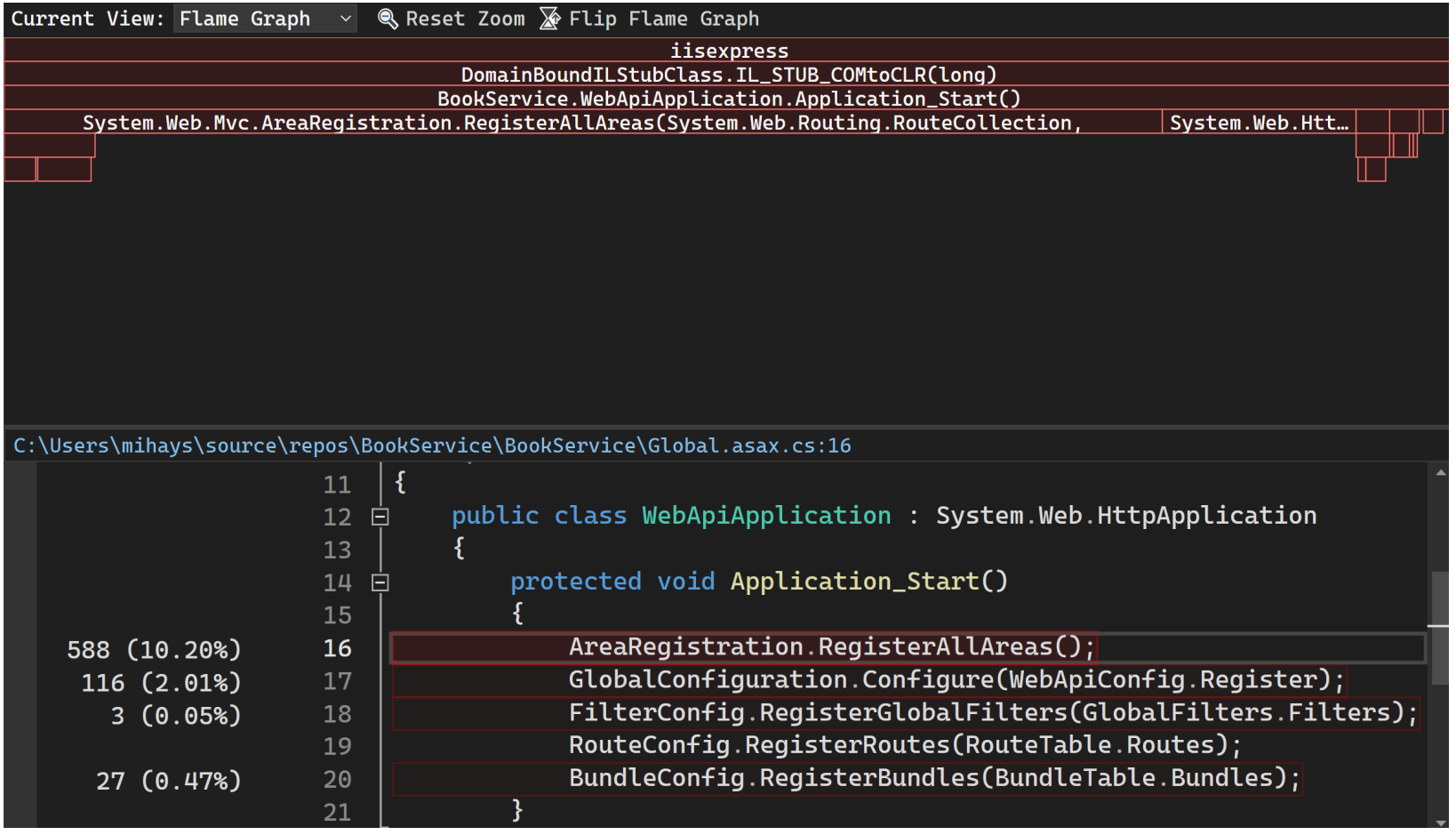


Source: https://www.datadoghq.com/knowledge-center/distributed-tracing/flame-graph/

Granulate: gprofiler (2022; now Intel)

Source: https://docs.gprofiler.io/about-gprofiler/gprofiler-features/views/flame-graph

Microsoft Visual Studio: Flame Graph (2022)



Source: https://learn.microsoft.com/en-us/visualstudio/profiling/flame-graph

GrafanaLabs: Grafana flame graph (2022)

Source: https://grafana.com/docs/grafana/next/panels-visualizations/visualizations/flame-graph

Flame Graph Adoption

Implementations: **>80**

Related open source projects: **>400**

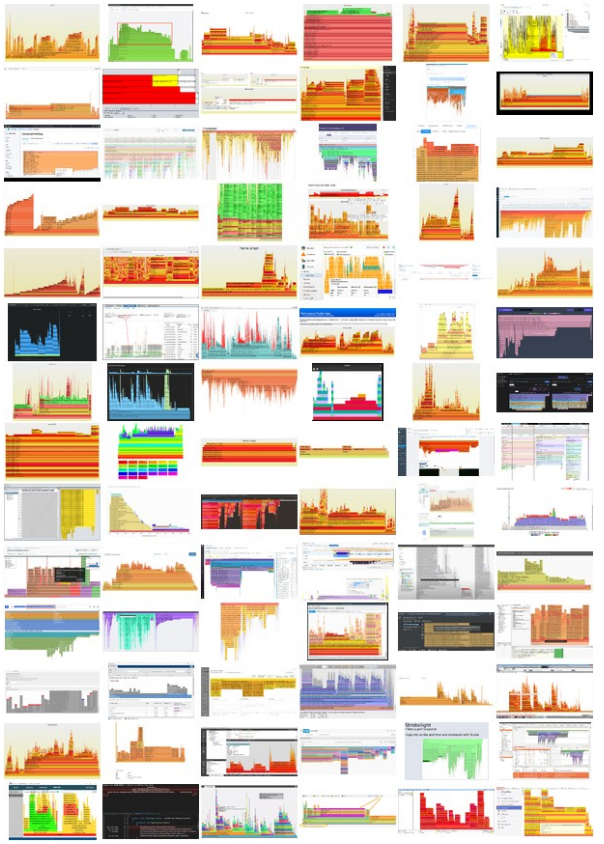
Commercial product adoptions: **>30**

New startups: **4** (so far)

Startup exits: **1** (so far)

Industry investment: **>AUD$1B**

End users: ? (a lot)



2. CPU PROFILING

An Introduction to Flame Graphs

Stack Traces

A code path snapshot. e.g., from jstack(1):

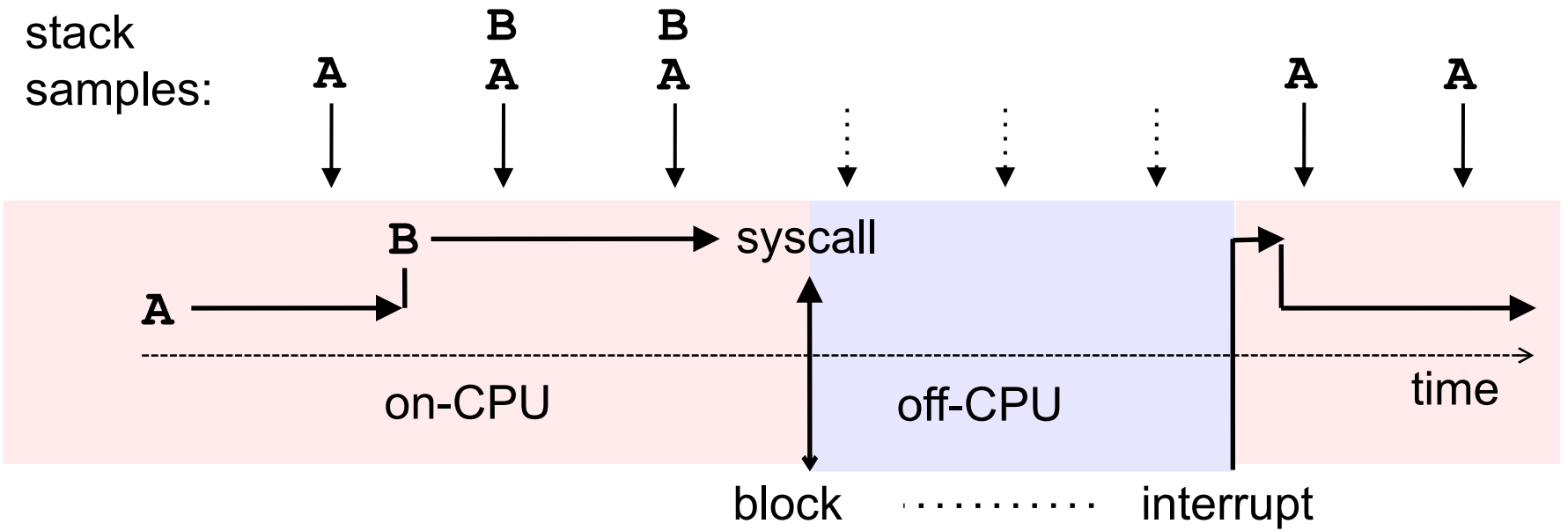
```
$ jstack 1819

[…]

"main" prio=10 tid=0x00007ff304009000
nid=0x7361 runnable [0x00007ff30d4f9000]

  java.lang.Thread.State: RUNNABLE

    at Func_abc.func_c(Func_abc.java:6)

    at Func_abc.func_b(Func_abc.java:16)

    at Func_abc.func_a(Func_abc.java:23)

    at Func_abc.main(Func_abc.java:27)
```

running
parent
g.parent
g.g.parent

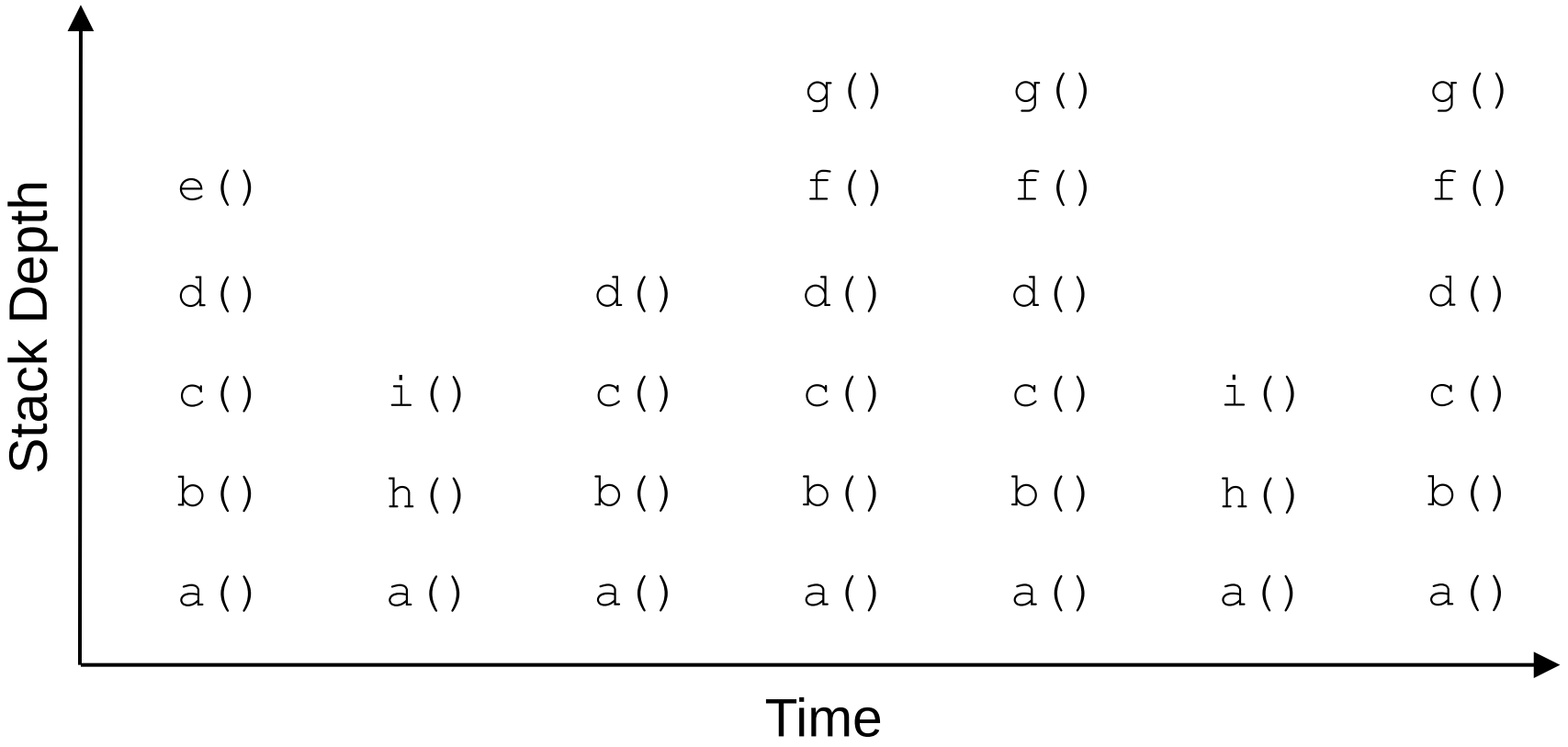
CPU Profiling

Record stacks at a timed interval

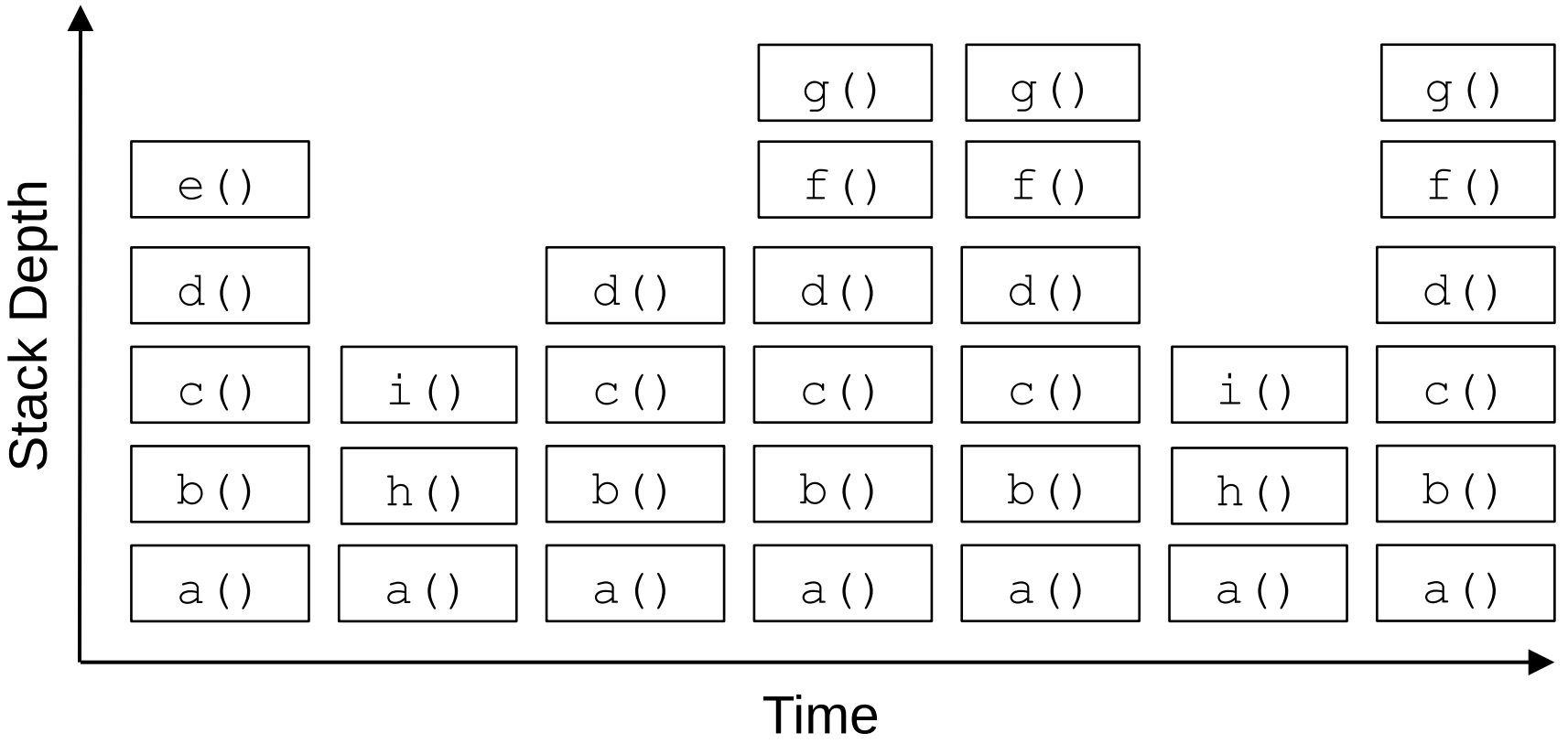
- Pros: Low (deterministic) overhead
- Cons: Coarse accuracy, but usually sufficient



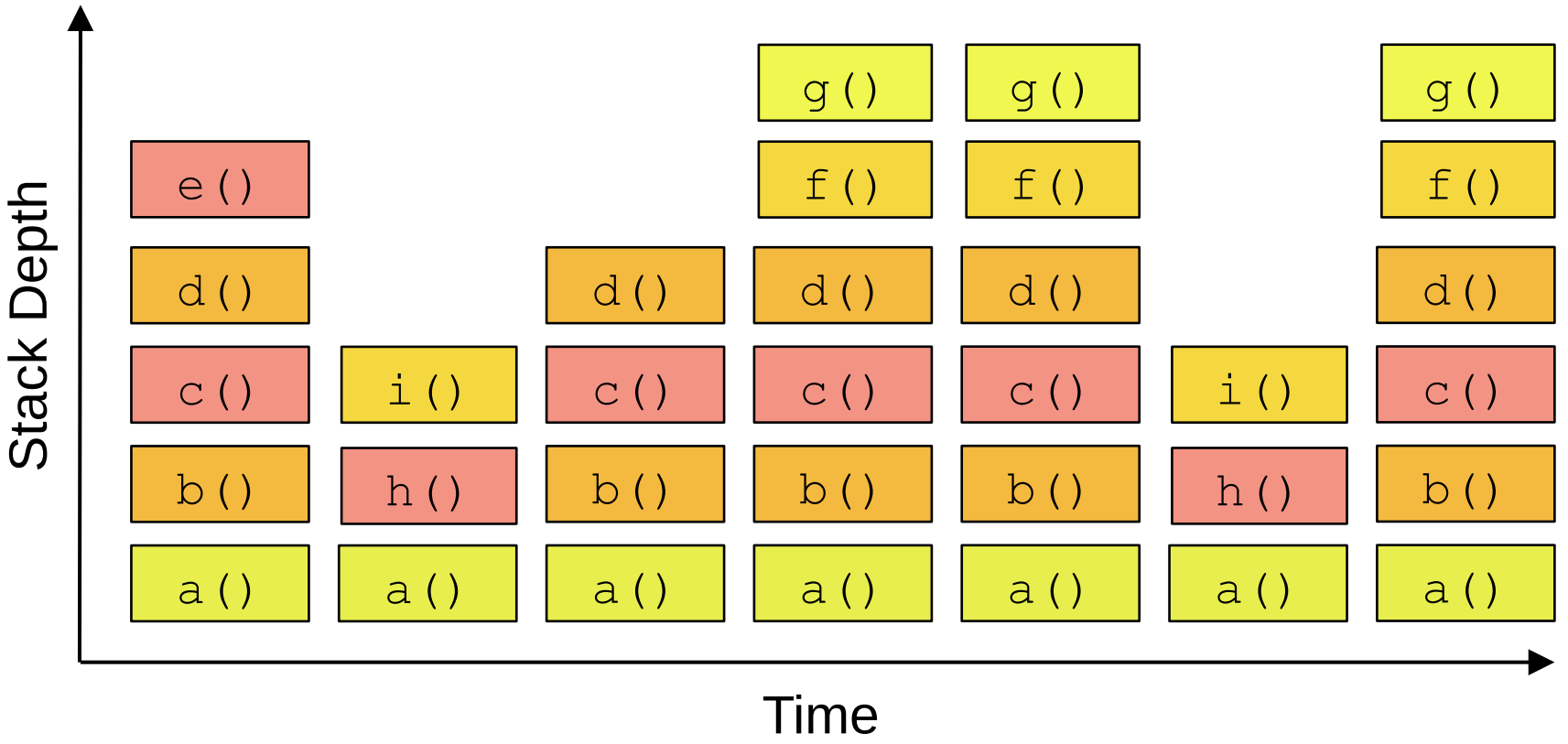
Stack Samples



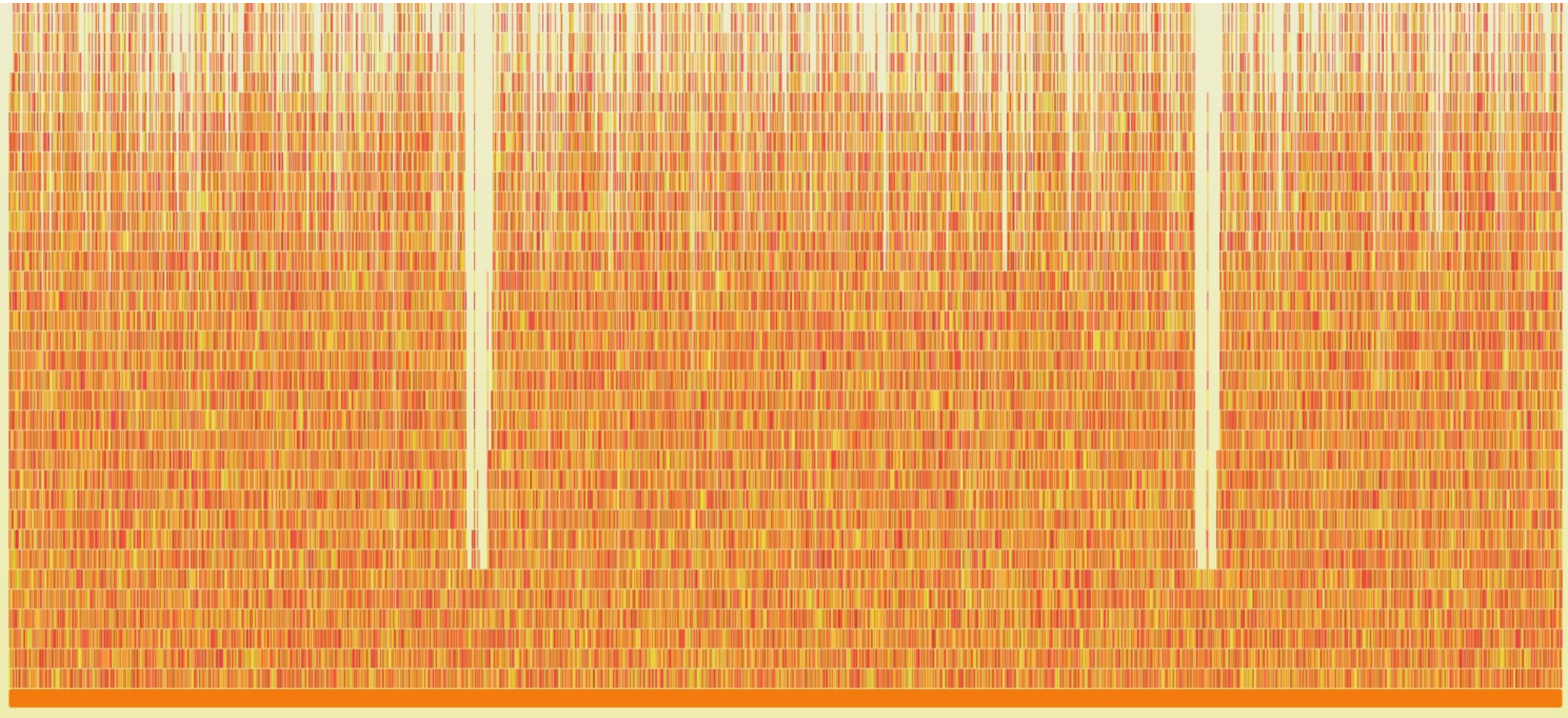
Stack Samples



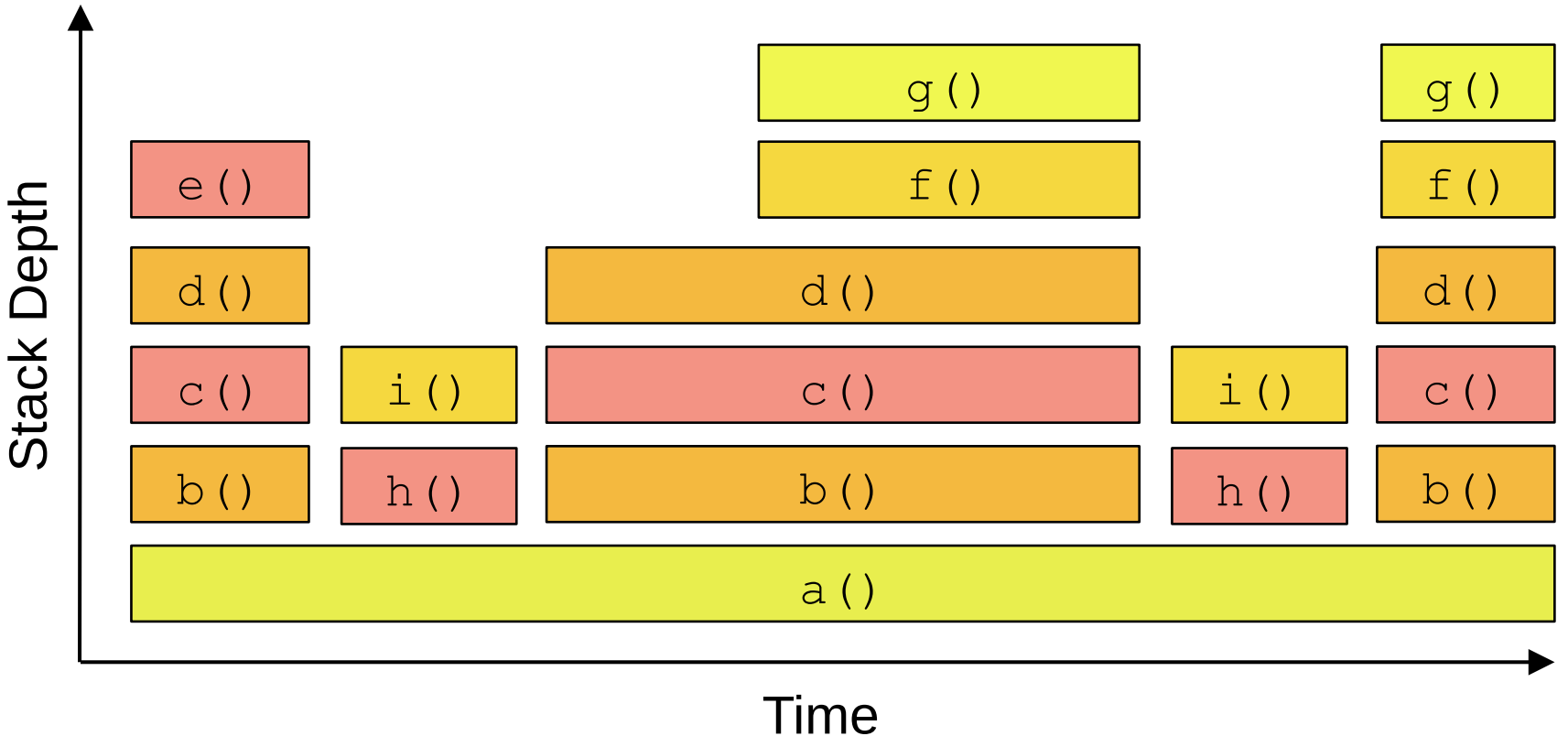
Stack Samples



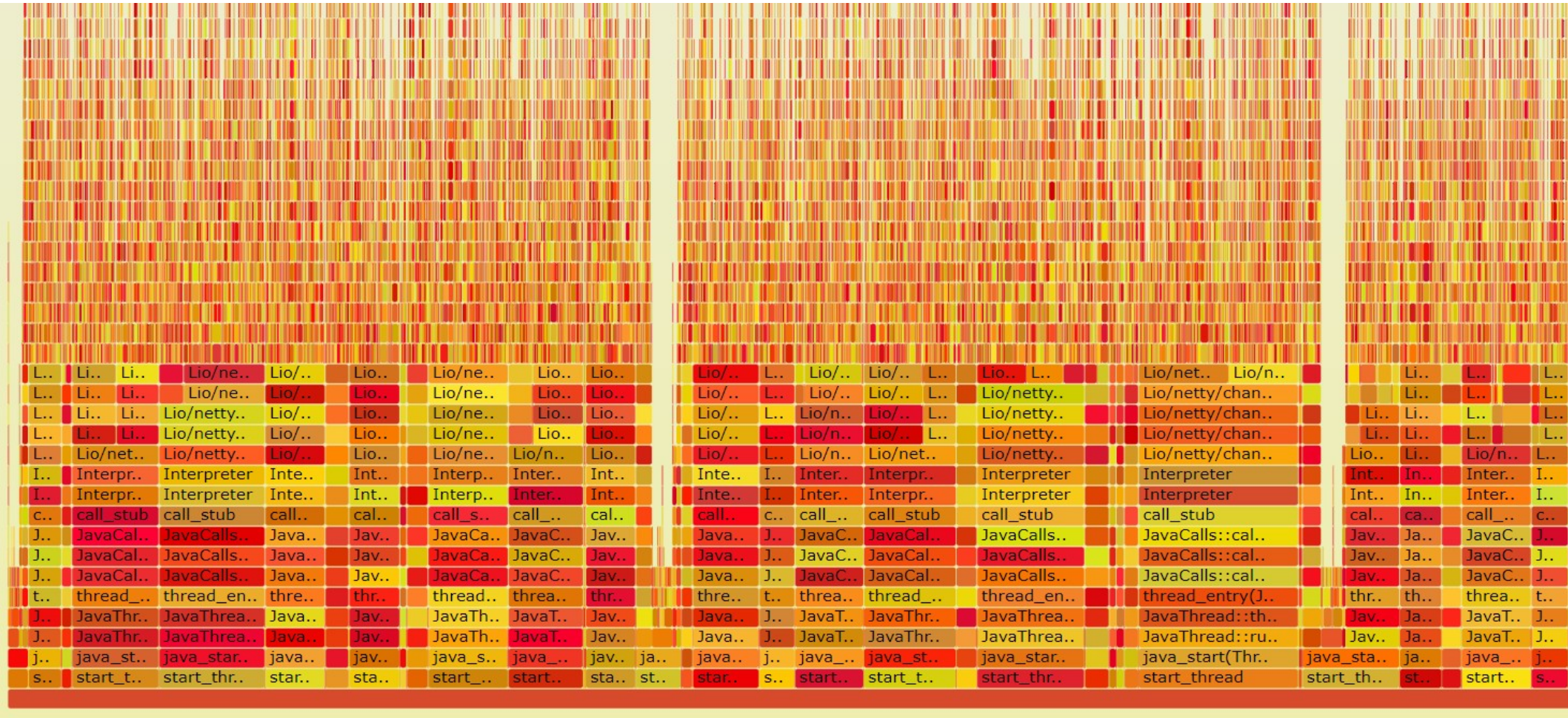
Example Profile (“hair graph”)



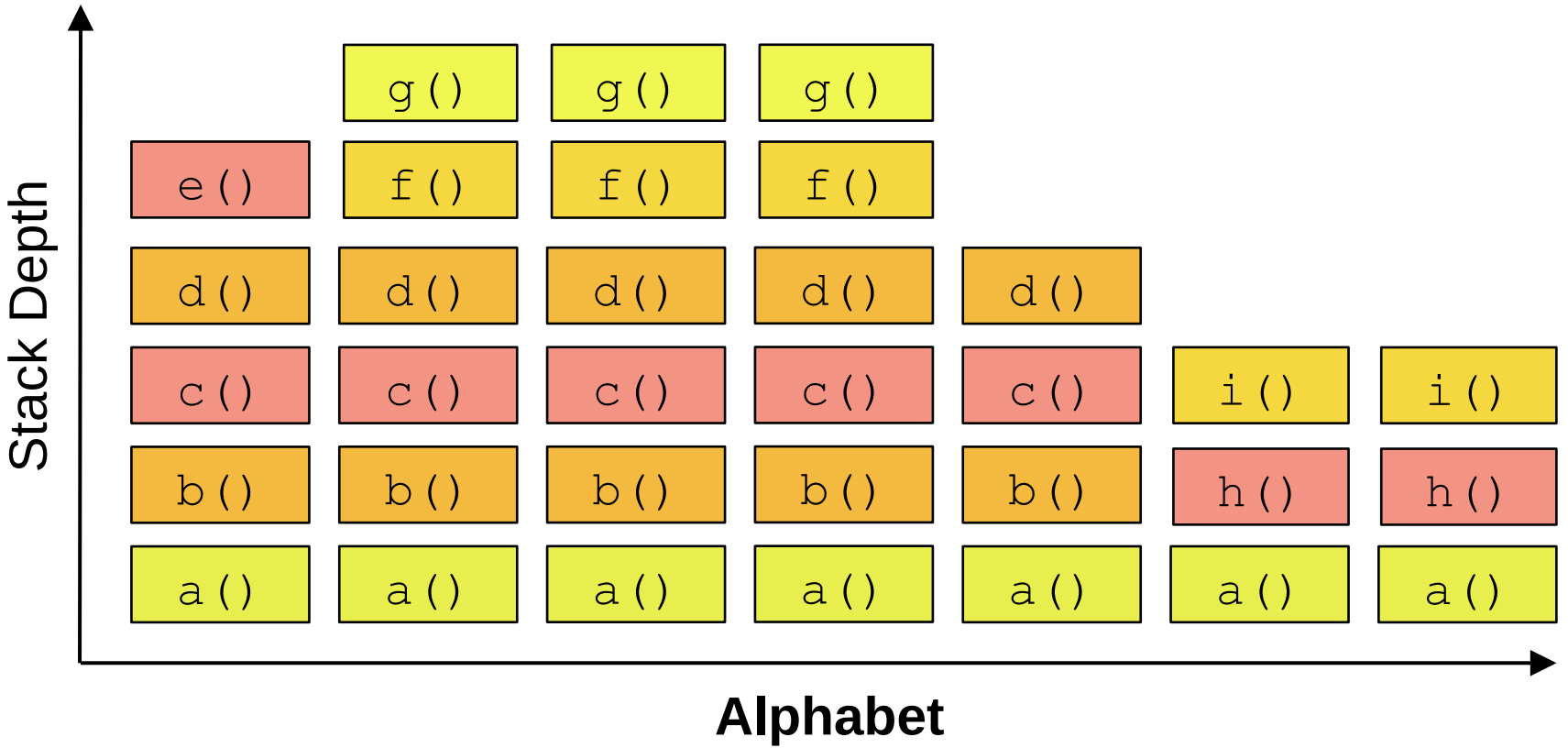
Stack Samples: Merged



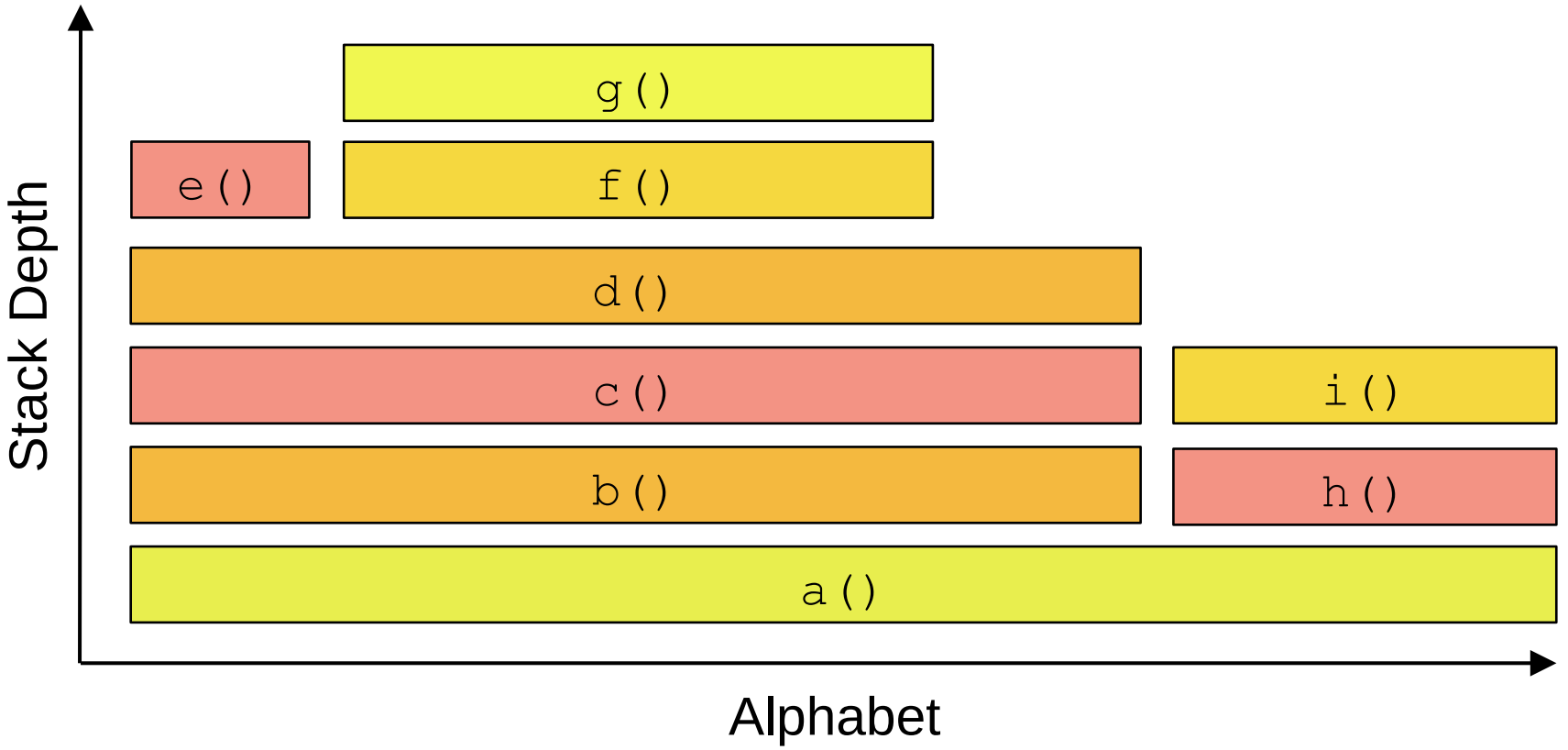
Example Profile: Merged



Alphabet Sort

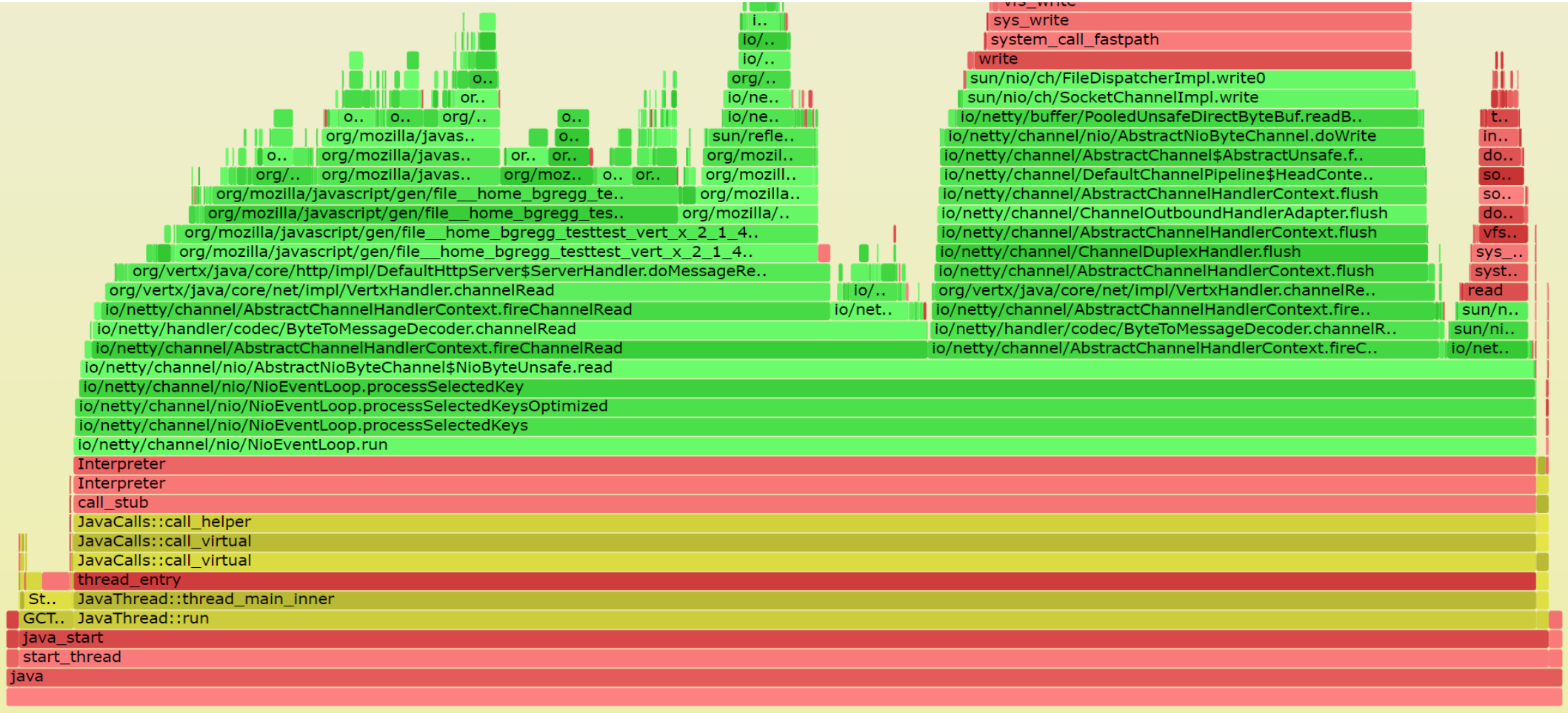


Alphabet Merged (“Flame Graph”)

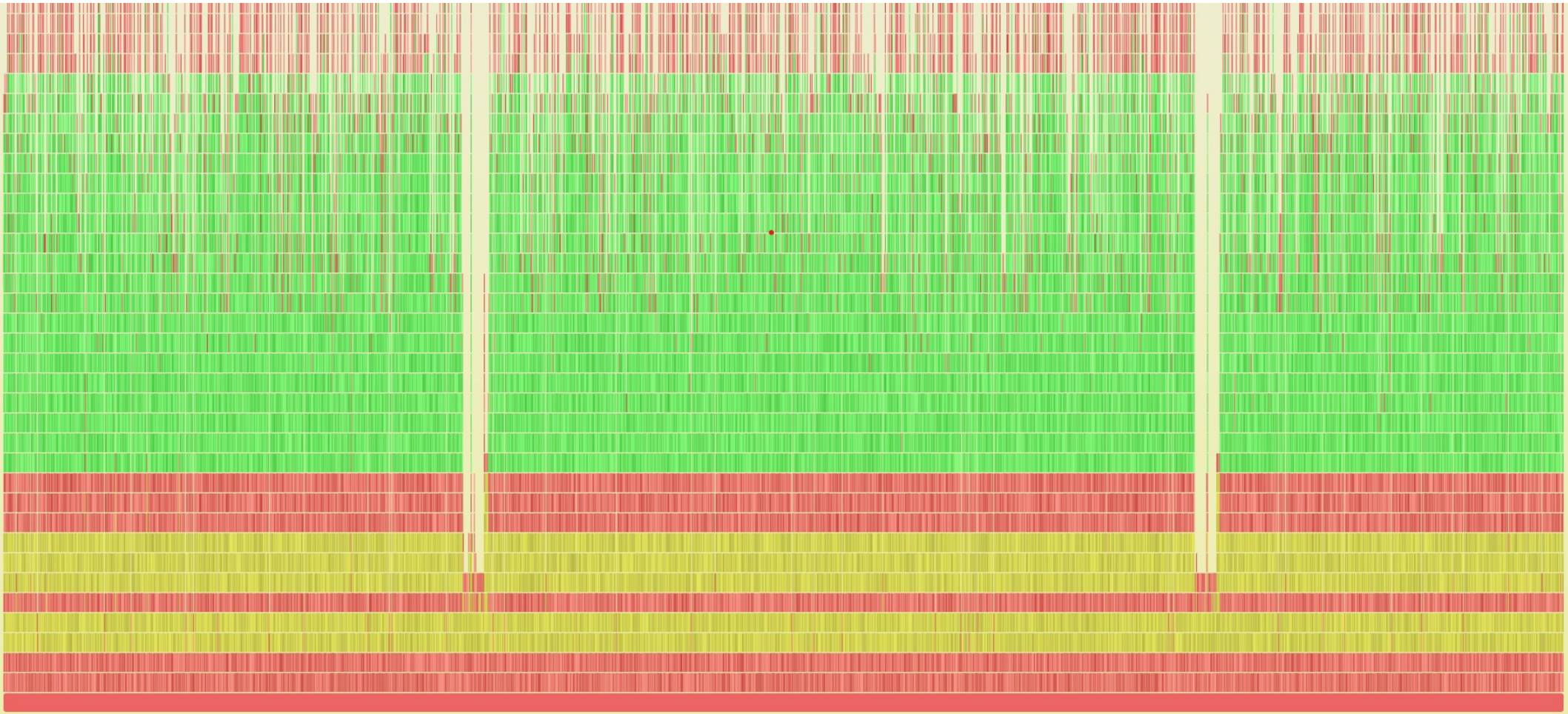


Example Profile: Flame Graph

Example Profile: Flame Graph (with code hues)



Replay 1/3: Time Columns



Replay 2/3: Time Merged (aka “Flame Chart”)

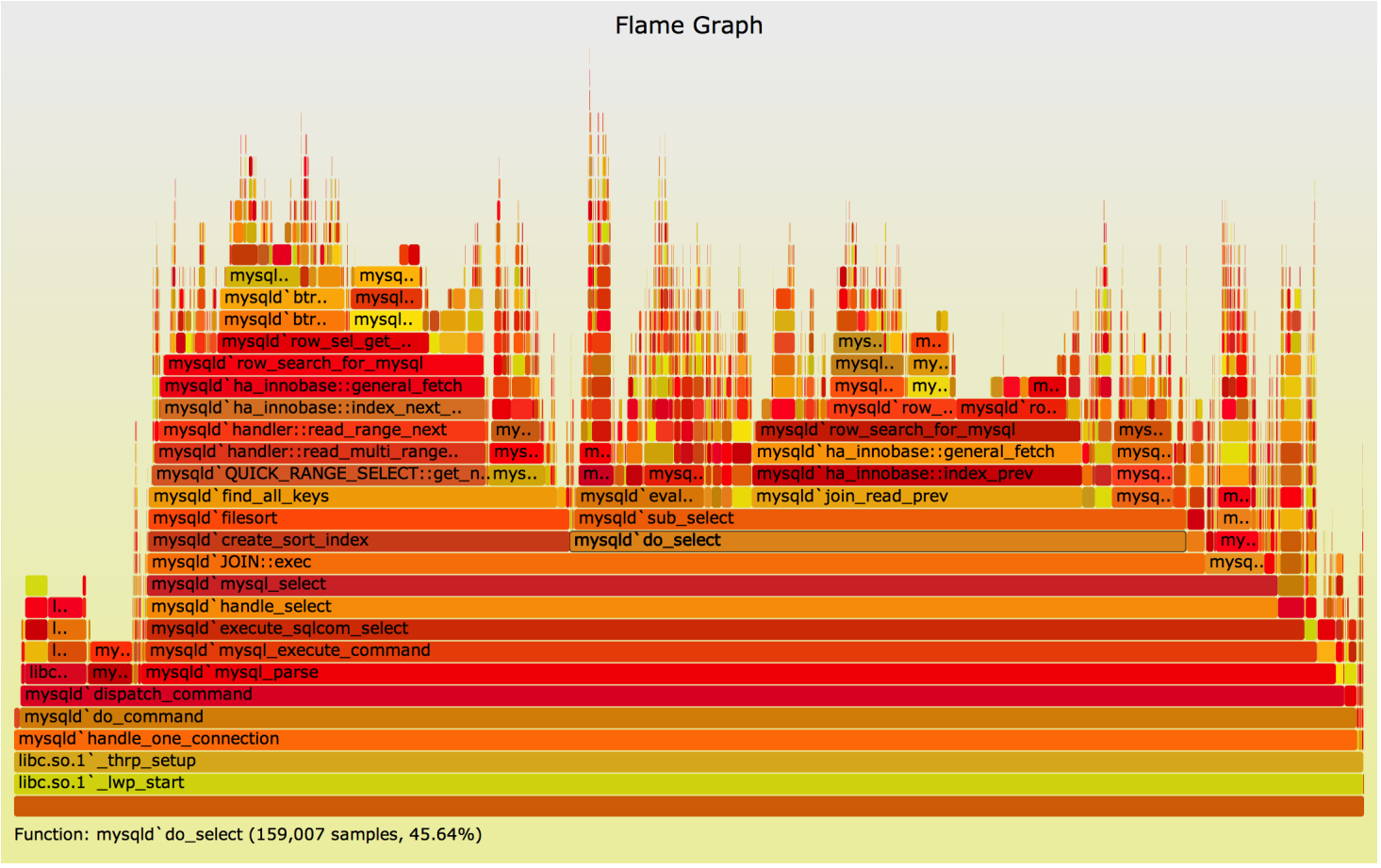
Replay 3/3: Flame Graph

Origin (2011): CPU Profiling

```
# dtrace -x ustackframes=100 -n 'profile-997 /execname == "mysqld"/ {
    @[ustack()] = count(); } tick-60s { exit(0); }'
[… over 500,000 lines truncated …]

              libc.so.1`__priocntlset+0xa
              libc.so.1`getparam+0x83
              libc.so.1`pthread_getschedparam+0x3c
              libc.so.1`pthread_setschedprio+0x1f
              mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x9ab
              mysqld`_Z10do_commandP3THD+0x198
              mysqld`handle_one_connection+0x1a6
              libc.so.1`_thrp_setup+0x8d
              libc.so.1`_lwp_start
             4884

              mysqld`_Z13add_to_statusP17system_status_varS0_+0x47
              mysqld`_Z22calc_sum_of_all_statusP17system_status_var+0x67
              mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x1222
              mysqld`_Z10do_commandP3THD+0x198
              mysqld`handle_one_connection+0x1a6
              libc.so.1`_thrp_setup+0x8d
              libc.so.1`_lwp_start
             5530
```

Full output

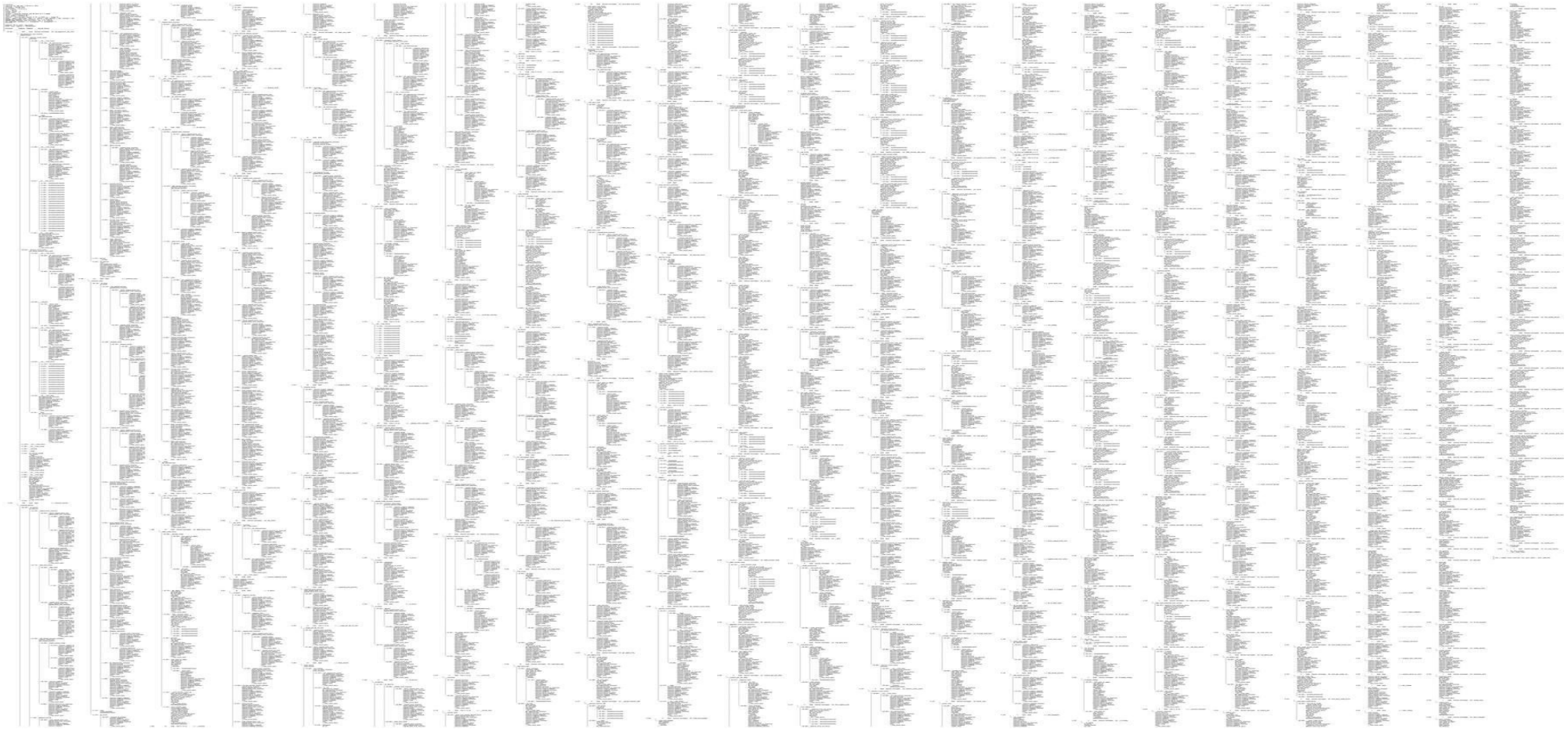
… as a Flame Graph



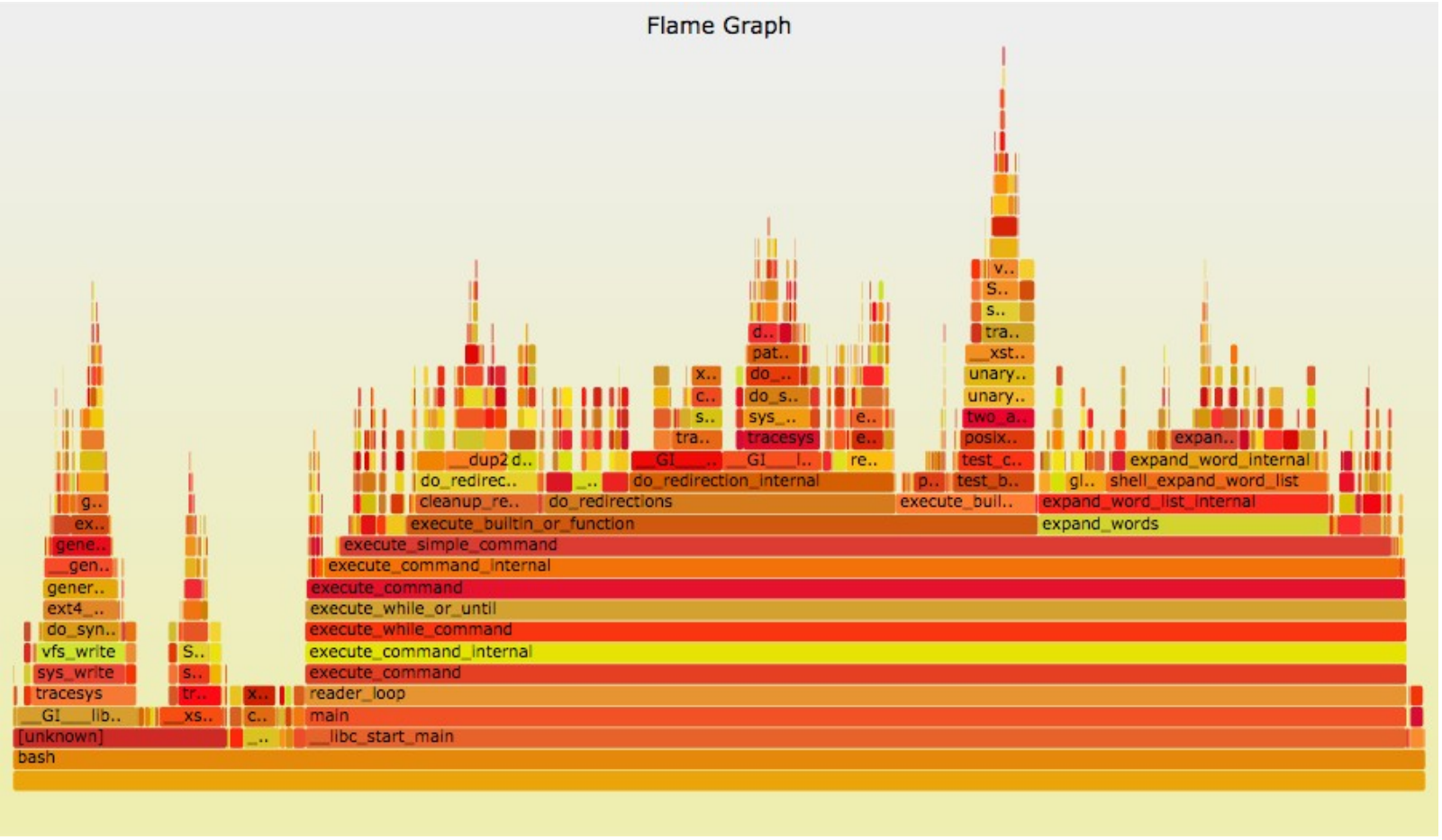
Linux example: perf Profiling

```
# perf record -F 99 -ag -- sleep 30
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 2.745 MB perf.data (~119930 samples) ]
# perf report -n -stdio
[…]
# Overhead       Samples  Command      Shared Object                          Symbol
# ........  ............  .......  .................  ..........................
#
    20.42%           605     bash  [kernel.kallsyms]  [k] xen_hypercall_xen_version
               |
               --- xen_hypercall_xen_version
                   check_events
                  |
                  |--44.13%-- syscall_trace_enter
                  |          tracesys
                  |          |
                  |          |--35.58%-- __GI___libc_fcntl
                  |          |          |
                  |          |          |--65.26%-- do_redirection_internal
                  |          |          |          do_redirections
                  |          |          |          execute_builtin_or_function
                  |          |          |          execute_simple_command
[… ~13,000 lines truncated …]
```

Full perf Output



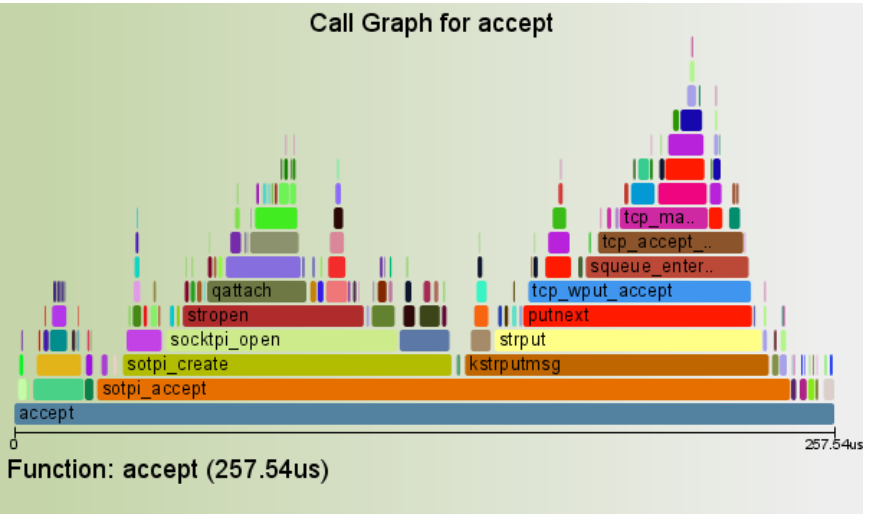
… as a Flame Graph



Inspiration

Neelakanth Nadgir's function_call_graph.rb (2007):

- It was inspired by Roch Bourbonnais's CallStackAnalyzer, which was inspired by Jan Boerhout's vftrace.
- The x-axis is time, and it shows a complete function trace.
- Flame graphs are different: The x-axis is the *population*, and they can show function traces or *stack samples*.




```
# more flamegraph.pl
[…]
# This was inspired by Neelakanth Nadgir's excellent function_call_graph.rb
# program, which visualized function entry and return trace events.  As Neel
# wrote: "The output displayed is inspired by Roch's CallStackAnalyzer which
# was in turn inspired by the work on vftrace by Jan Boerhout".  See:
# https://blogs.oracle.com/realneel/entry/visualizing_callstacks_via_dtrace_and
[…]
```

Image source: https://blogs.oracle.com/realneel/entry/visualizing_callstacks_via_dtrace_and

Flame Graph Summary

Visualizes a collection of stack traces

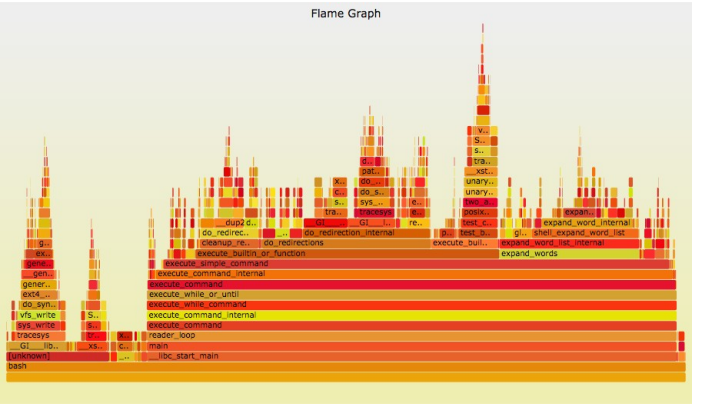
- **x-axis**: population: e.g., alphabetical sort to maximize merging
- **y-axis**: stack depth
- **color**: random (default) or a dimension

Original implementation: Perl + SVG + JavaScript

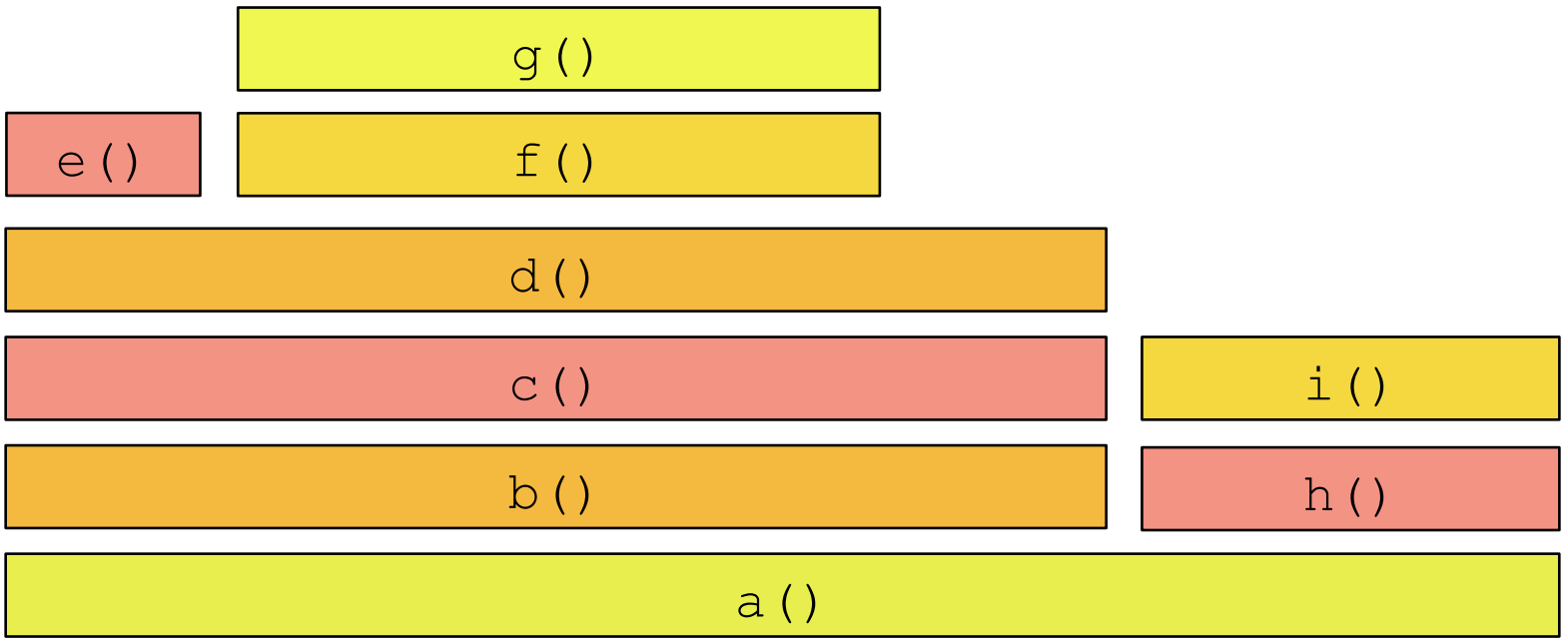
- **https://github.com/brendangregg/FlameGraph**
- Takes input from many different profilers

References:

- **http://www.brendangregg.com/flamegraphs.html**
- **http://queue.acm.org/detail.cfm?id=2927301**
- **"The Flame Graph" CACM, June 2016**



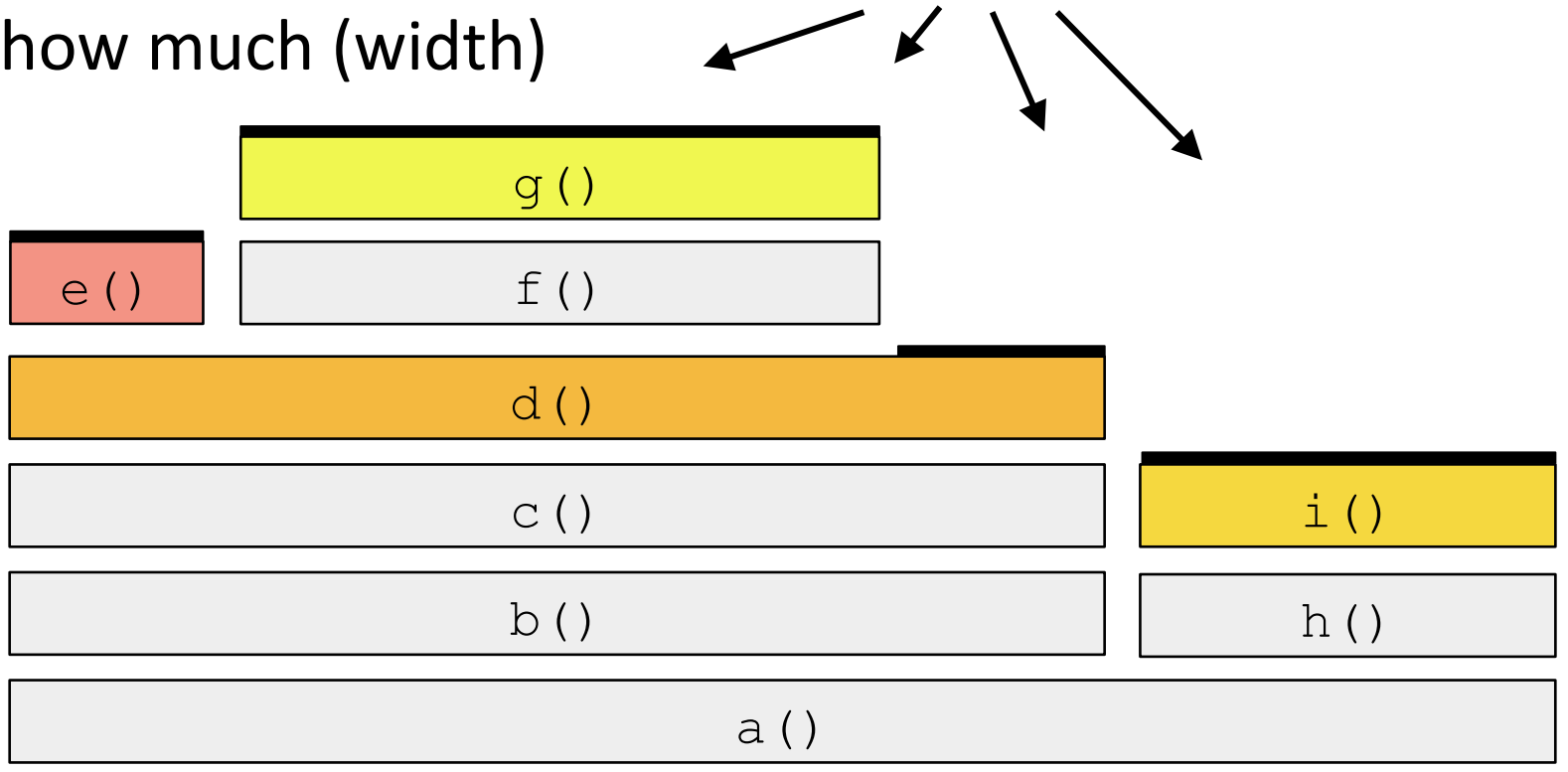
Flame Graph Interpretation



Flame Graph Interpretation (1/4)

Top edge shows who is running on-CPU,

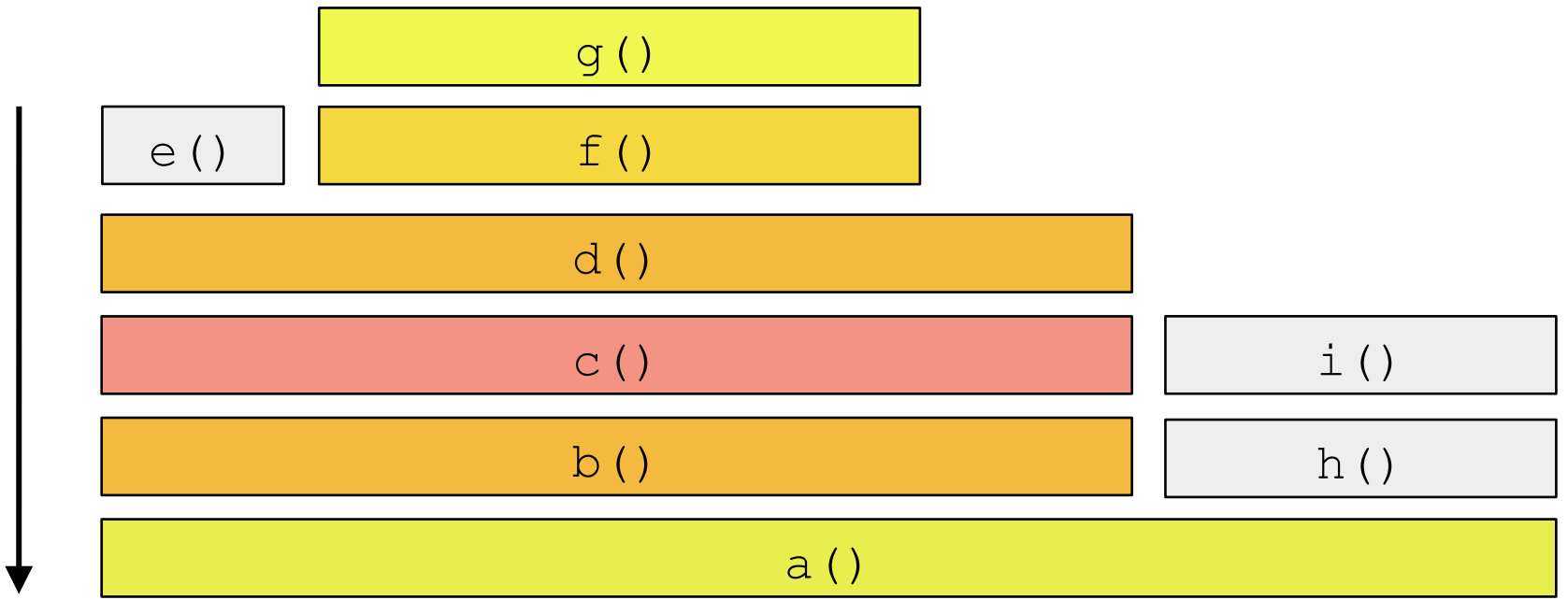
and how much (width)



Flame Graph Interpretation (2/4)

Top-down shows ancestry

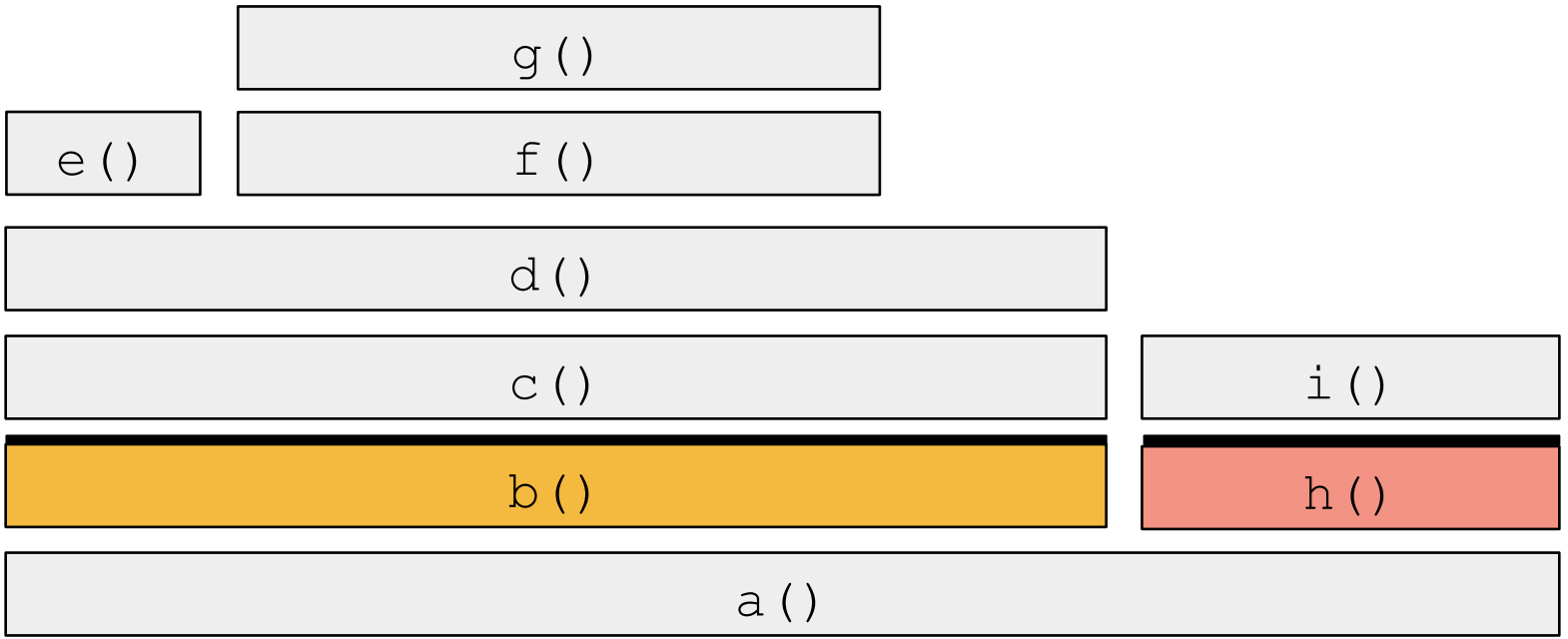
e.g., from g():



Flame Graph Interpretation (3/4)

Widths are proportional to presence in samples

e.g., comparing b() to h() (incl. children)

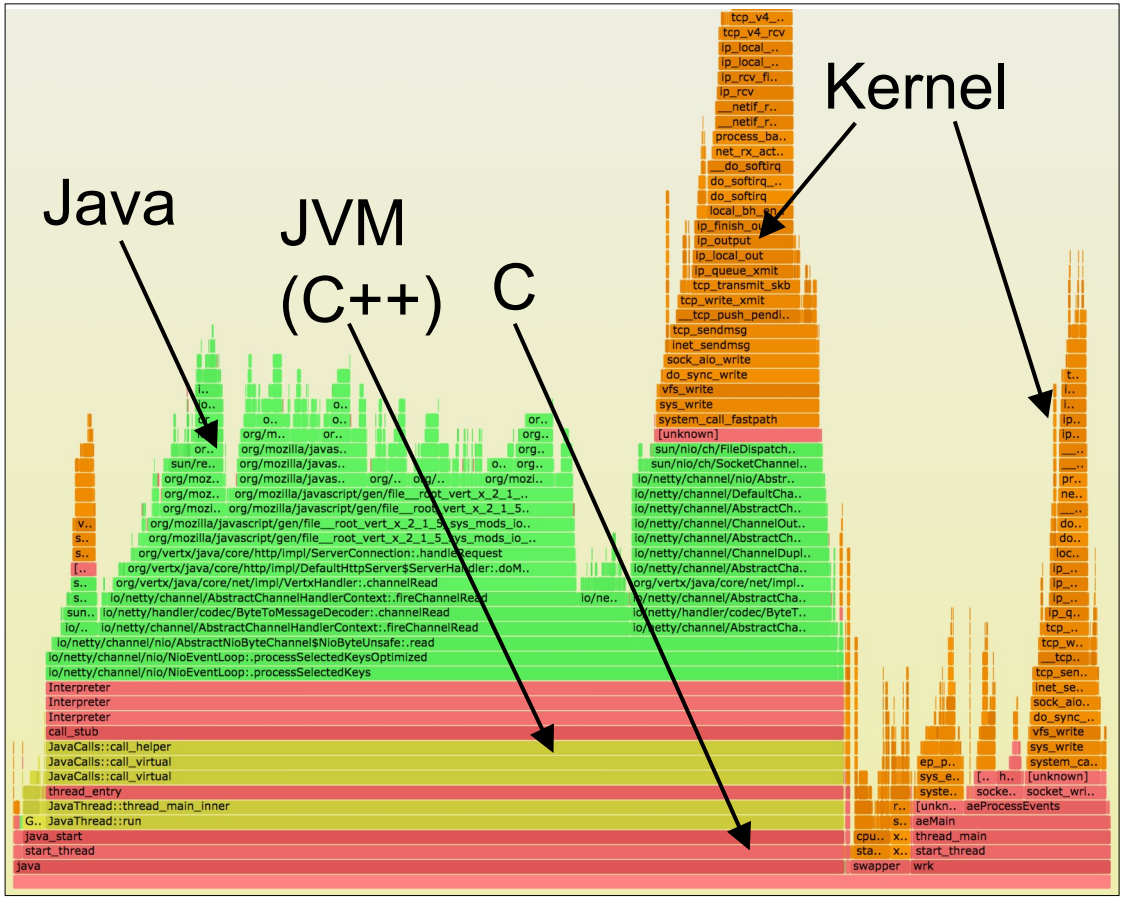


Flame Graph Interpretation (4/4)

Colors randomized to differentiate frames

Or used for code type; e.g.:

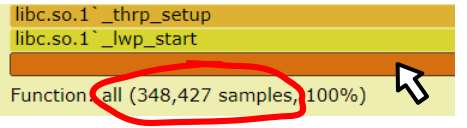
- green == JIT (e.g., Java)
- aqua == inlined
- red == user-level
- orange == kernel
- yellow == C++
- magenta == search term



CPU Flame Graph Tips & Tricks

A) Check sample count (bottom frame): idle system?

- E.g., 49 Hertz x 30 sec x 16 CPUs == 23,520 samples at 100% CPU utilization. <500 samples total would mean <2% busy and probably not interesting!



B) Off-CPU time (I/O, locks) not present

- But their initialization/spin breadcrumbs may be present
- Can use off-CPU flame graphs for this (covered later)

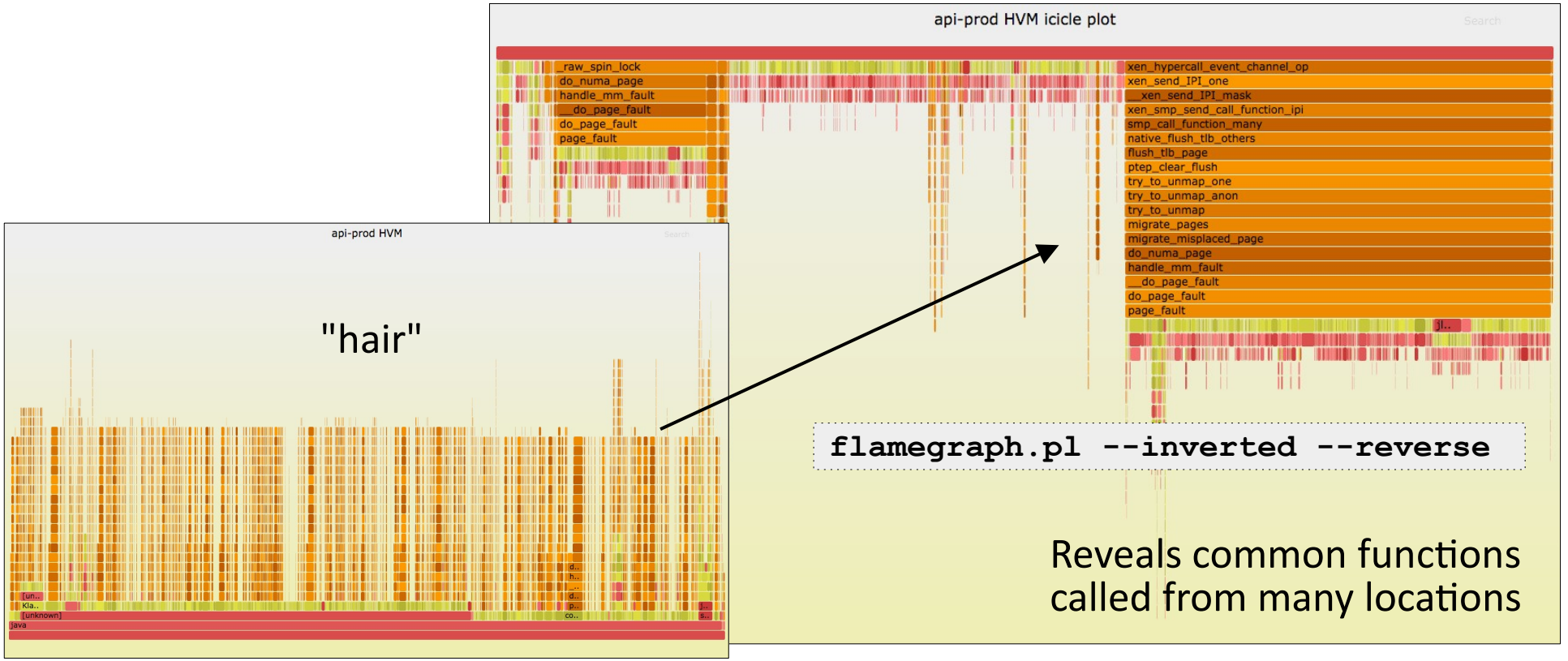
C) Some tiny CPU code paths may be missing

- E.g., some kernel code paths that disable interrupts, and hypervisor time from the guest.
- This is a detail of the profiler and target. Flame graphs just show what the profiler sees.
- Flame graphs may drop tiny frames that are <1 pixel wide unless zoomed in, just to speed up rendering.

D) Too much "hair"? Try a leaf merge…

- Merge stack frames from leaf to root. The default is root to leaf.

Icicle Graph with Leaf Merge



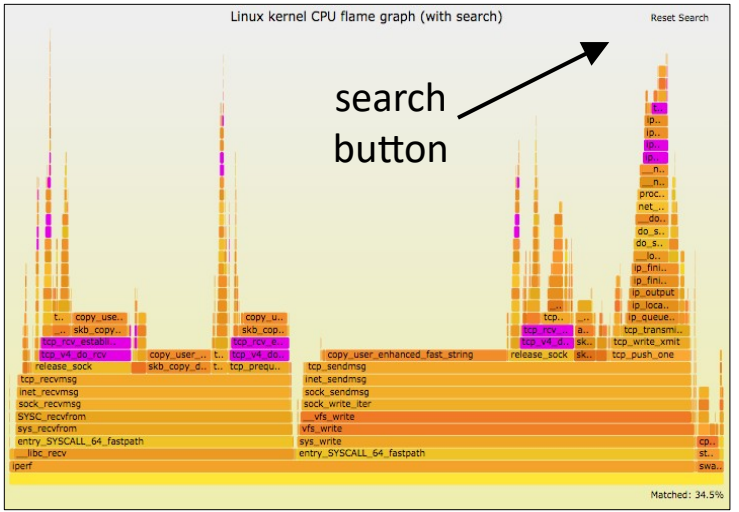
Flame Graph Interactivity

Essentials:

- Mouse-over for frame info (tool tips, status bar)
- Click to zoom
- Search (Ctrl-F or button)

Nice to have:

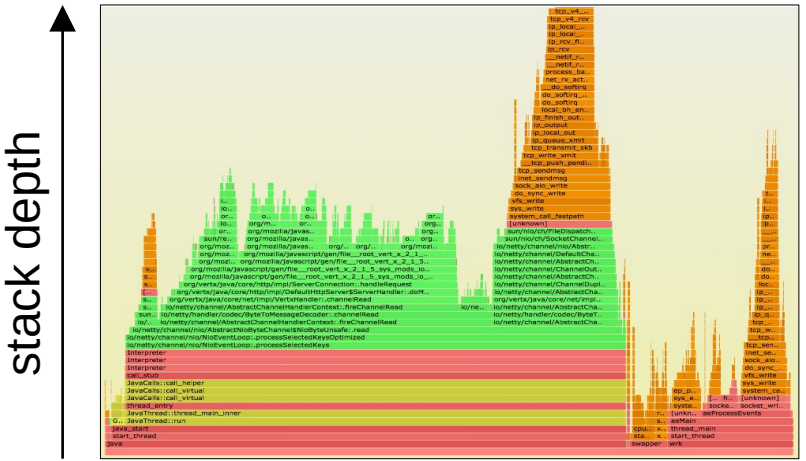
- Merge control: root, leaf, middle
- Y-axis direction: flame or icicle
- Flame chart toggle
- Canned searches
- Collapse filters
- Code links

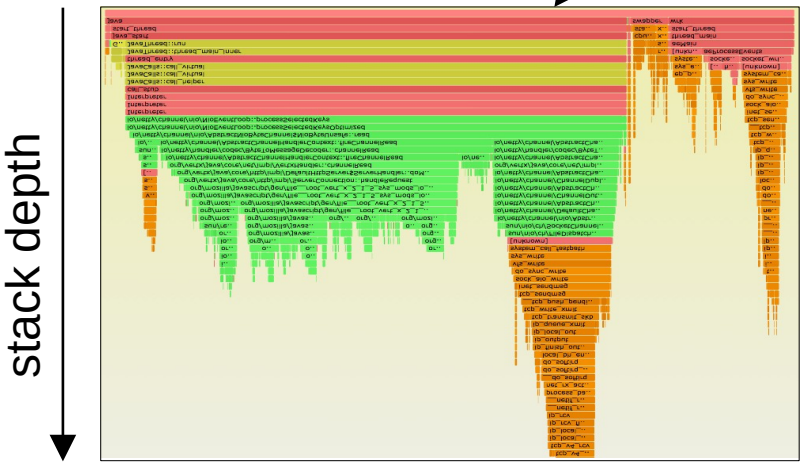


search matches in magenta

Which way up?

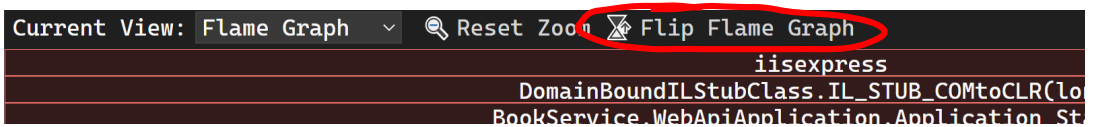
My original flamegraph.pl has `--inverted` for an “icicle graph”





Either way is fine!

- Icicle layout helps avoid scrolling when starting at the top
- Let the end-user choose



Source: https://learn.microsoft.com/en-us/visualstudio/profiling/flame-graph

Differential Flame Graphs

Hues:

- red == more samples
- blue == less samples

Intensity:

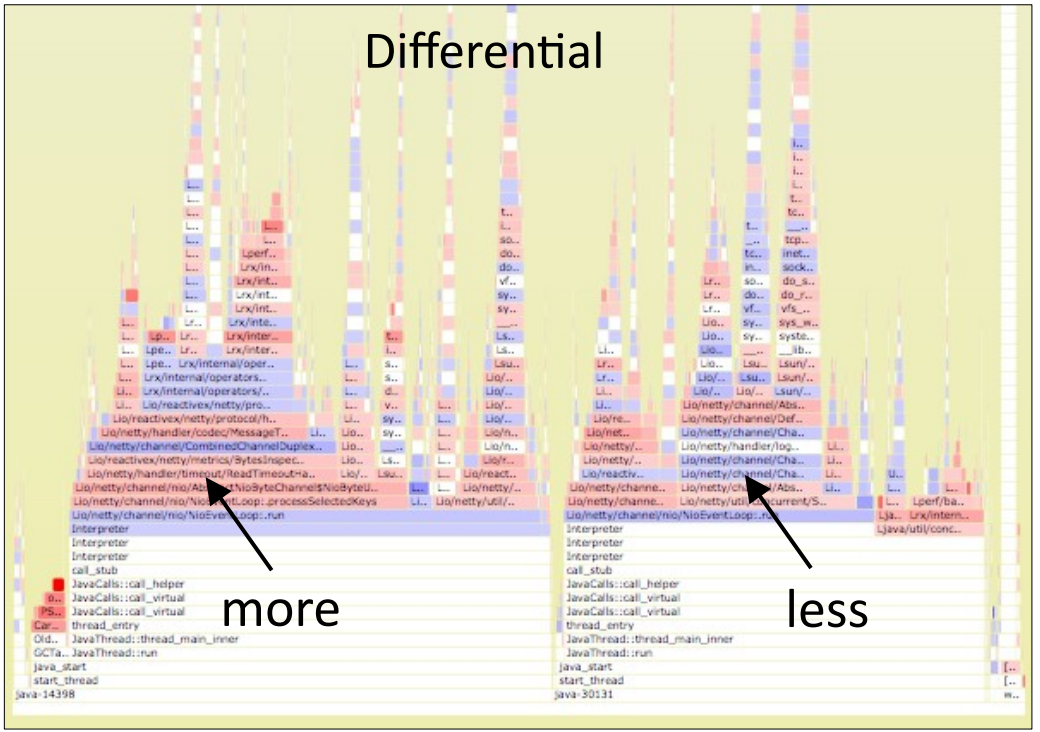
- Degree of difference

Other examples

- flamegraphdiff

This spectrum can show other metrics, like CPI.

Remember to show elided frames!



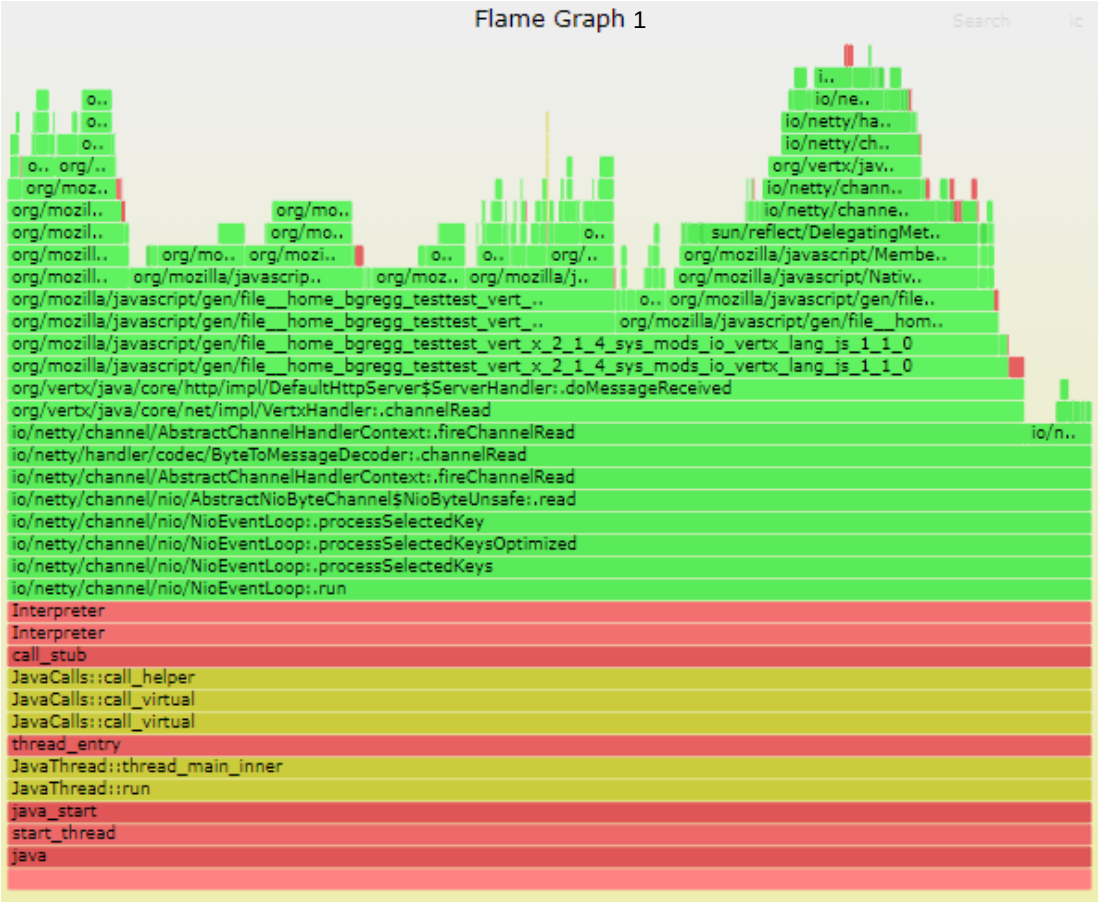
Poor Man's Differential Flame Graphs

Toggle between tabs in your browser

- Like searching for Pluto!

Or flip between slides

- Cue exciting demo!



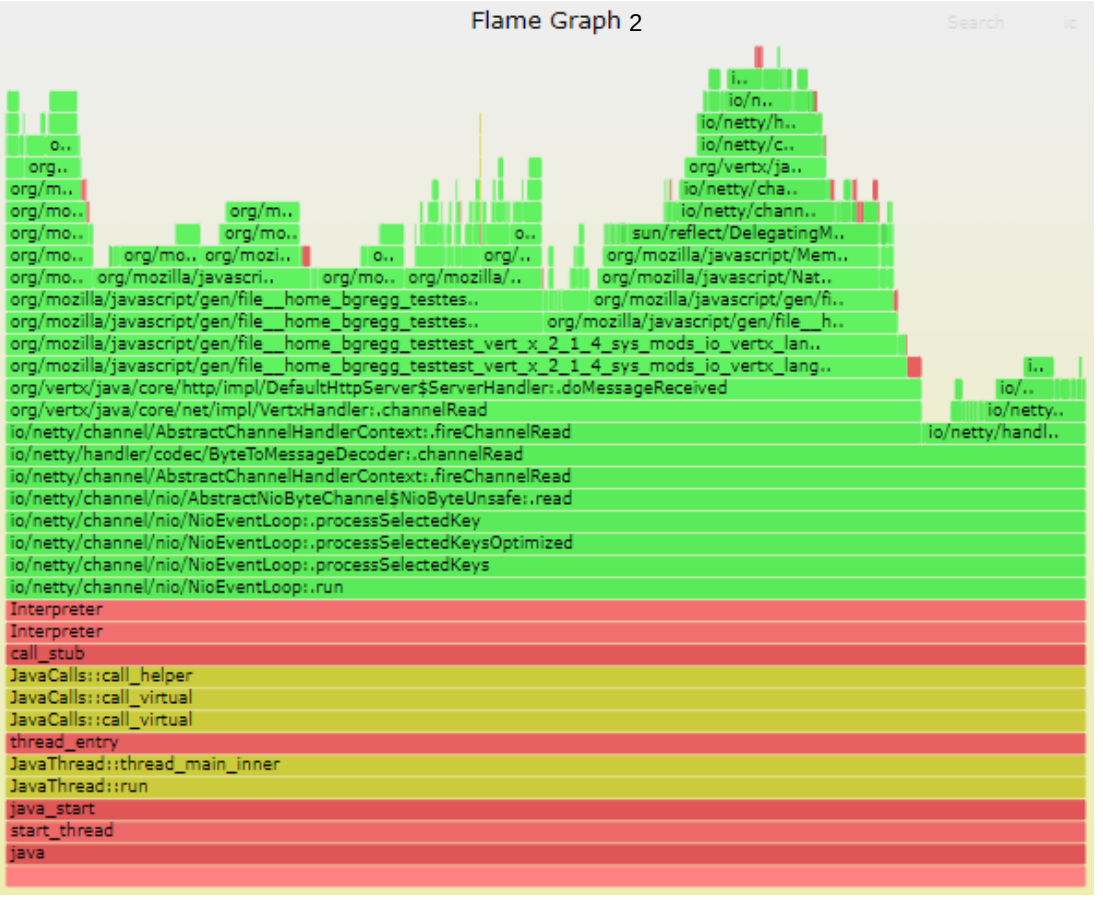
Poor Man's Differential Flame Graphs

Toggle between tabs in your browser

- Like searching for Pluto!

Or flip between slides

- Cue exciting demo!



System CPU Profilers

- Linux
  - perf_events (aka "perf")
  - bcc profile (eBPF-based)
- Windows
  - XPerf, WPA (now has flame graphs!)
- OS X
  - Instruments
- And many others…

Tip: use system profilers whenever possible. Runtime profilers (e.g., Java JVMTI-based) are user space and typically don't include kernel CPU time or kernel stacks.

Linux CPU Flame Graphs

Linux 5.8+ via perf for *simplicity* (2020):

```
perf script flamegraph -F 49 -a -- sleep 30
```

- Generates flamegraph.html. One command! Thanks Andreas Gerstmayr.

Linux 4.9+ via eBPF for *efficiency* (2016):

```
apt-get install bpfcc-tools
git clone https://github.com/brendangregg/FlameGraph
profile-bcc.py -dF 49 30 | ./FlameGraph/flamegraph.pl > perf.svg
```

- eBPF (no longer an acronym) is the name of an in-kernel execution environment, used in this case for aggregating stack samples in kernel context
- Most efficient: no perf.data file, summarizes in-kernel

Some runtimes (e.g., JVM) require extra steps for stacks & symbols (next section)

Older Linux CPU Flame Graphs

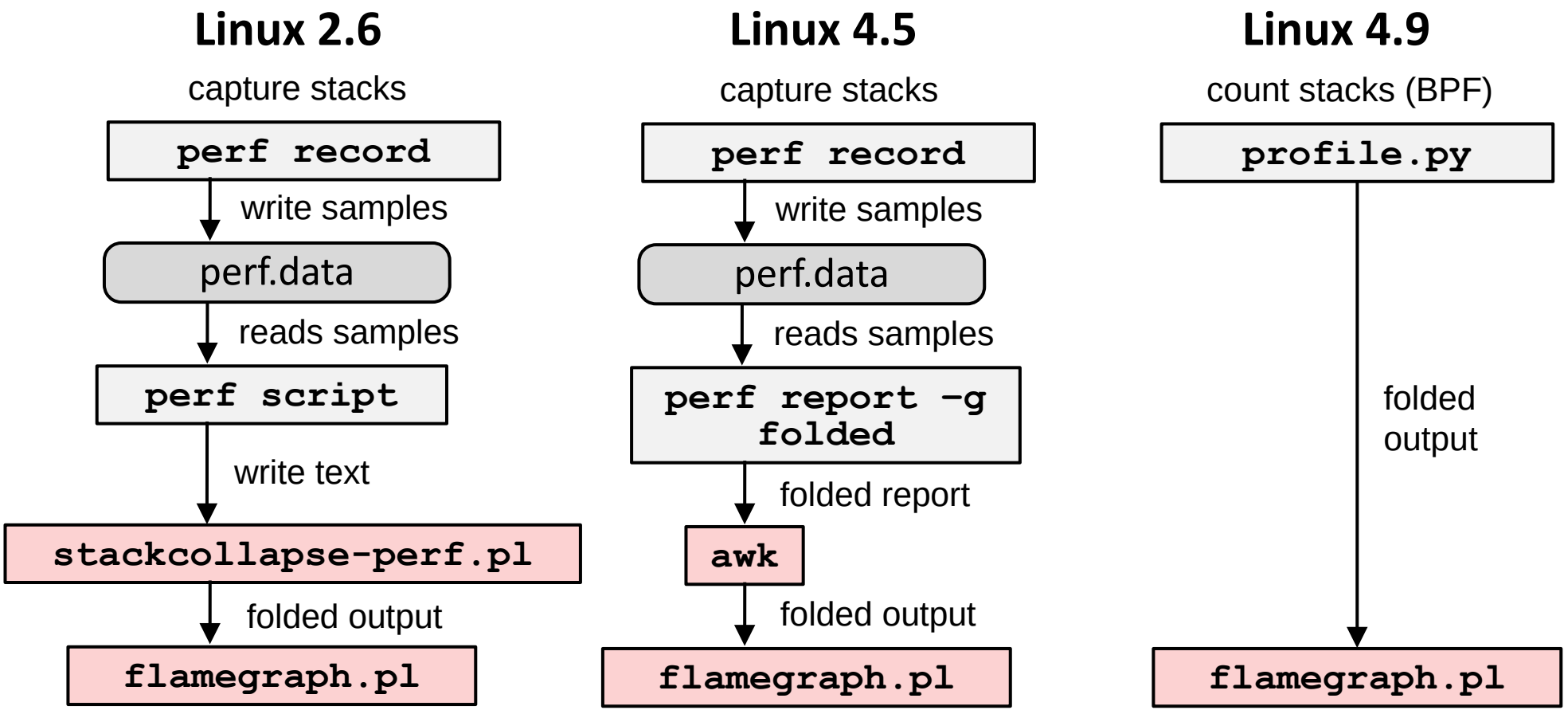
Linux 2.6+ via perf.data and perf script (2009):

```
git clone https://github.com/brendangregg/FlameGraph; cd FlameGraph
perf record -F 49 -a -g -- sleep 30
perf script | ./stackcollapse-perf.pl |./flamegraph.pl > perf.svg
```

Linux 4.5 can use folded output (2016):

- Skips the CPU-costly stackcollapse-perf.pl step; see:
  http://www.brendangregg.com/blog/2016-04-30/linux-perf-folded.html

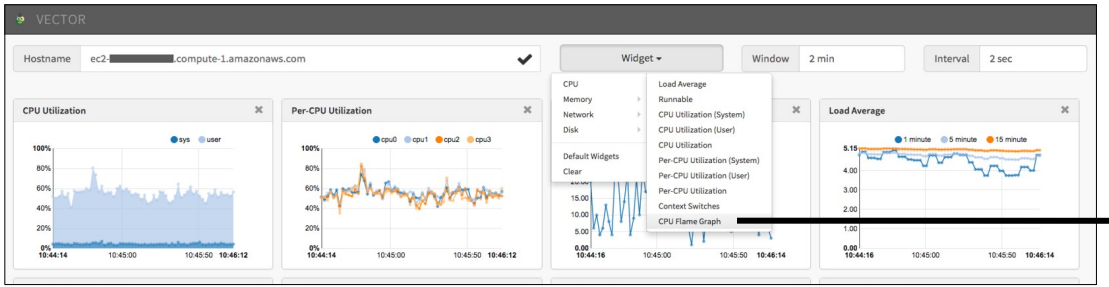
Linux Profiling Optimizations

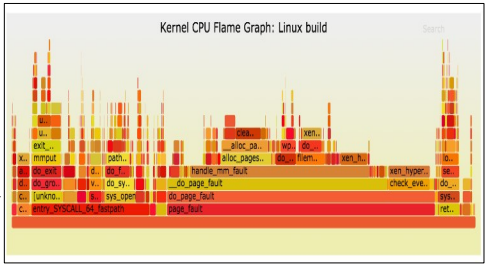


GUI Automation

There are many options nowadays. I've worked on five:

- **Netflix Vector** (now retired!):





- **Netflix FlameScope** (covered later)
- **Netflix FlameCommander** (continuous profiling; not open source yet)
- I'm now helping with **Intel vTune** and **Intel gProfiler**

Open source examples include **Granulate gProfiler**, **Eclipse TraceCompass**, **Grafana flame graphs**, **Firefox profiler**, and more (see implementation slides). Build your own!

Flame Charts (2013)

Inspired by flame graphs: https://bugs.webkit.org/show_bug.cgi?id=111162

Ilya Tikhonovsky 2013-03-01 04:33:26 PST Description

Flame Chart may give to the developer a better clue what is going on with the performance without expanding the entire tree.

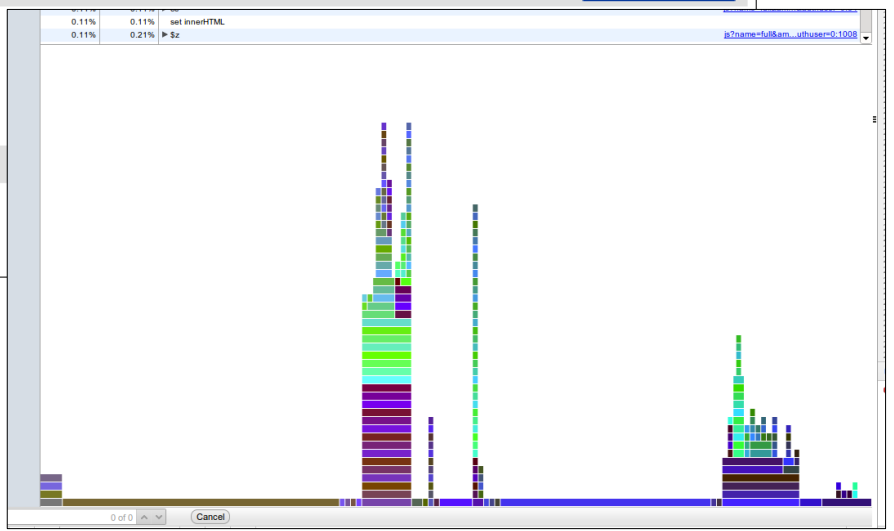
http://dtrace.org/blogs/brendan/2011/12/16/flame-graphs/

Ilya Tikhonovsky 2013-03-01 04:35:00 PST Comment 1

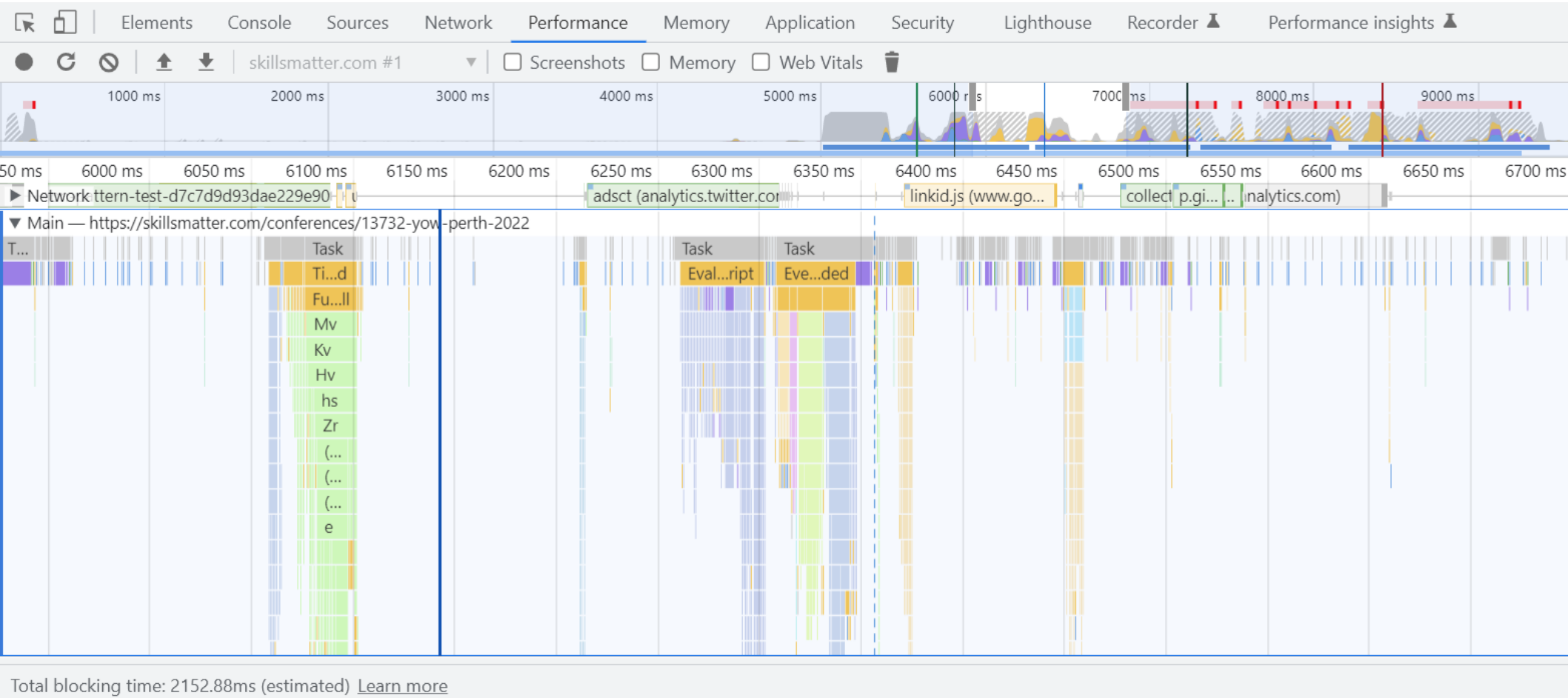
Created attachment 190933 [details]
screenshot

Ilya Tikhonovsky 2013-03-01 04:37:34 PST

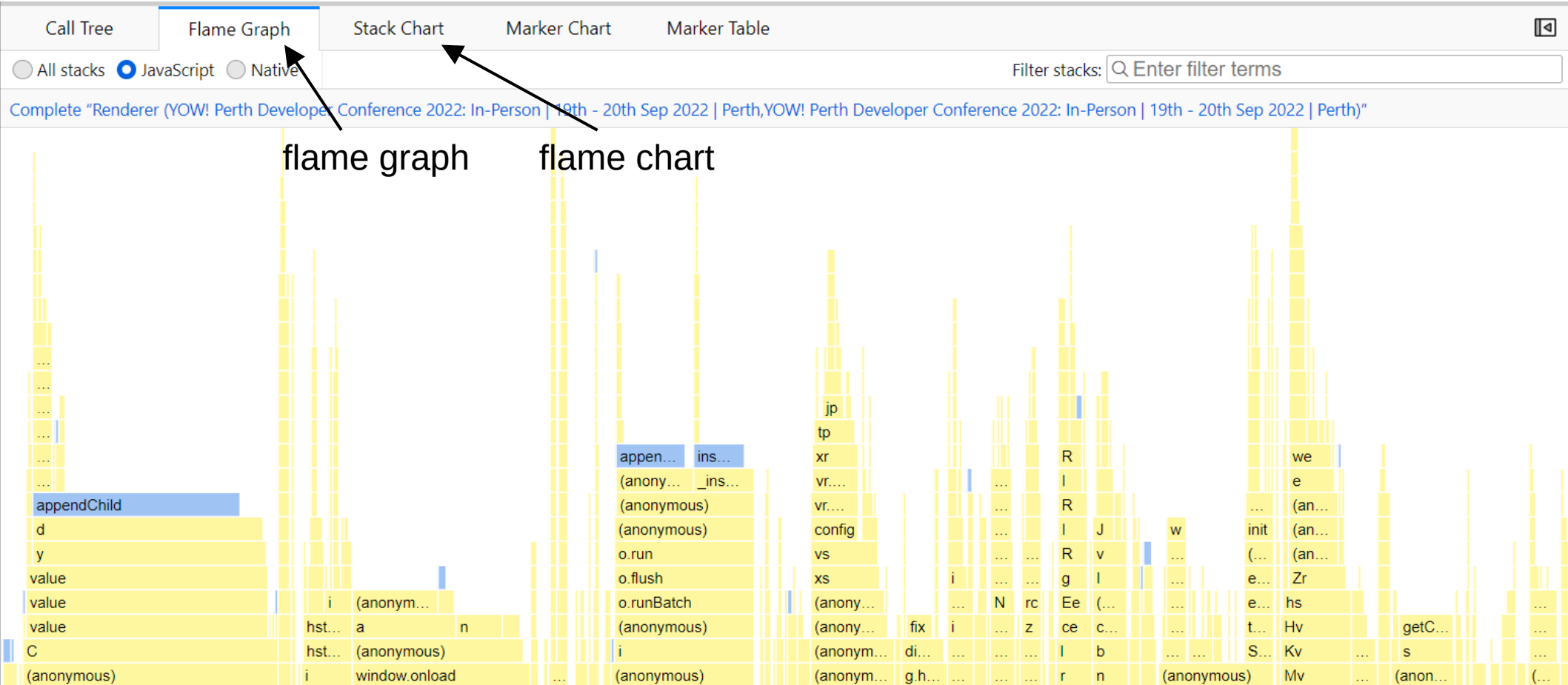
Created attachment 190934 [details]
Patch



Chrome DevTools Flame Charts (2022)



Firefox Profiler Flame Graph (2022)



Flame Charts

x-axis: time

Flame Graphs

x-axis: population

alphabet sort or another frame merging algorithm

3. STACKS AND SYMBOLS

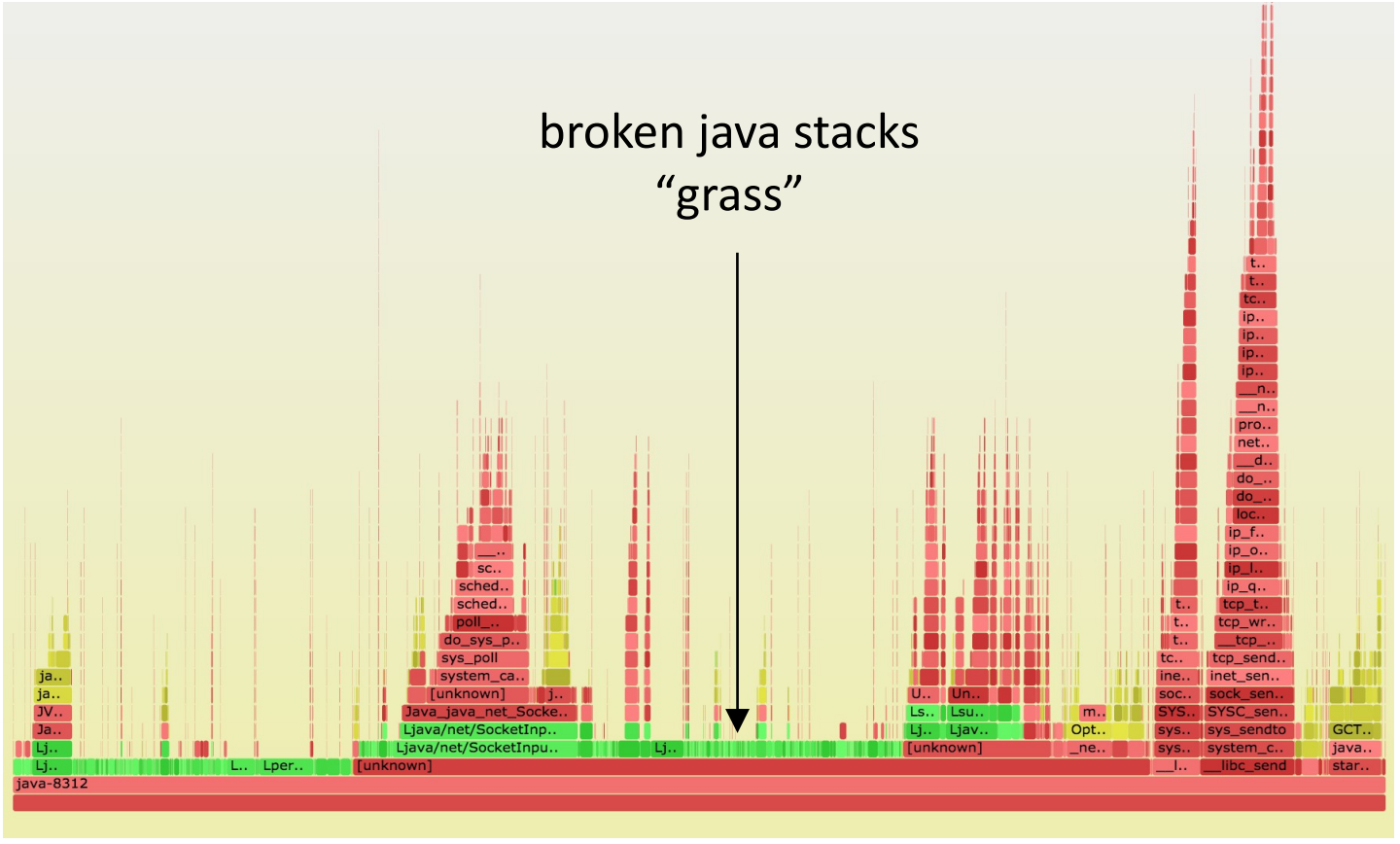
And Other Issues

Broken Stack Traces are Common

```
# perf record -F 99 -a -g - sleep 30
# perf script
[…]
java  4579 cpu-clock:
      7f417908c10b [unknown] (/tmp/perf-4458.map)

java  4579 cpu-clock:
      7f41792fc65f [unknown] (/tmp/perf-4458.map)
  a2d53351ff7da603 [unknown] ([unknown])
[…]
```

should probably have more frames

… as a Flame Graph



Fixing Stack Walking

A. Frame pointer-based
   - Fix by disabling that compiler optimization: gcc's -fno-omit-frame-pointer
   - Pros: simple, supported by many tools
   - Cons: might cost a little extra CPU (usually <1%)
B. Debug info (DWARF) walking
   - Cons: costs disk space, and not supported by all profilers, expensive for real-time tracing
C. JIT-provided runtime walkers
   - Pros: include more internals, such as inlined frames (e.g., JVMTI stacks)
   - Cons: limited to application internals: no kernel
D. Last branch record (LBR)
E. Add-on walkers (eBPF)

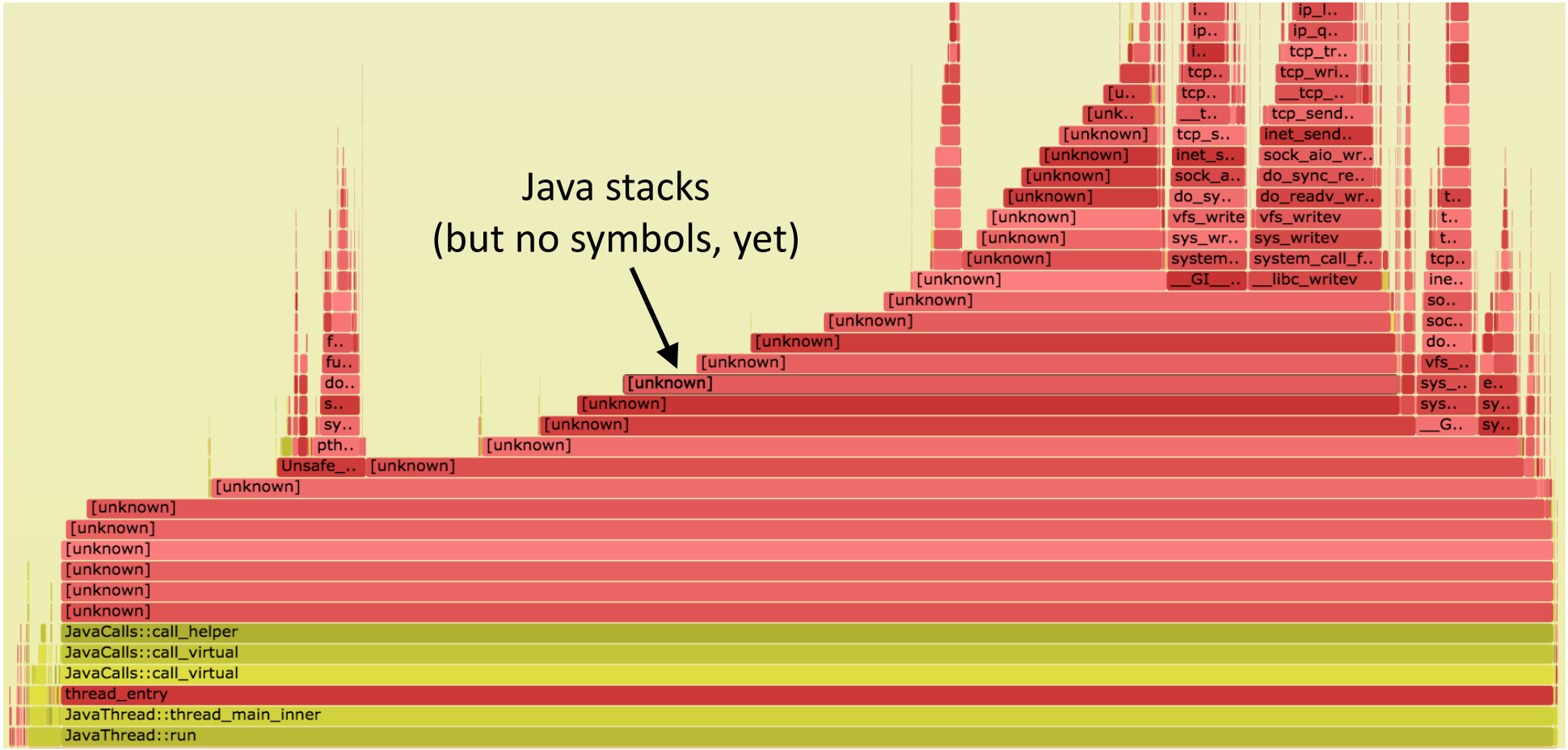
Fixing Java Stack Traces

```
# perf script
[…]
java  4579 cpu-clock:
      7f417908c10b [unknown] (/tmp/…

java  4579 cpu-clock:
      7f41792fc65f [unknown] (/tmp/…
  a2d53351ff7da603 [unknown] ([unkn…
[…]
```

I prototyped JVM frame pointers. Oracle rewrote it and included it in Java as **-XX:+PreserveFramePointer** (JDK 8 u60b19)

```
# perf script
[…]
java 8131 cpu-clock:
    7fff76f2dce1 [unknown] ([vdso])
    7fd3173f7a93 os::javaTimeMillis() (/usr/lib/jvm…
    7fd301861e46 [unknown] (/tmp/perf-8131.map)
    7fd30184def8 [unknown] (/tmp/perf-8131.map)
    7fd30174f544 [unknown] (/tmp/perf-8131.map)
    7fd30175d3a8 [unknown] (/tmp/perf-8131.map)
    7fd30166d51c [unknown] (/tmp/perf-8131.map)
    7fd301750f34 [unknown] (/tmp/perf-8131.map)
    7fd3016c2280 [unknown] (/tmp/perf-8131.map)
    7fd301b02ec0 [unknown] (/tmp/perf-8131.map)
    7fd3016f9888 [unknown] (/tmp/perf-8131.map)
    7fd3016ece04 [unknown] (/tmp/perf-8131.map)
    7fd30177783c [unknown] (/tmp/perf-8131.map)
    7fd301600aa8 [unknown] (/tmp/perf-8131.map)
    7fd301a4484c [unknown] (/tmp/perf-8131.map)
    7fd3010072e0 [unknown] (/tmp/perf-8131.map)
    7fd301007325 [unknown] (/tmp/perf-8131.map)
    7fd301007325 [unknown] (/tmp/perf-8131.map)
    7fd3010004e7 [unknown] (/tmp/perf-8131.map)
    7fd3171df76a JavaCalls::call_helper(JavaValue*,…
    7fd3171dce44 JavaCalls::call_virtual(JavaValue*…
    7fd3171dd43a JavaCalls::call_virtual(JavaValue*…
    7fd31721b6ce thread_entry(JavaThread*, Thread*)…
    7fd3175389e0 JavaThread::thread_main_inner() (/…
    7fd317538cb2 JavaThread::run() (/usr/lib/jvm/nf…
    7fd3173f6f52 java_start(Thread*) (/usr/lib/jvm/…
    7fd317a7e182 start_thread (/lib/x86_64-linux-gn…
```

Fixed Stacks Flame Graph



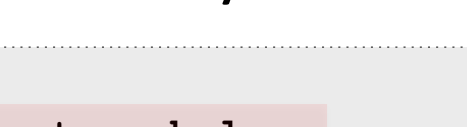
Fixing Native Symbols

A. Add a -dbgsym package, if available
B. Recompile from source

Fixing JIT Symbols (Java, Node.js, …)

Just-in-time runtimes don't have a pre-compiled symbol table

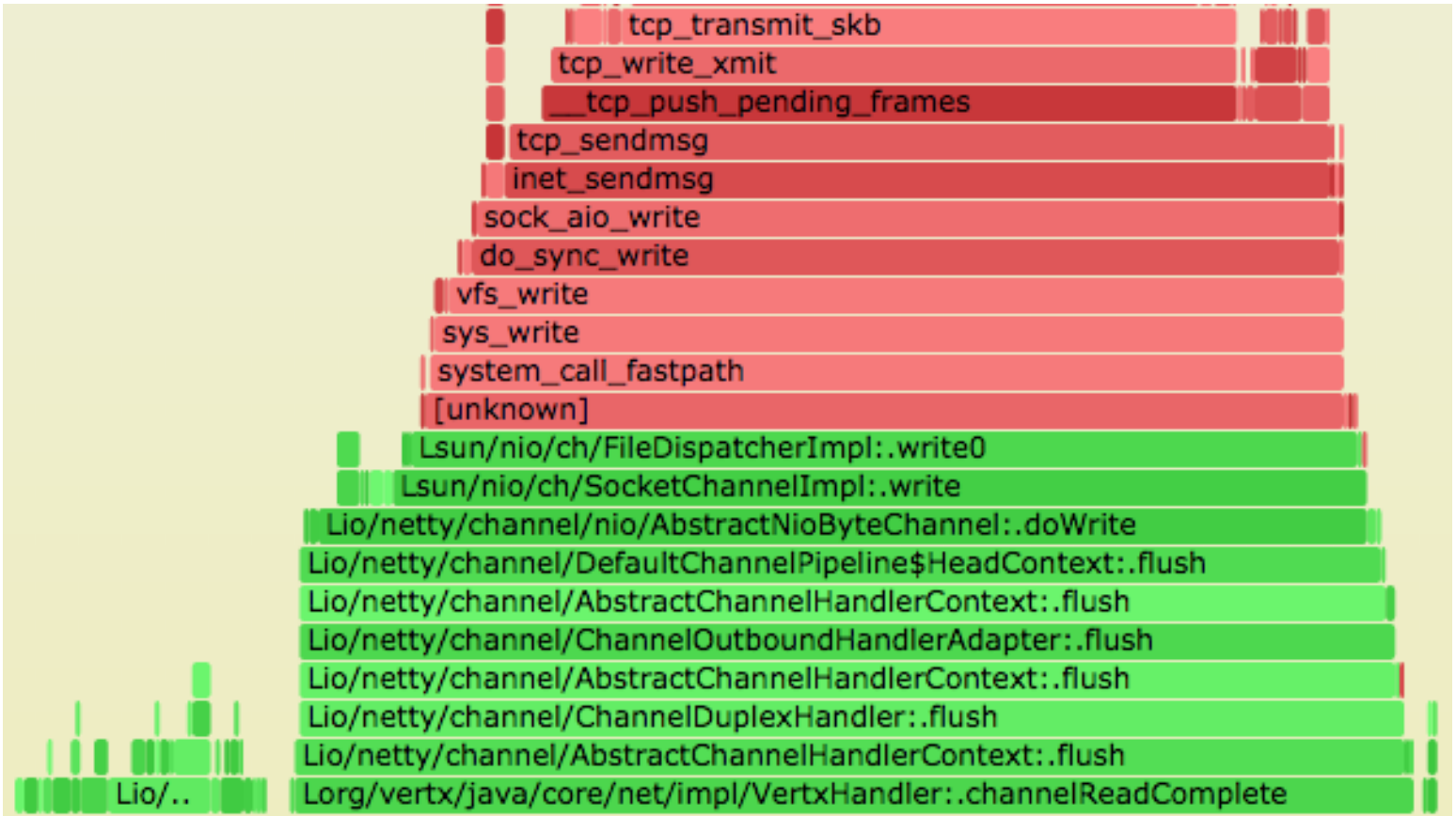
So Linux perf looks for an externally provided symbol file



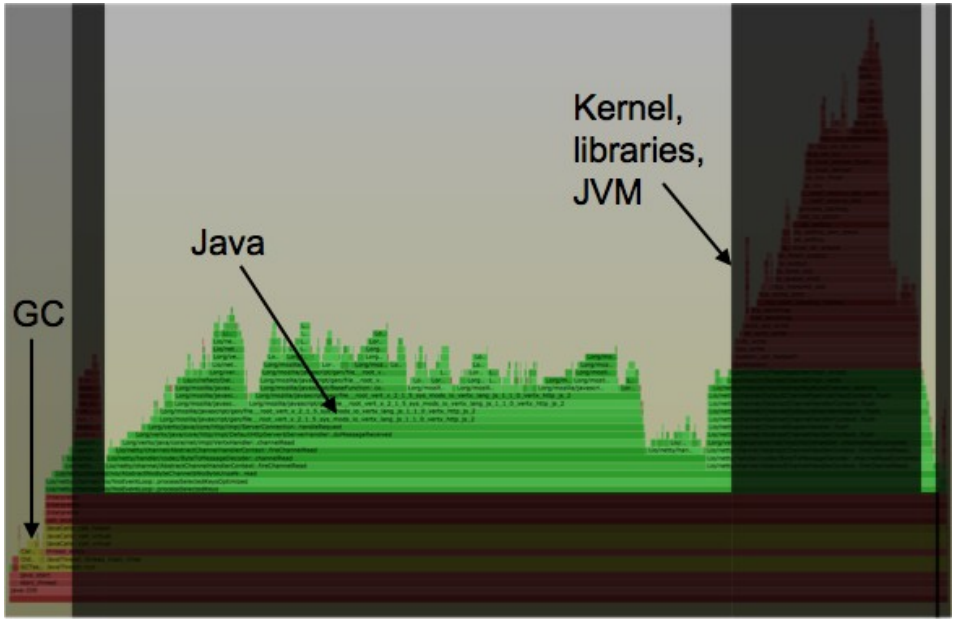
```
# perf script
Failed to open /tmp/perf-8131.map, continuing without symbols
[…]
java 8131 cpu-clock:
    7fff76f2dce1 [unknown] ([vdso])
    7fd3173f7a93 os::javaTimeMillis() (/usr/lib/jvm…
    7fd301861e46 [unknown] (/tmp/perf-8131.map)
[…]
```

- This can be created by runtimes; e.g., Java's perf-map-agent
- Not the only solution; can also integrate with JIT-based walkers, or have an external symbol translator (perf script, or eBPF-based).

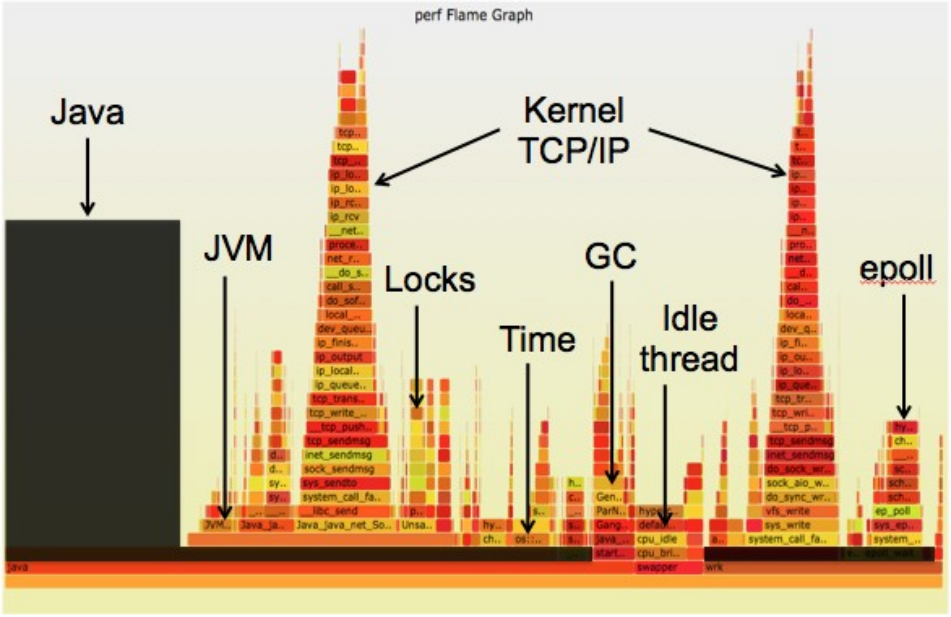
Fixed Symbols (zoom)



2014: Java Profiling (broken stacks)

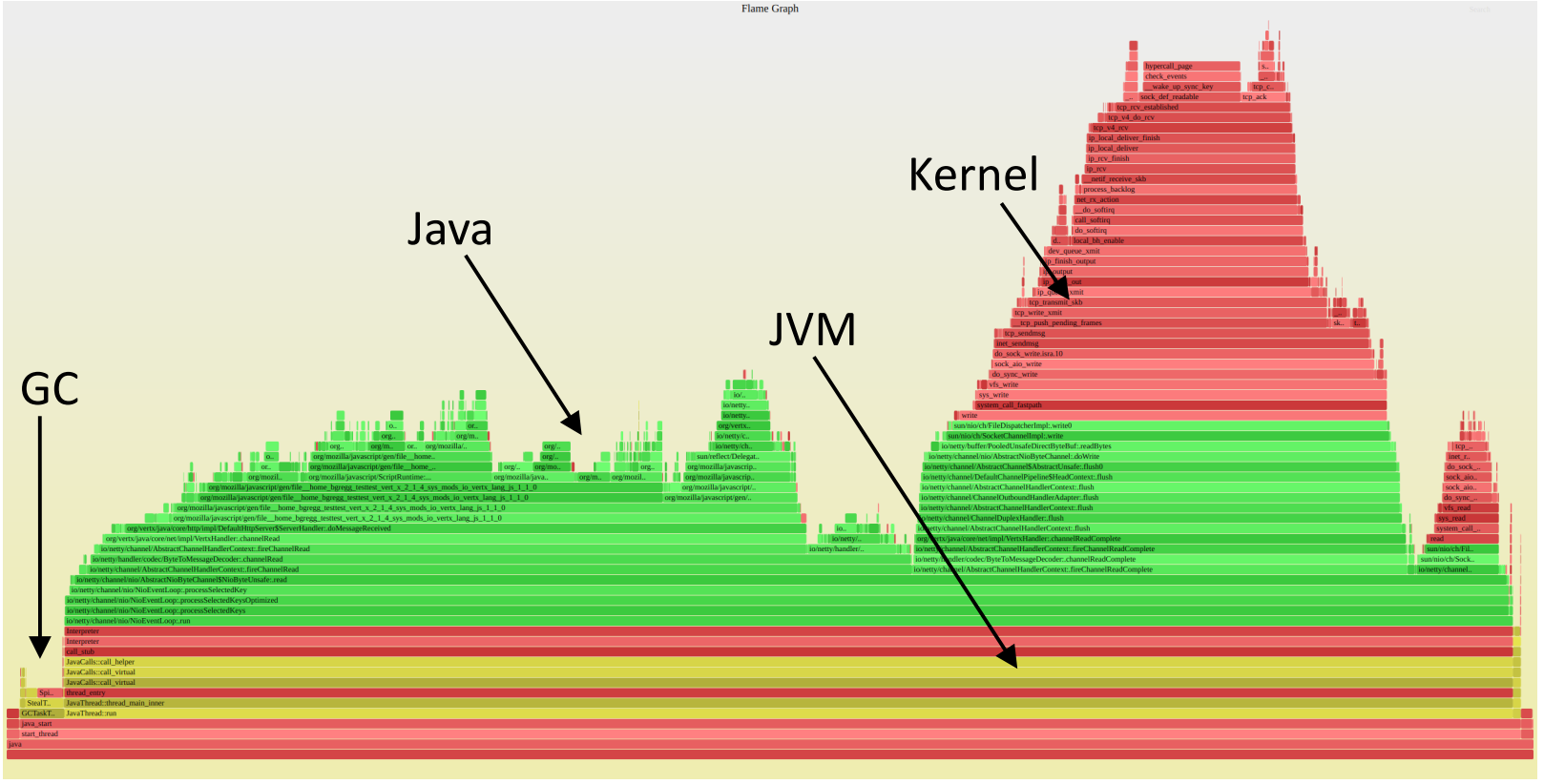


Java Profilers



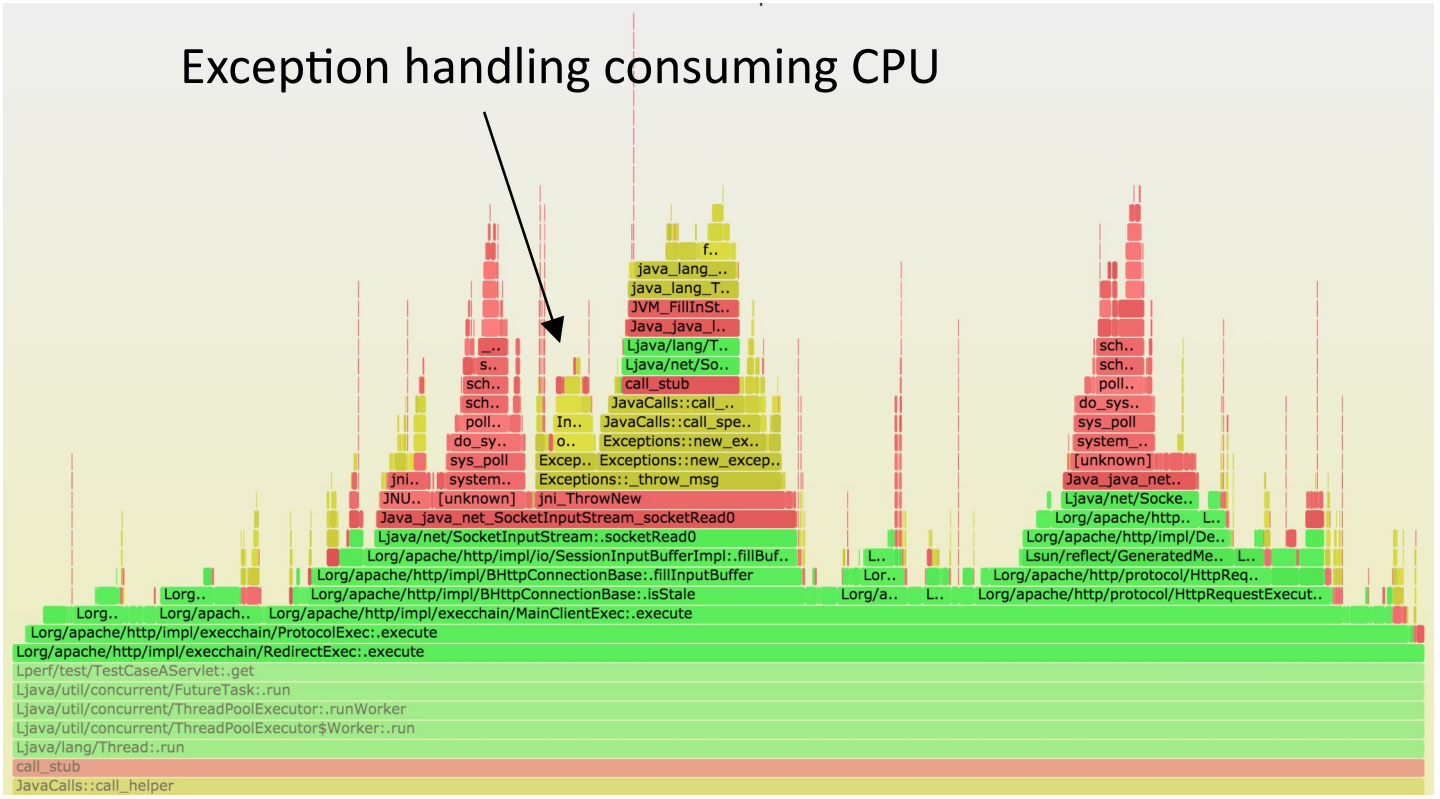
System Profilers

2018: Java Profiling (fixed stacks)



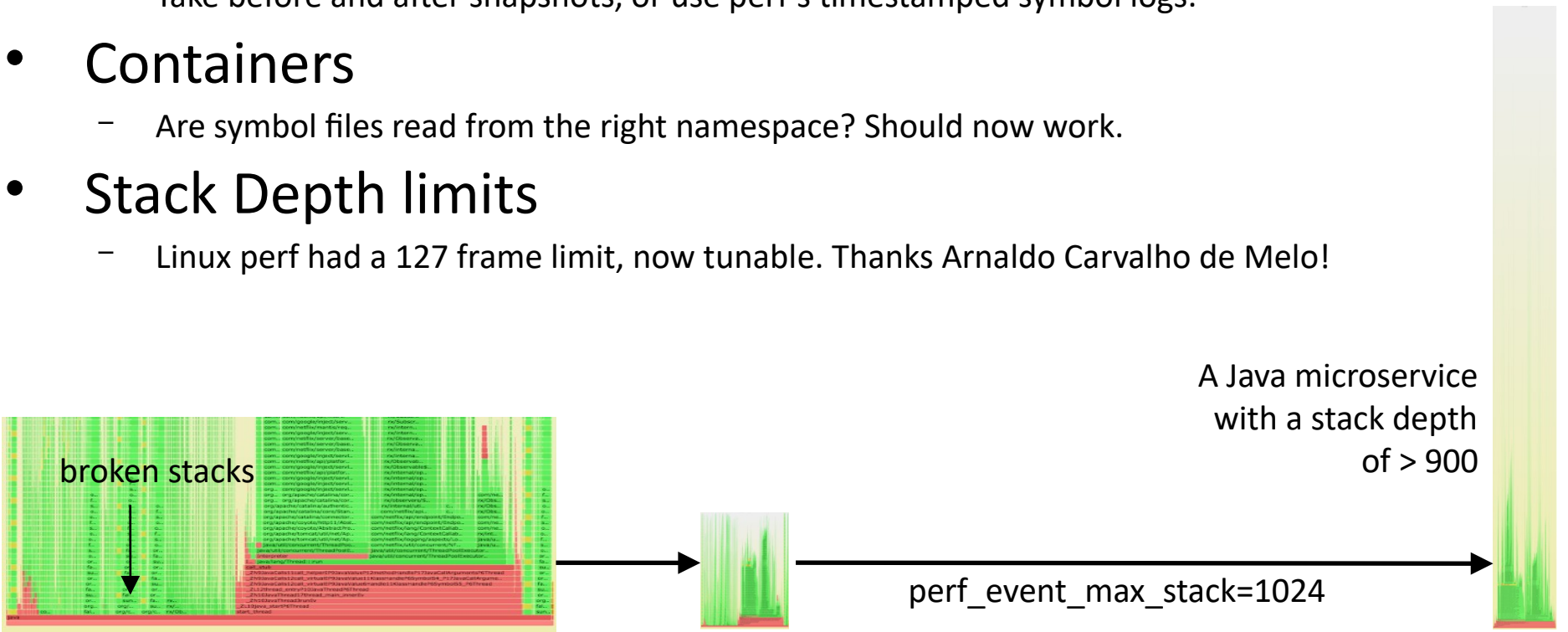
CPU Mixed-mode Flame Graph

Mixed-Mode Case Study



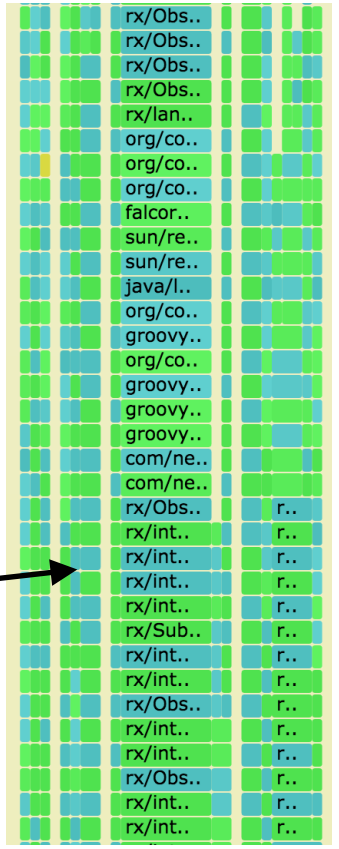
Other Issues

- JIT Symbol churn
  - Take before and after snapshots, or use perf's timestamped symbol logs.
- Containers
  - Are symbol files read from the right namespace? Should now work.
- Stack Depth limits
  - Linux perf had a 127 frame limit, now tunable. Thanks Arnaldo Carvalho de Melo!



Inlining

- Many frames may be missing (inlined)
  - Flame graph may still make enough sense
- Inlining can often be be tuned
  - e.g. Java's -XX:-Inline to disable, but can be 80% slower
  - Java's -XX:MaxInlineSize and -XX:InlineSmallCode can be tuned a little to reveal more frames: can even improve performance!
- Runtimes can un-inline on demand
  - So that exception stack traces make sense
  - e.g. Java's perf-map-agent can un-inline (unfoldall option)



Language/Runtime Issues

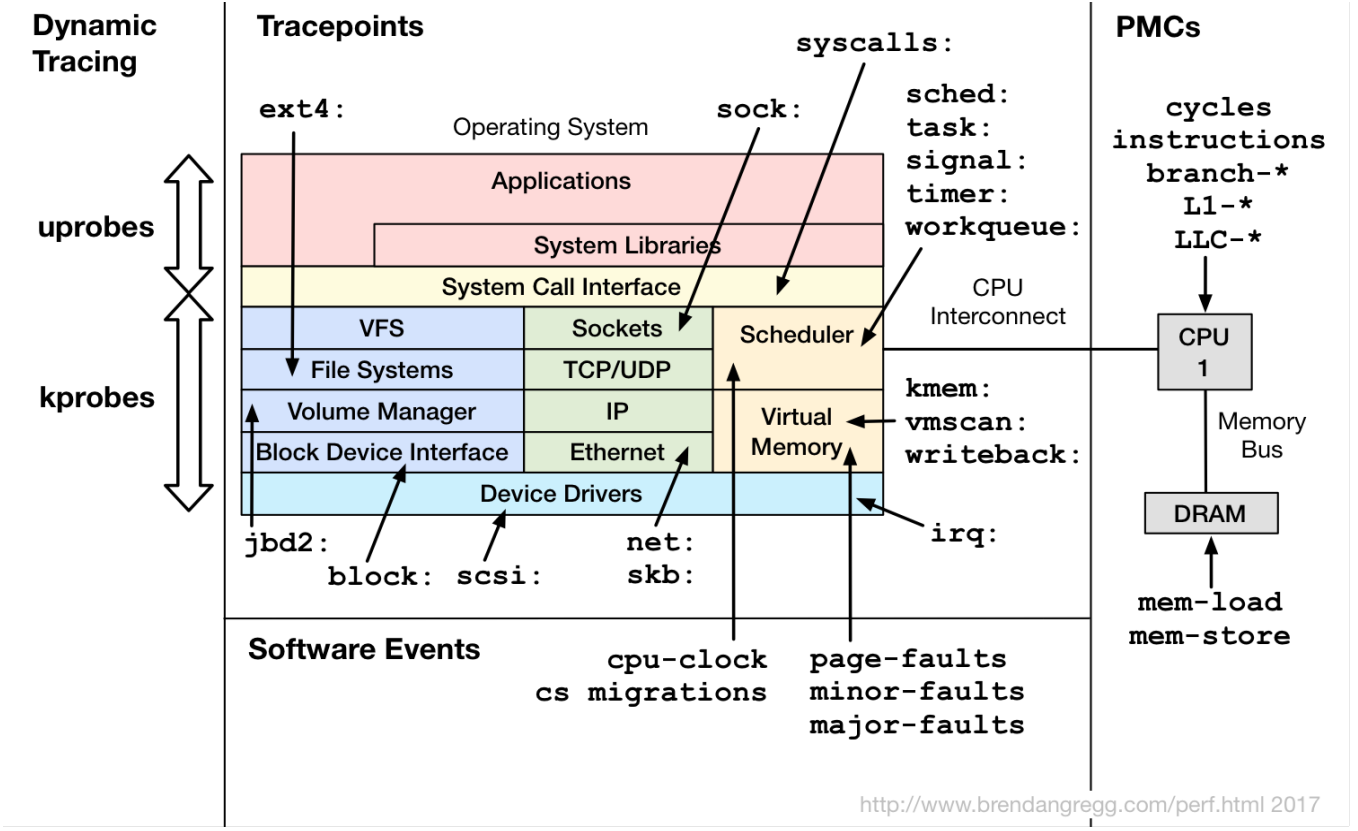
Each may have special stack/symbol instructions

- **Java**, **Node.js**, **Python**, **Ruby**, **C++**, **Go**, …
- See: https://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
- Check if flame graphs are already in the “official” profiler
- Try an Internet search

4. ADVANCED FLAME GRAPHS

Flame graphs can visualize any stack trace collection

On Linux, stacks from any of these events:



Page Faults

Show what triggered main memory (resident) to grow:

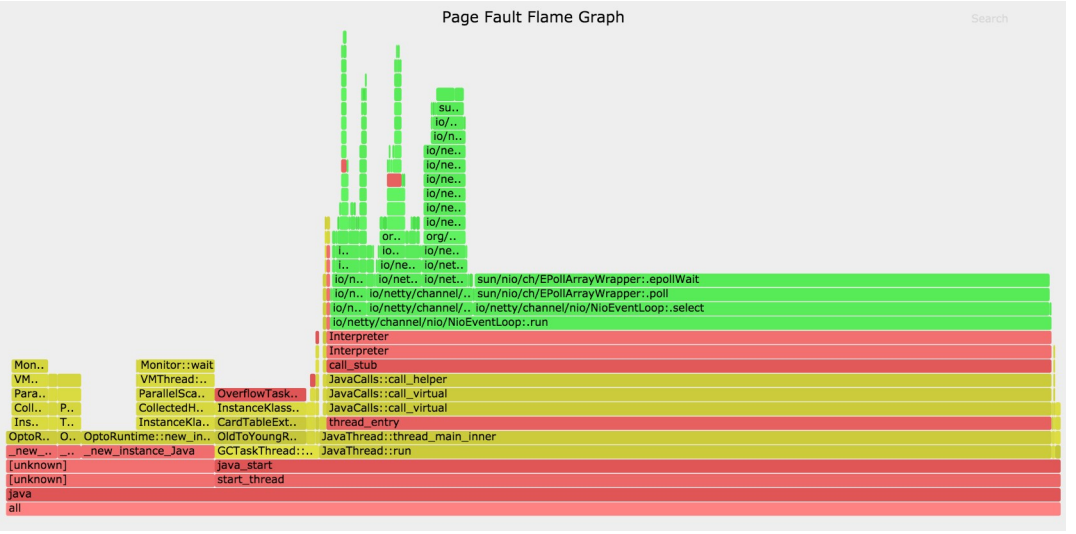
```
# perf record -e page-faults -p PID -g -- sleep 120
```

- "fault" as (physical) main memory is allocated on-demand, when a virtual page is first populated
- **Low overhead tool** to solve some types of memory leak

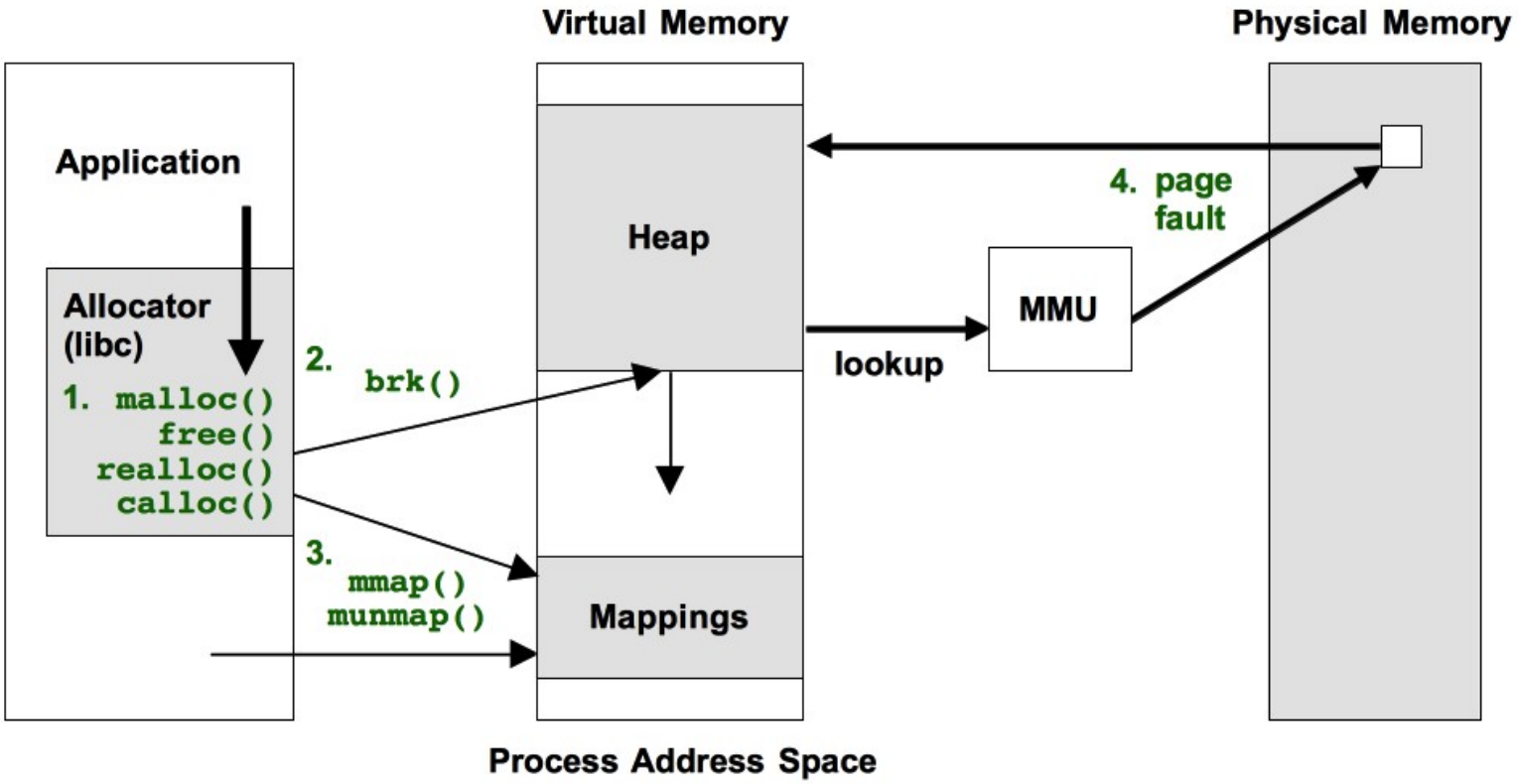
RES column in top(1)

```
   VIRT    RES  COMMAND
3972756 376876  java
 344752 231344  ab
      0      0  kworker/1:1
1069716  44032  evolution-calen
      0      0  ksoftirqd/2
```

grows because



Other Memory Sources

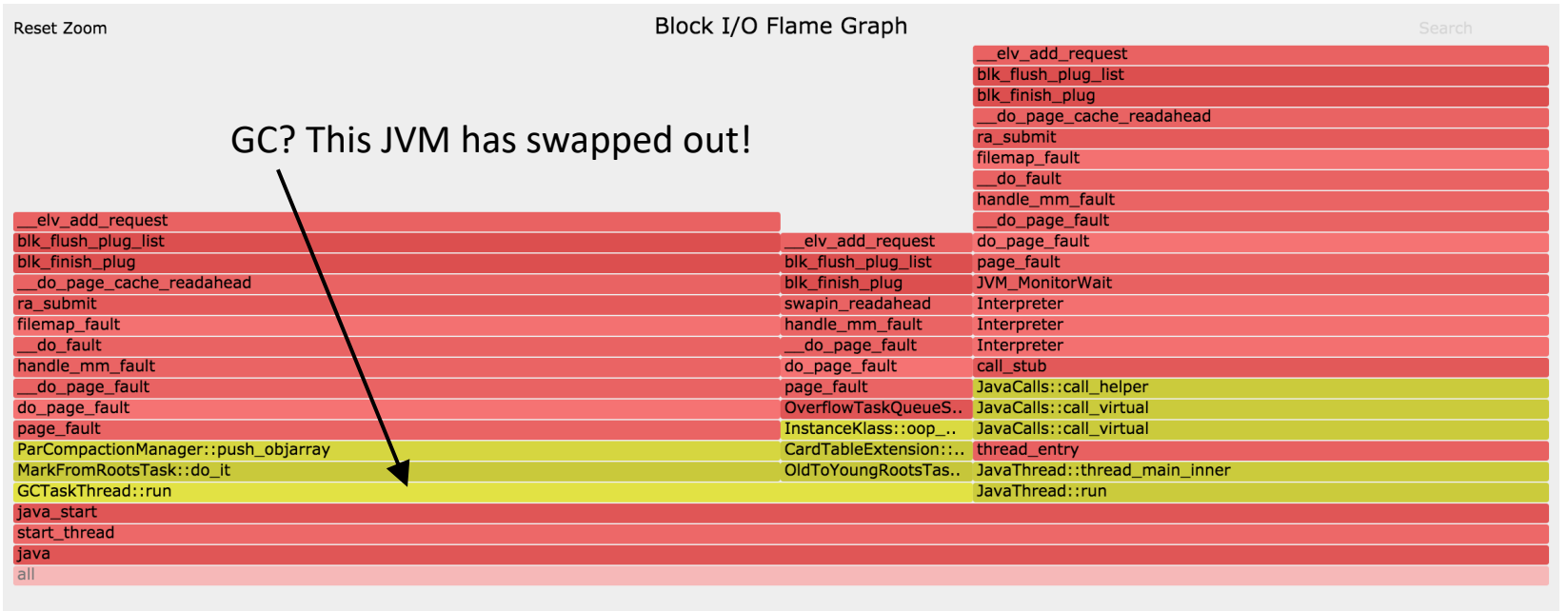


http://www.brendangregg.com/FlameGraphs/memoryflamegraphs.html

Disk I/O Requests

Shows who issued disk I/O (sync reads & writes):

```
# perf record -e block:block_rq_insert -a -g -- sleep 60
```



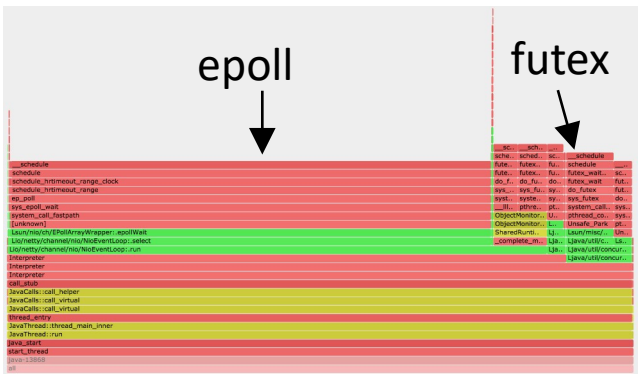
Context Switches

Show why Java blocked and stopped running on-CPU:

```
# perf record -e context-switches -p PID -g -- sleep 5
```

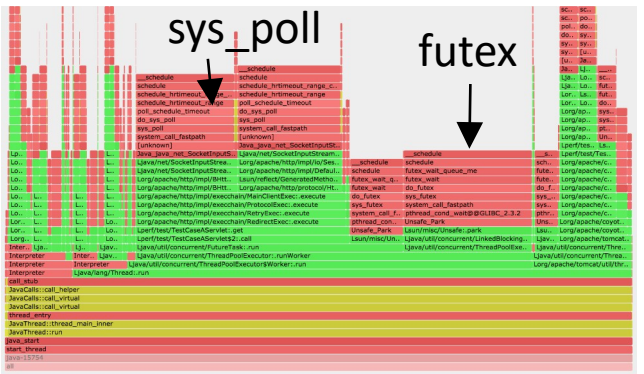
- Identifies locks, I/O, sleeps
- If code path shouldn't block and looks random, it's an involuntary context switch. I often filter these, but I've usually already solved this type of issue (CPU load) long before trying advanced flame graphs.

E.g., analyzing framework differences:



rxNetty

vs



Tomcat

TCP Events

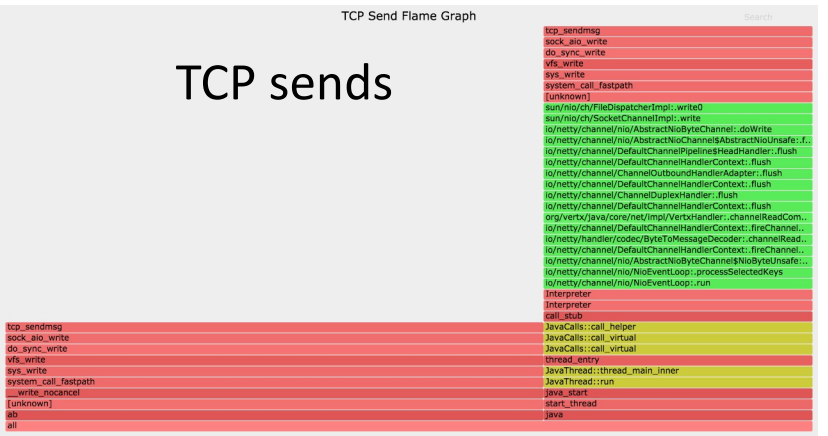
TCP transmit, using eBPF:

```
# bpftrace -e 'kprobe:tcp_sendmsg { @[kstack, ustack] = count(); }'
```

- For eBPF, can cost **noticeable overhead** for high packet rates (test and measure)
- For perf, can have *prohibitive overhead* due to the trace, dump, post-process cycle
- Note that **TCP receive is async**, so stack traces are meaningless. Trace socket read instead.

Can also trace TCP connect, accept

- Lower frequency, therefore lower overhead



CPU Cache Misses

In this example, sampling via Last Level Cache loads:

```
# perf record -e LLC-loads -c 10000 -a -g -- sleep 5; jmaps
# perf script -f comm,pid,tid,cpu,time,event,ip,sym,dso > out.stacks
```

- -c is the count (samples once per count)
- Can also sample hits, misses, stalls

Needs PEBS for IP accuracy

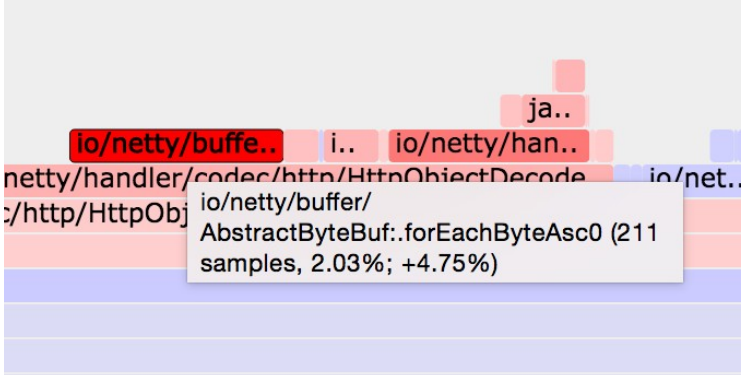
- Precise Event Based Sampling for Instruction Pointer accuracy. Not yet enabled in AWS EC2 VMs.

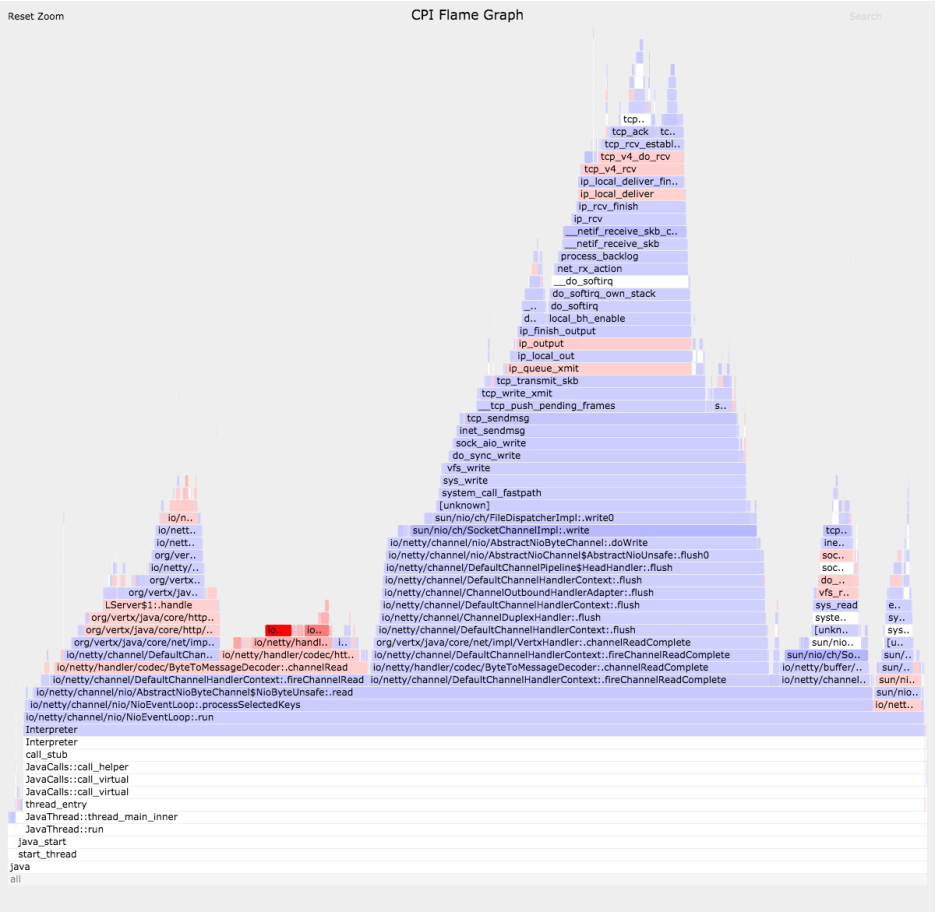
CPI Flame Graph

Cycles Per Instruction (CPI)

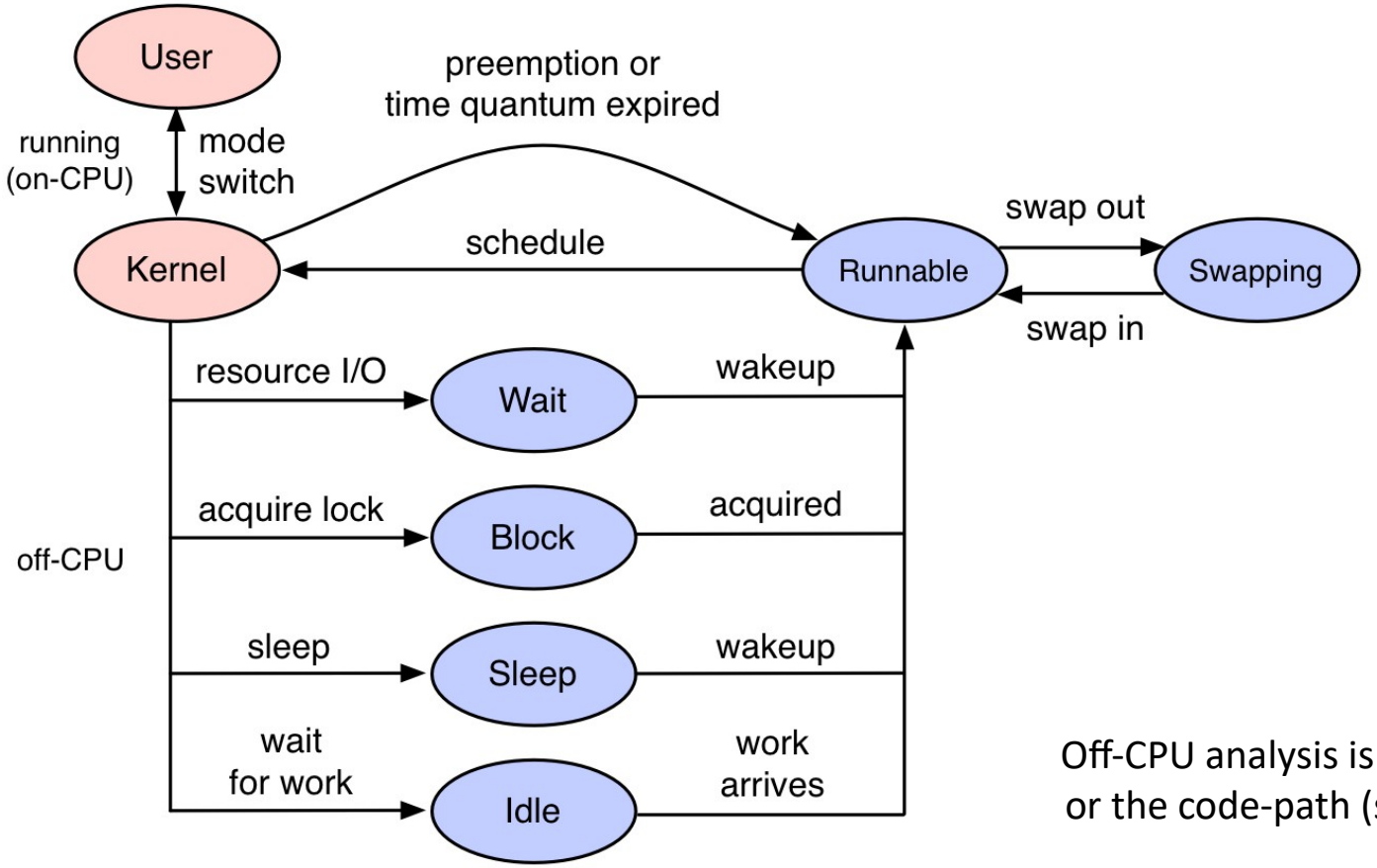
- red == instruction heavy
- blue == cycle heavy
  (likely memory stall cycles)

zoomed:



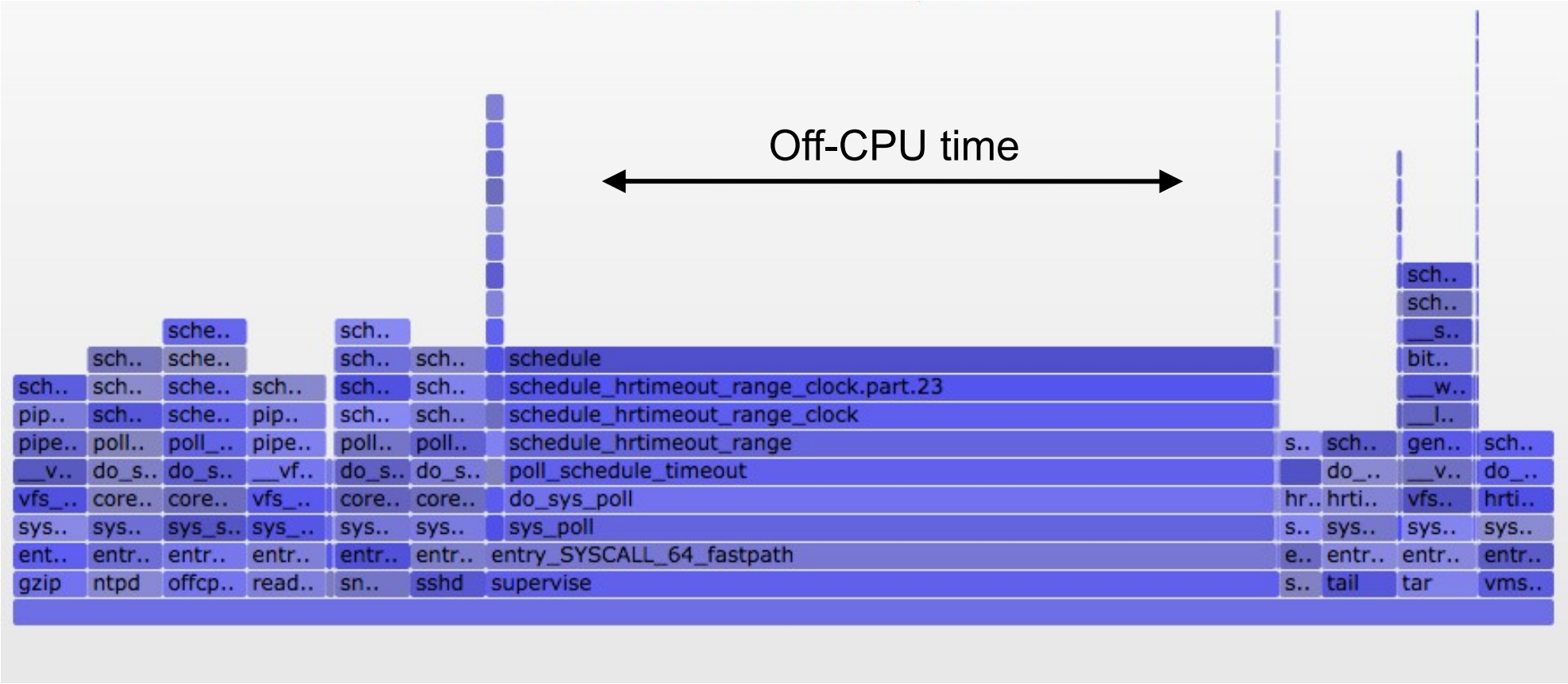


Off-CPU Analysis



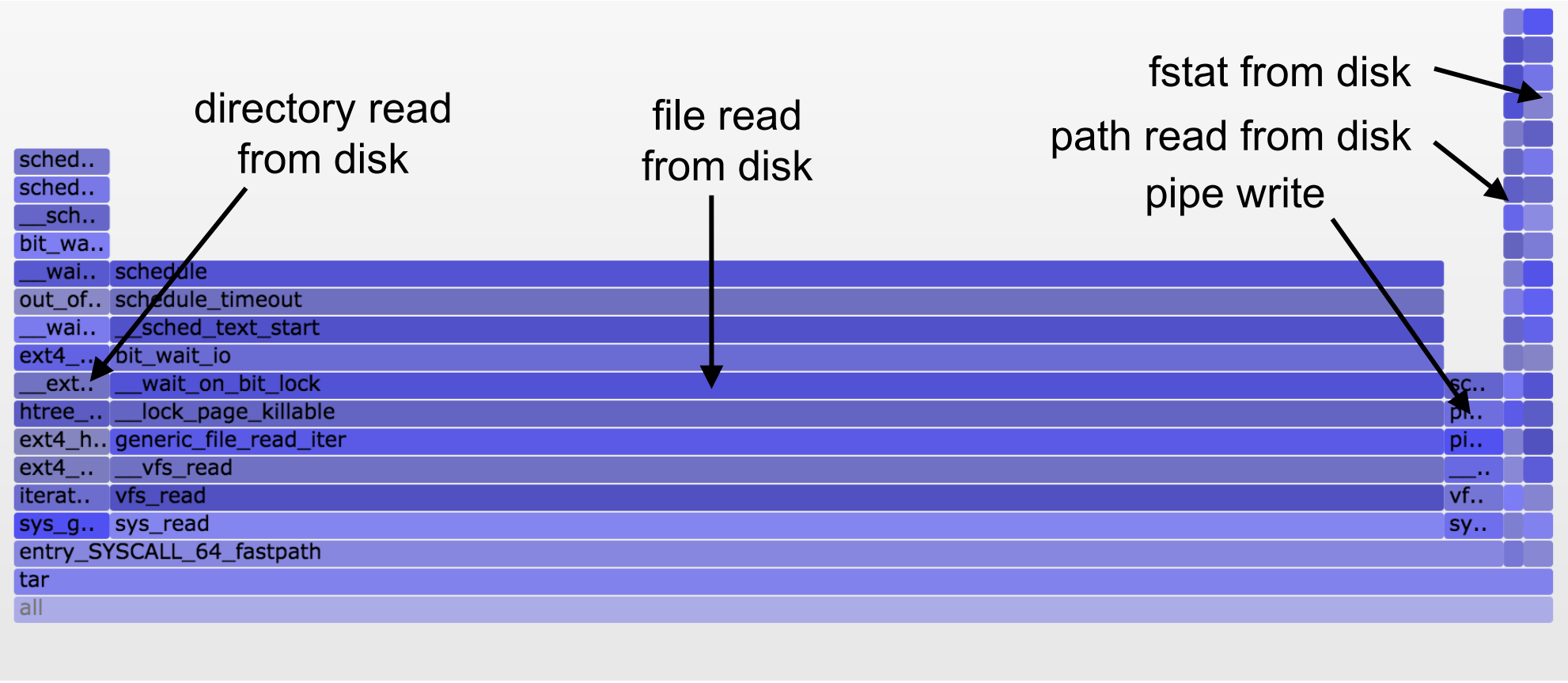
Off-CPU analysis is the study of blocking states, or the code-path (stack trace) that led to them

Off-CPU Time Flame Graph



More info http://www.brendangregg.com/blog/2016-02-01/linux-wakeup-offwake-profiling.html

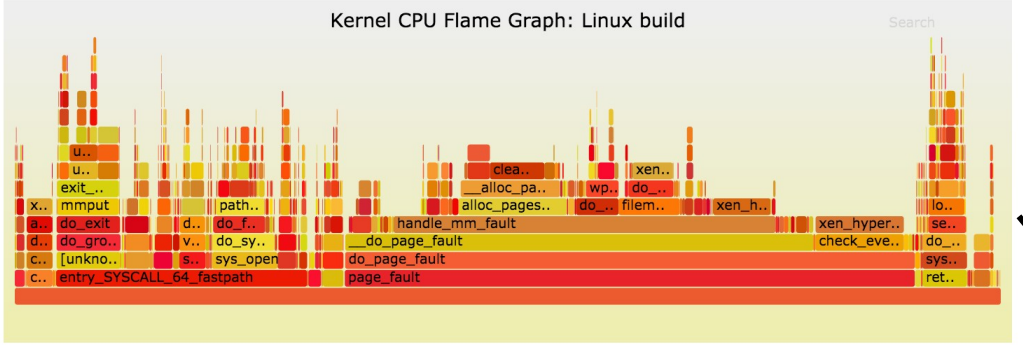
Off-CPU Time (zoomed): tar(1)



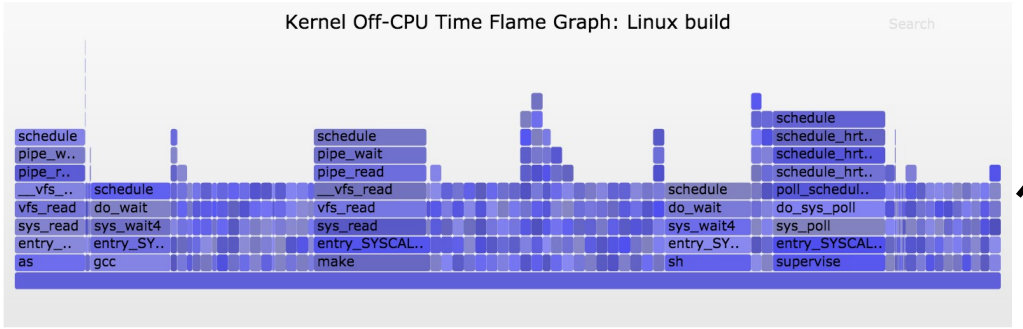
Only showing kernel stacks in this example

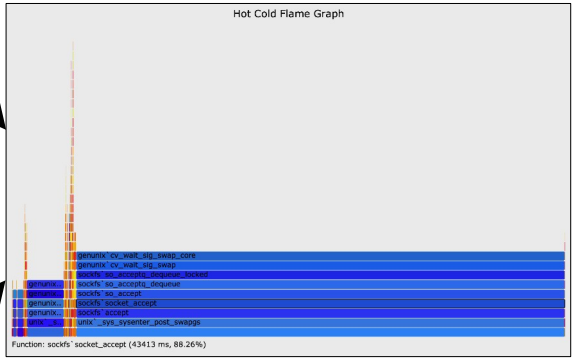
CPU + Off-CPU Flame Graphs: See Everything

CPU



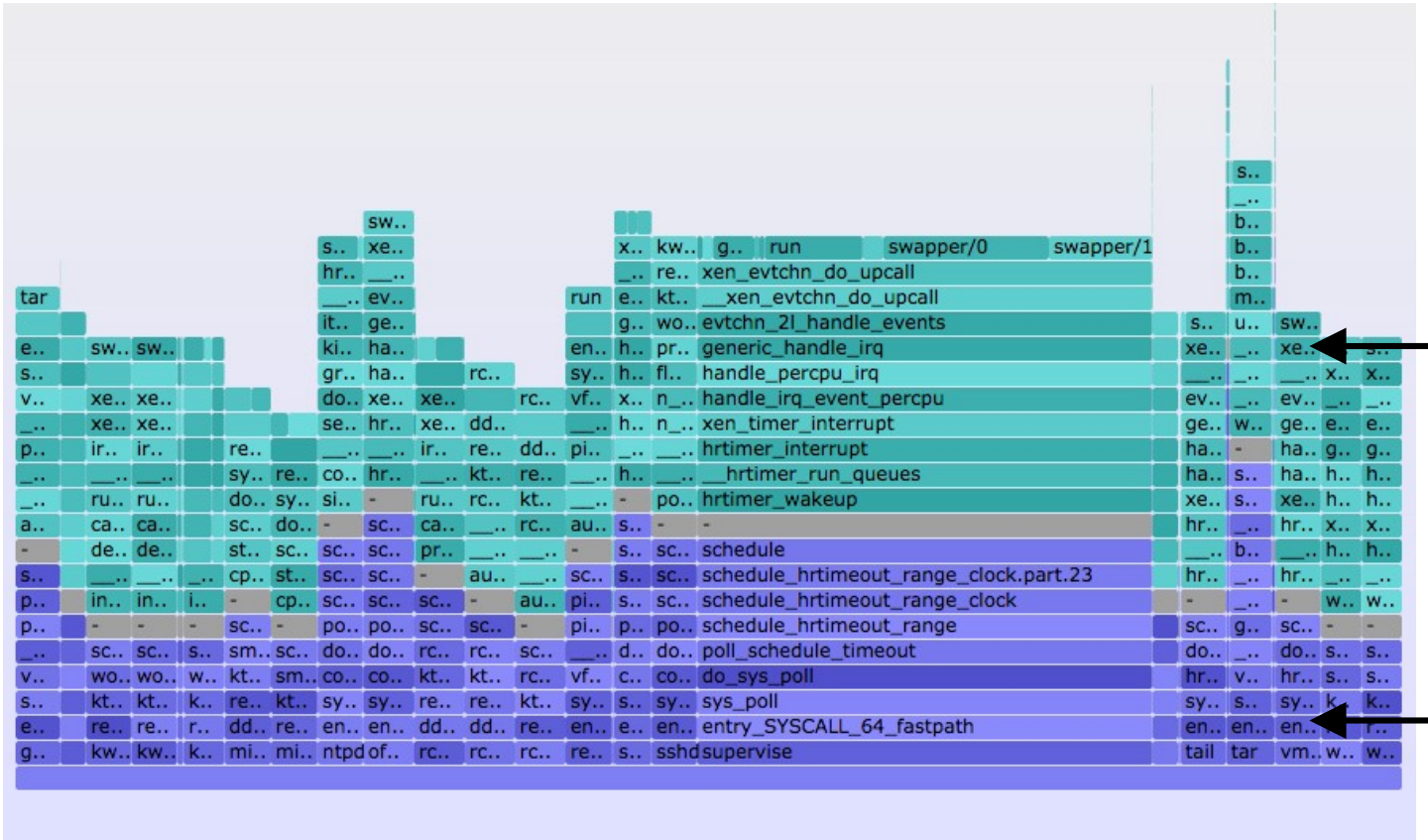
Off-CPU





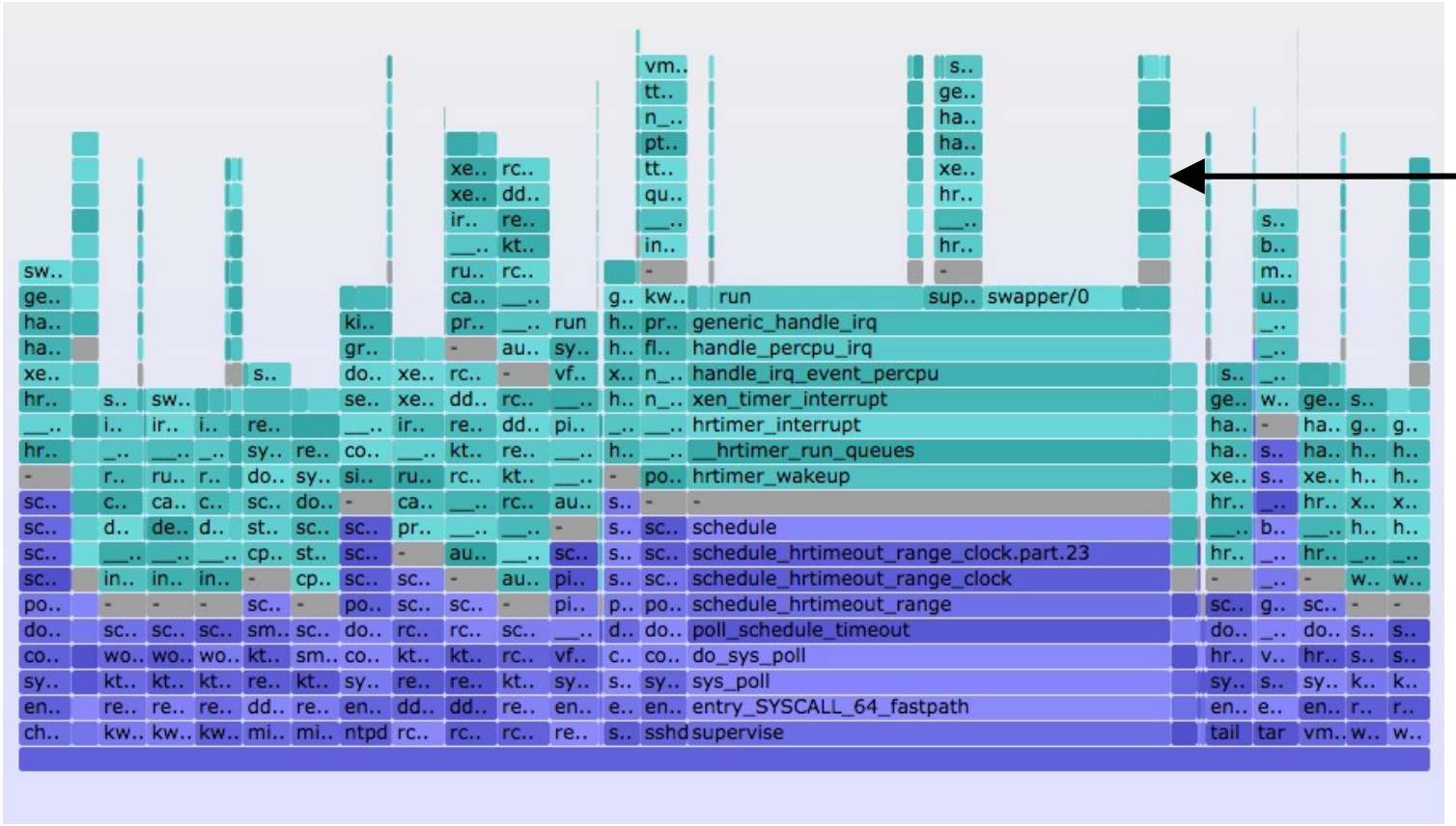
Everything
(All thread time)

Off-Wake Time Flame Graph



Uses Linux enhanced BPF to merge off-CPU and waker stack in kernel context

Chain Graphs

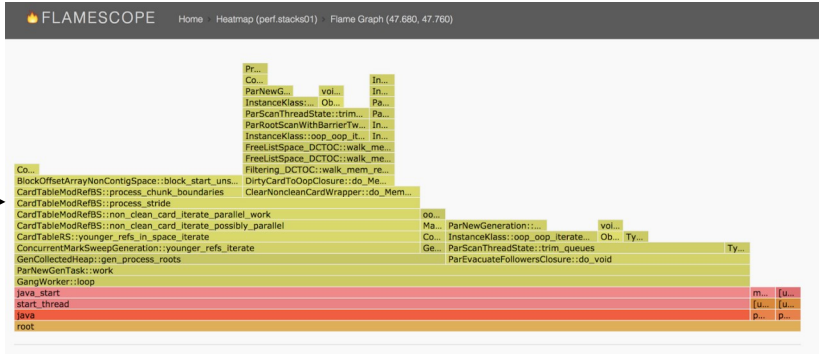


Walking the chain of wakeup stacks to reach root cause

FlameScope

Flame graphs can hide time-based issues of **variation** and **perturbations**.

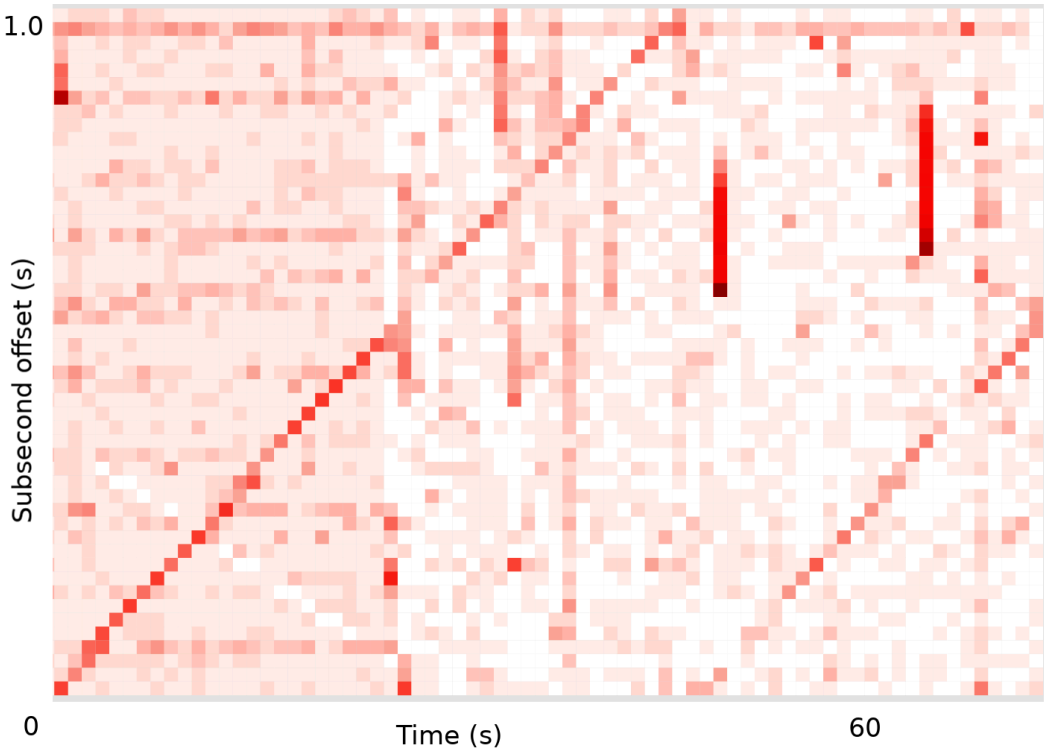
FlameScope uses **subsecond-offeset heat maps** to show these issues. They can then be selected for the corresponding flame graph.



https://brendangregg.com/blog/2018-12-15/flamescope-origin.html
https://www.brendangregg.com/HeatMaps/subsecondoffset.html

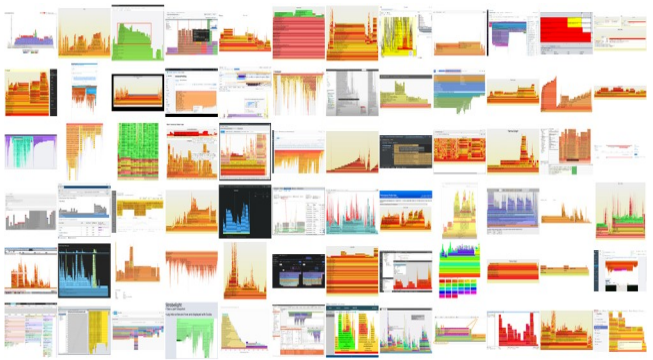
FlameScope Example

How many patterns can you see?

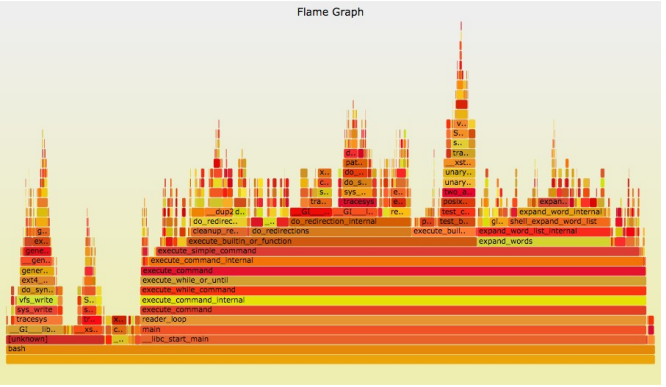


https://www.brendangregg.com/blog/2018-11-08/flamescope-pattern-recognition.html

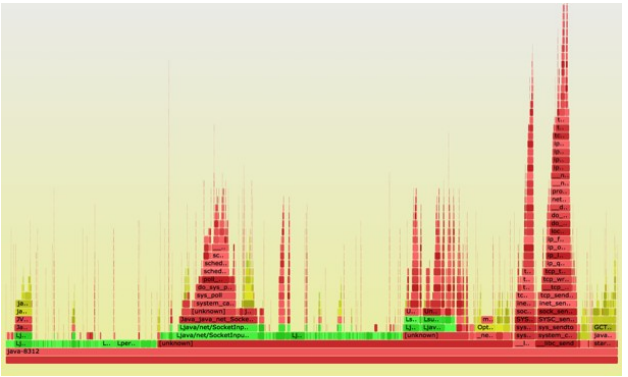
Agenda Recap



1. Implementations



2. CPU Flame graphs



3. Stacks & Symbols

4. Advanced flame graphs

Take Aways

1. Interpret CPU flame graphs
2. Understand runtime challenges
3. Why eBPF for advanced flame graphs

A new tool to lower your cost, latency, and carbon

Links & References

Flame Graphs
- "The Flame Graph" Communications of the ACM, Vol. 56, No. 6 (June 2016)
- http://queue.acm.org/detail.cfm?id=2927301
- http://www.brendangregg.com/flamegraphs.html
- **http://www.brendangregg.com/flamegraphs.html#Updates**
- http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
- http://www.brendangregg.com/FlameGraphs/memoryflamegraphs.html
- http://www.brendangregg.com/FlameGraphs/offcpuflamegraphs.html
- http://techblog.netflix.com/2015/07/java-in-flames.html
- http://techblog.netflix.com/2016/04/saving-13-million-computational-minutes.html
- http://www.brendangregg.com/blog/2014-11-09/differential-flame-graphs.html
- http://www.brendangregg.com/blog/2016-01-20/ebpf-offcpu-flame-graph.html
- http://www.brendangregg.com/blog/2016-02-01/linux-wakeup-offwake-profiling.html
- http://www.brendangregg.com/blog/2016-02-05/ebpf-chaingraph-prototype.html
- https://brendangregg.com/blog/2018-12-15/flamescope-origin.html
- https://github.com/brendangregg/FlameGraph
- https://github.com/spiermar/d3-flame-graph
- https://github.com/Netflix/flamescope
- http://corpaul.github.io/flamegraphdiff/
- https://www.intel.com/content/www/us/en/develop/documentation/vtune-help/top/reference/user-interface-reference/window-flame-graph.html
- https://gprofiler.io/

Linux perf
- https://perf.wiki.kernel.org/index.php/Main_Page
- http://www.brendangregg.com/perf.html

Linux eBPF
- https://ebpf.io/ https://www.brendangregg.com/ebpf.html

These slides: https://www.brendangregg.com/Slides/YOW2022_flame_graphs.pdf

YOW! 2022

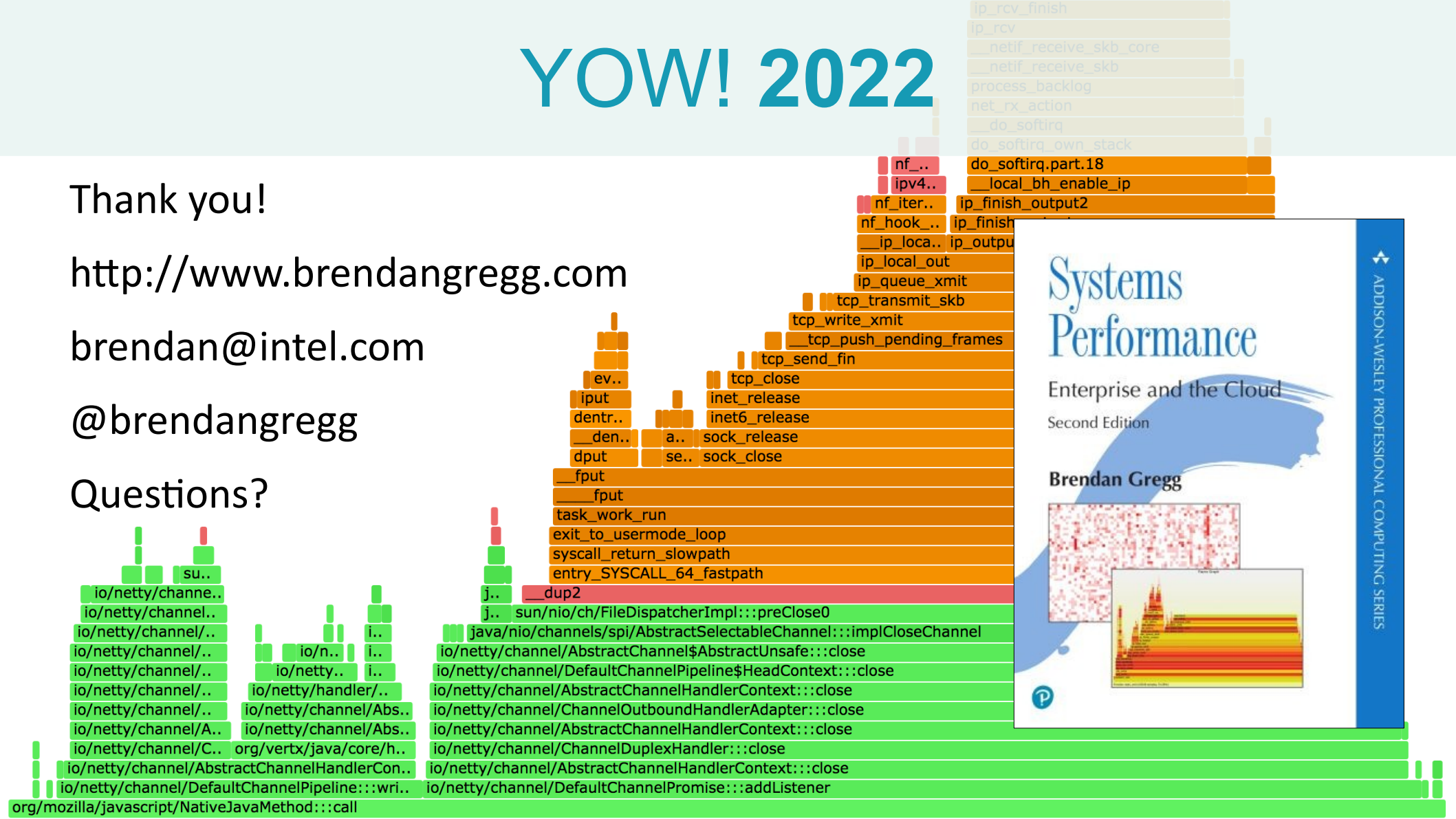
Thank you!

http://www.brendangregg.com

brendan@intel.com

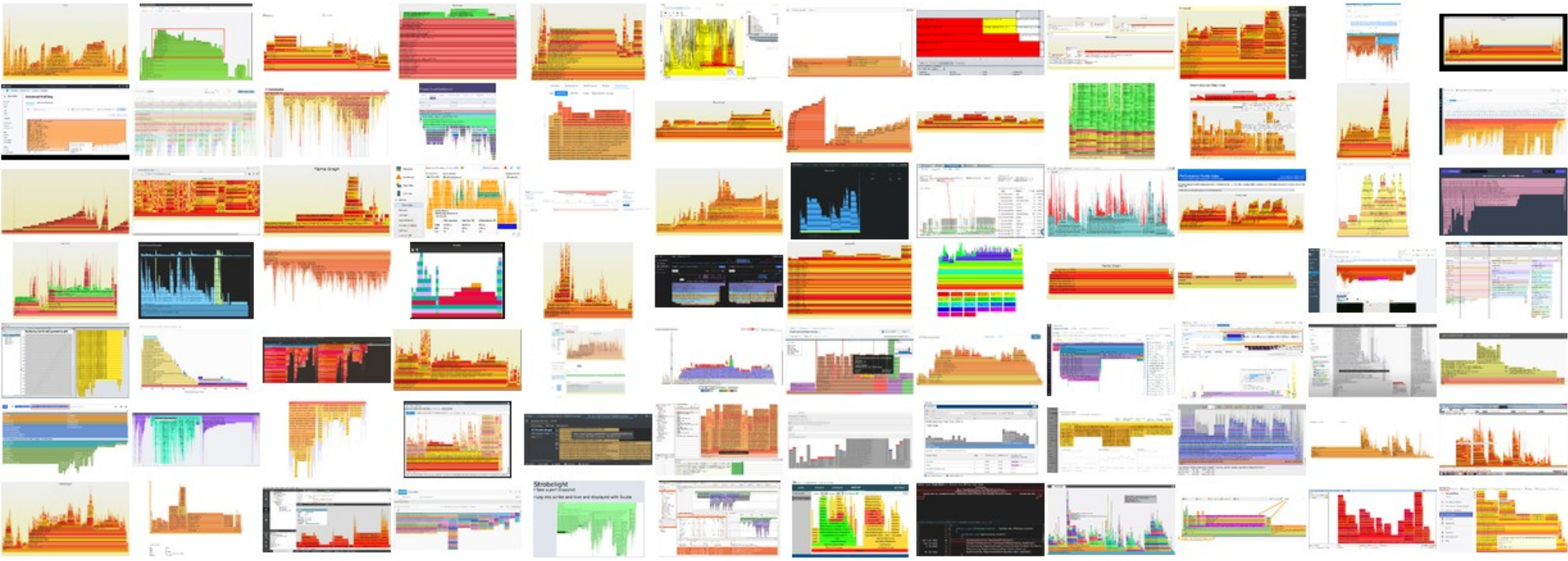
@brendangregg

Questions?



BONUS SLIDES

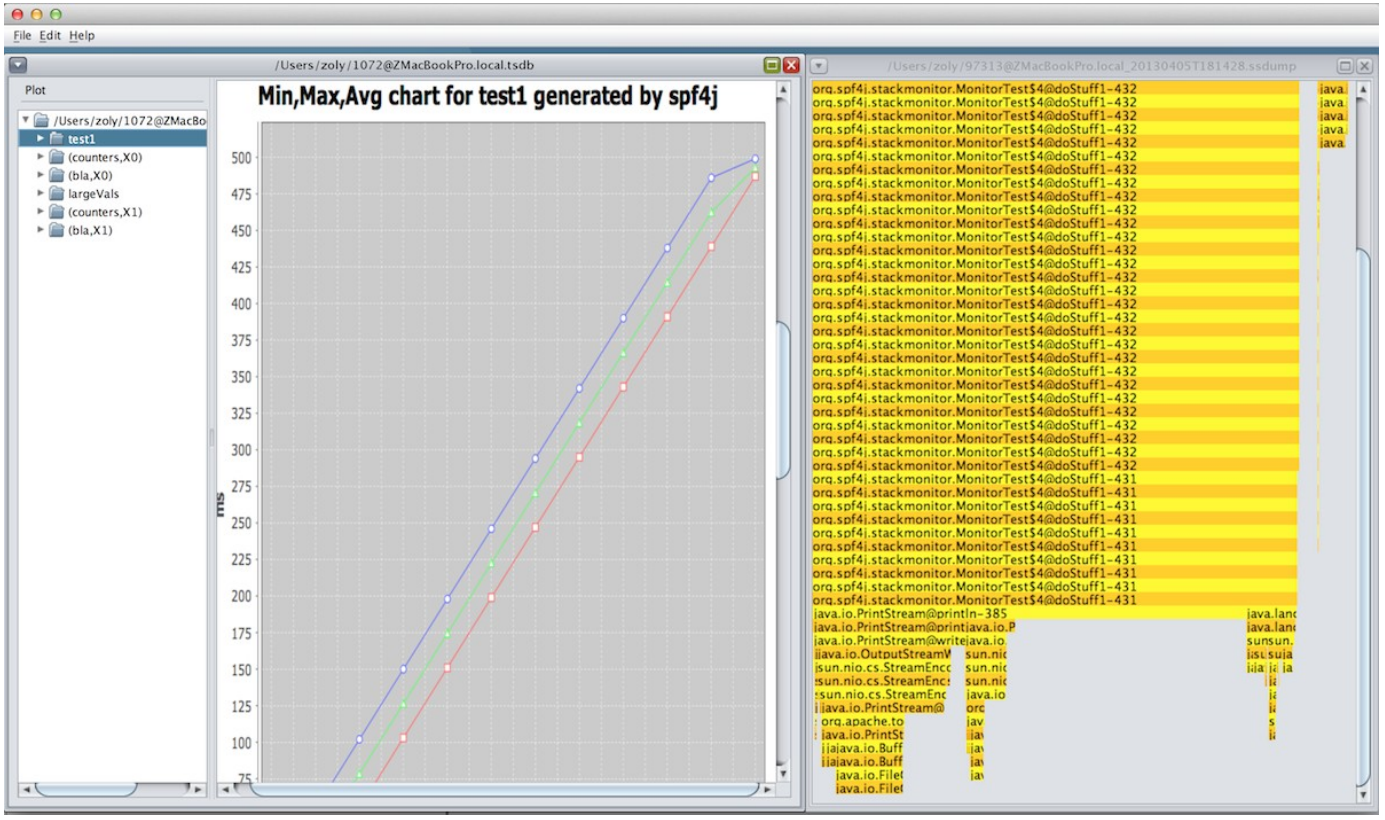
More Implementations



These are in addition to the earlier examples.

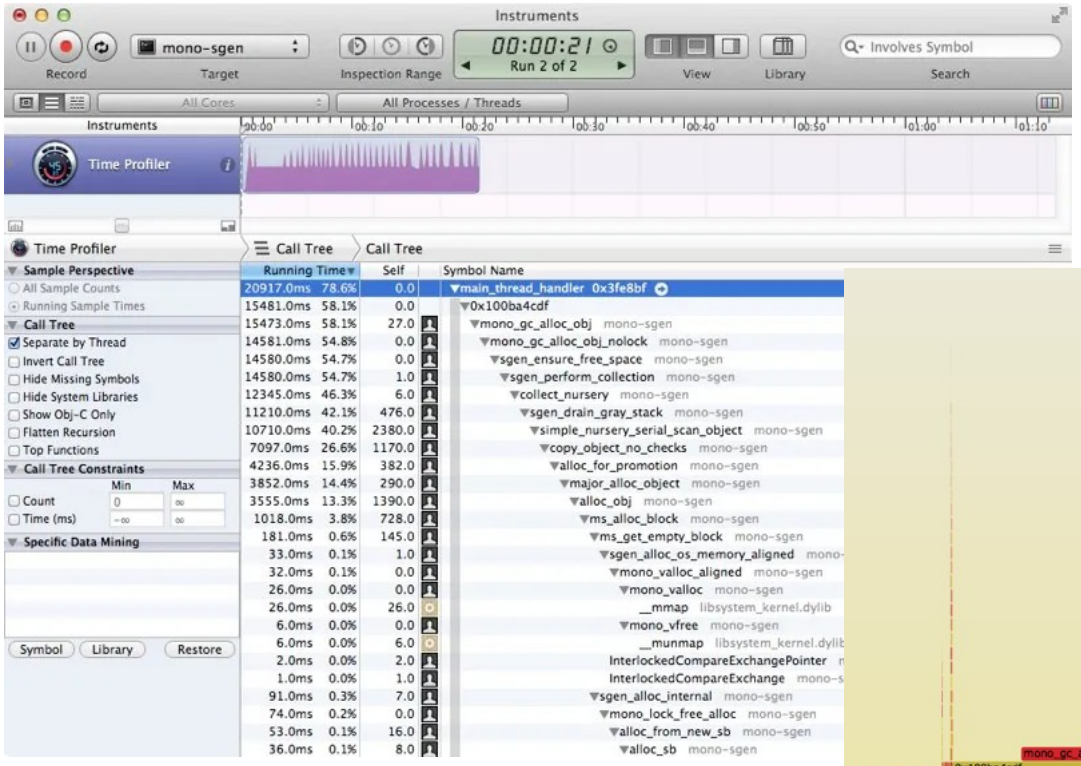
(Note: This is not an an endorsement of any company/product or sponsored by anyone.)

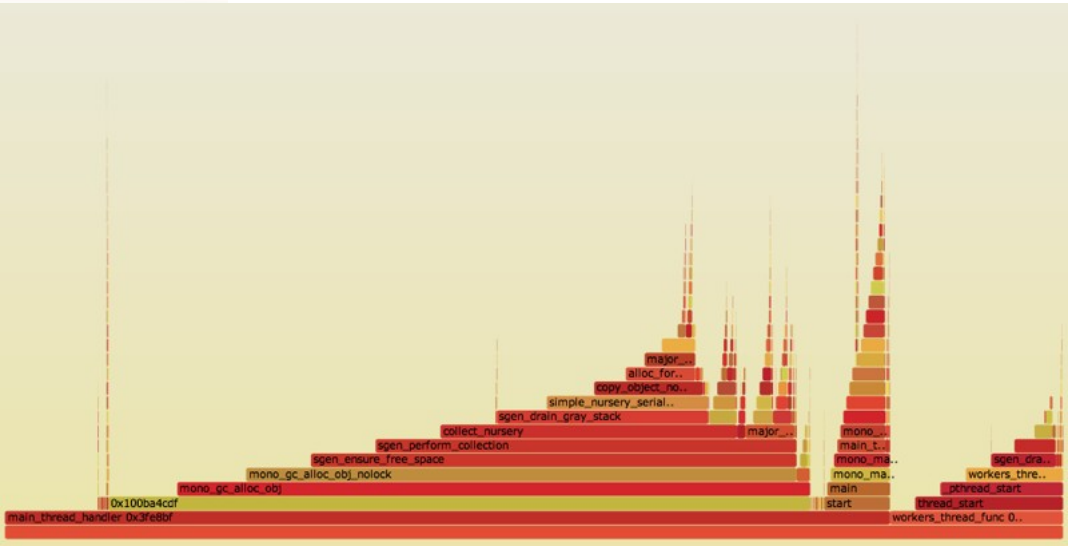
Java: SPF4J (2012)



Source: http://zolyfarkas.github.io/spf4j/#

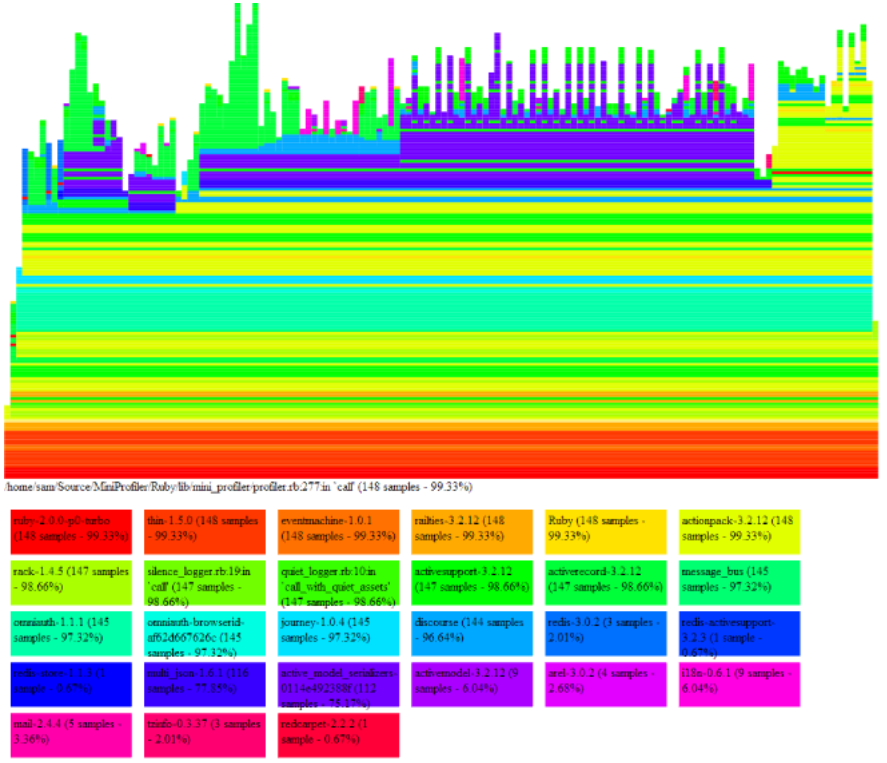
OSX: Instruments (2012; converter)





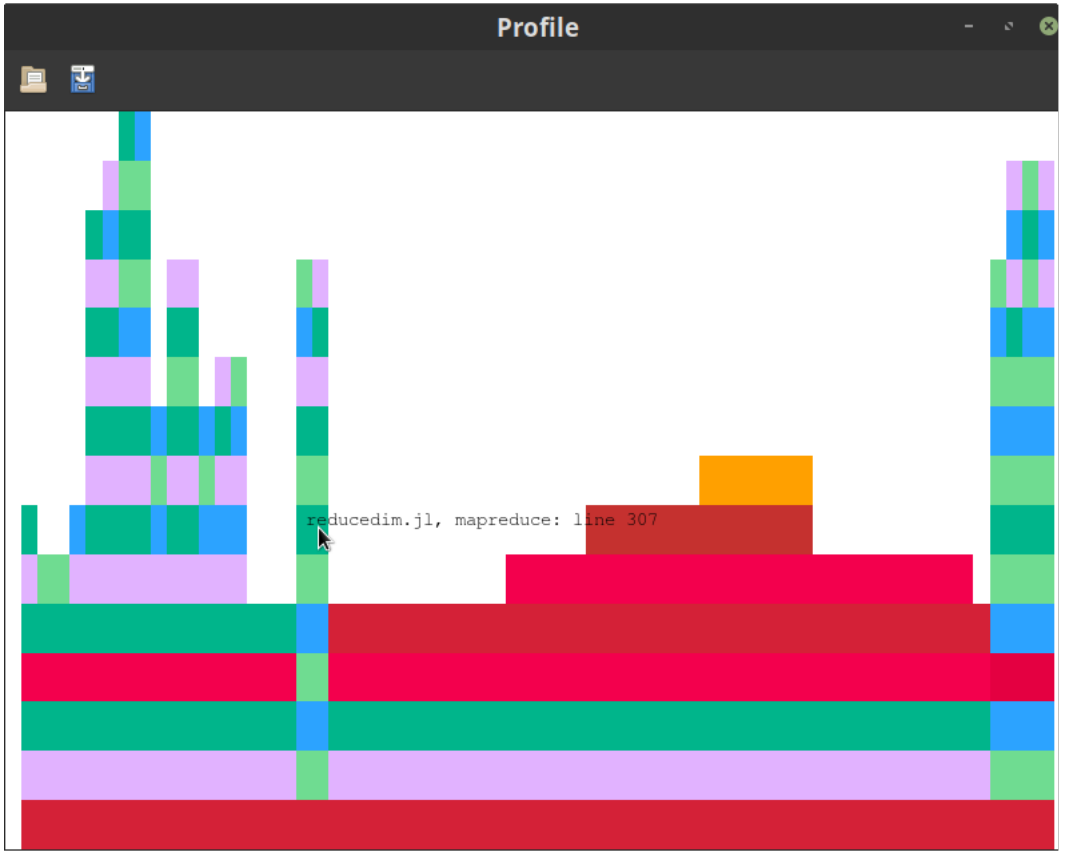
Source: https://schani.wordpress.com/2012/11/16/flame-graphs-for-instruments/

Ruby: mini-profiler (2013)



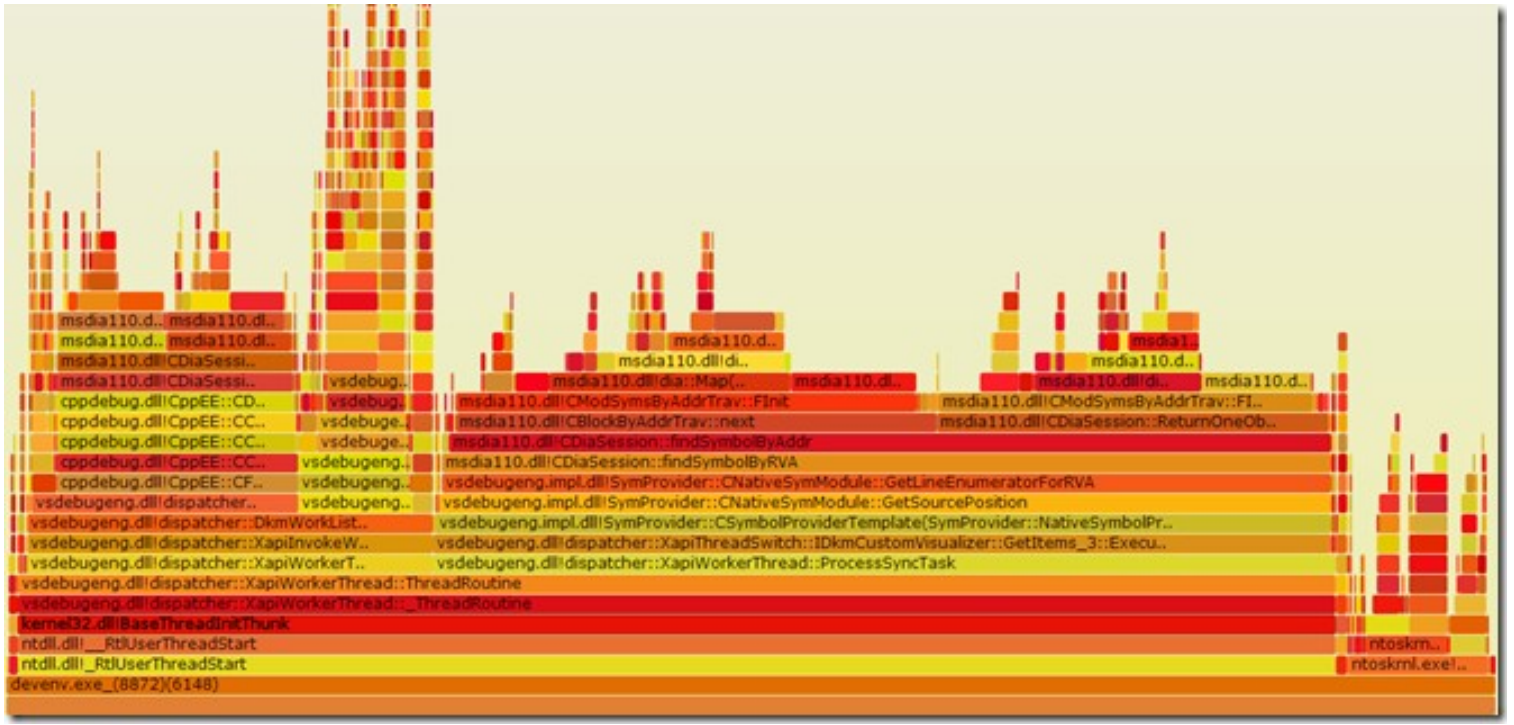
Source: https://samsaffron.com/archive/2013/03/19/flame-graphs-in-ruby-miniprofiler

Julia: ProfileView.jl (2013)



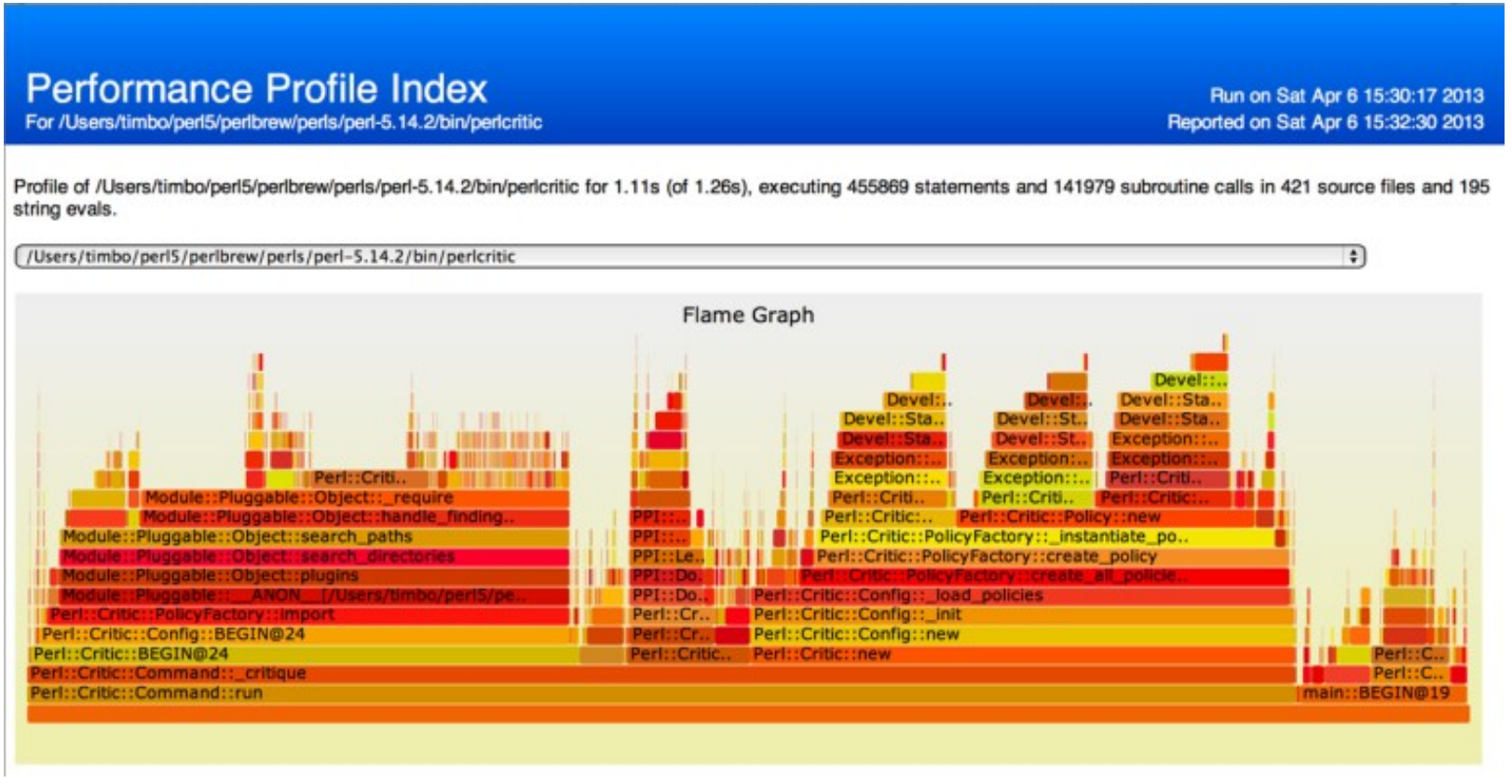
Source: https://github.com/timholy/ProfileView.jl (Tim Holy)

Windows: Xperf (2013; converter)



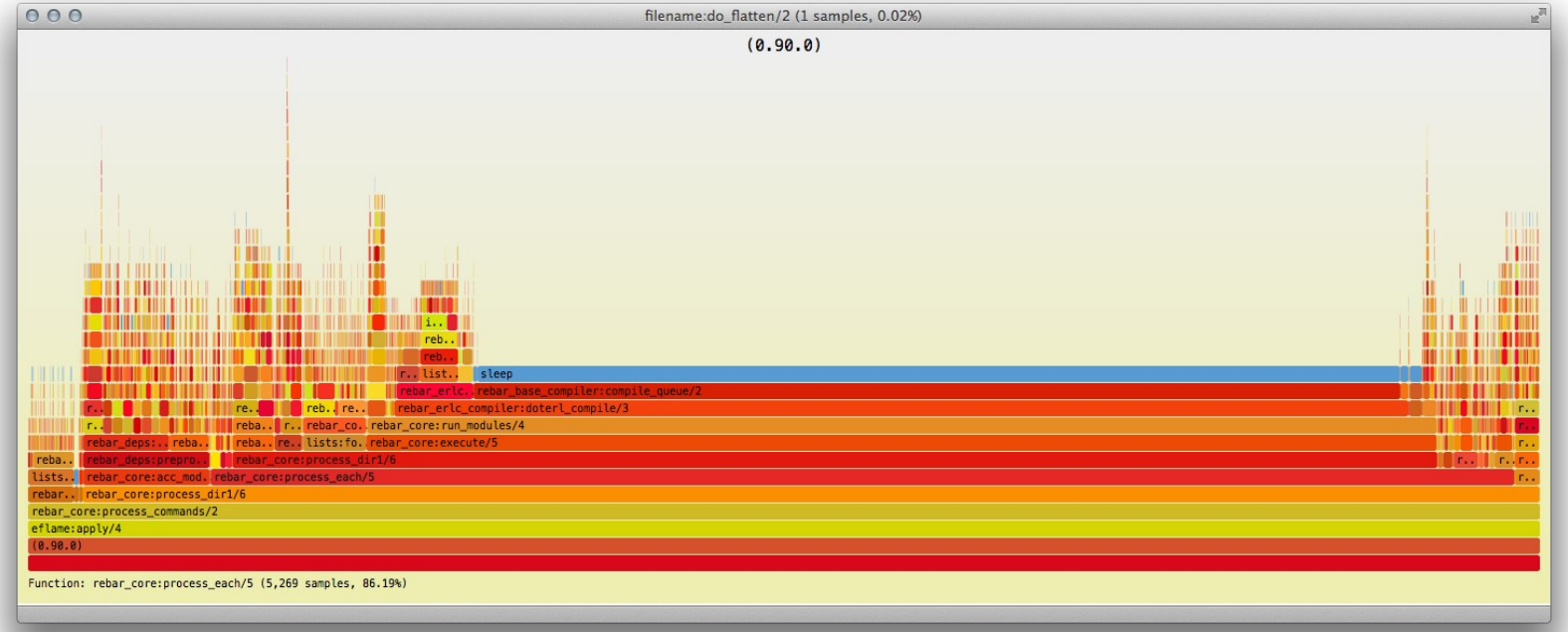
Source: https://randomascii.wordpress.com/2013/03/26/summarizing-xperf-cpu-usage-with-flame-graphs/ (Bruce Dawson)

Perl: NYTProf (2013)



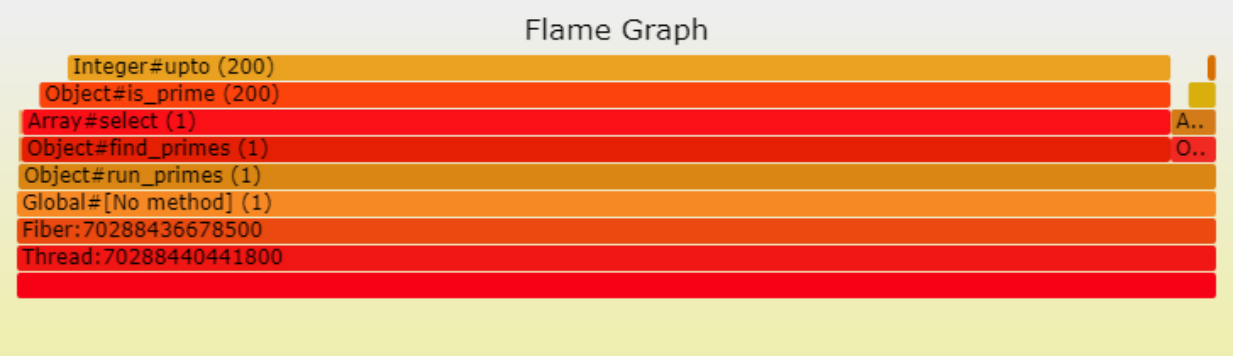
Source: https://blog.timbunce.org/2013/04/08/nytprof-v5-flaming-precision/ (Tim Bunce)

Erlang: Eflame (2013)



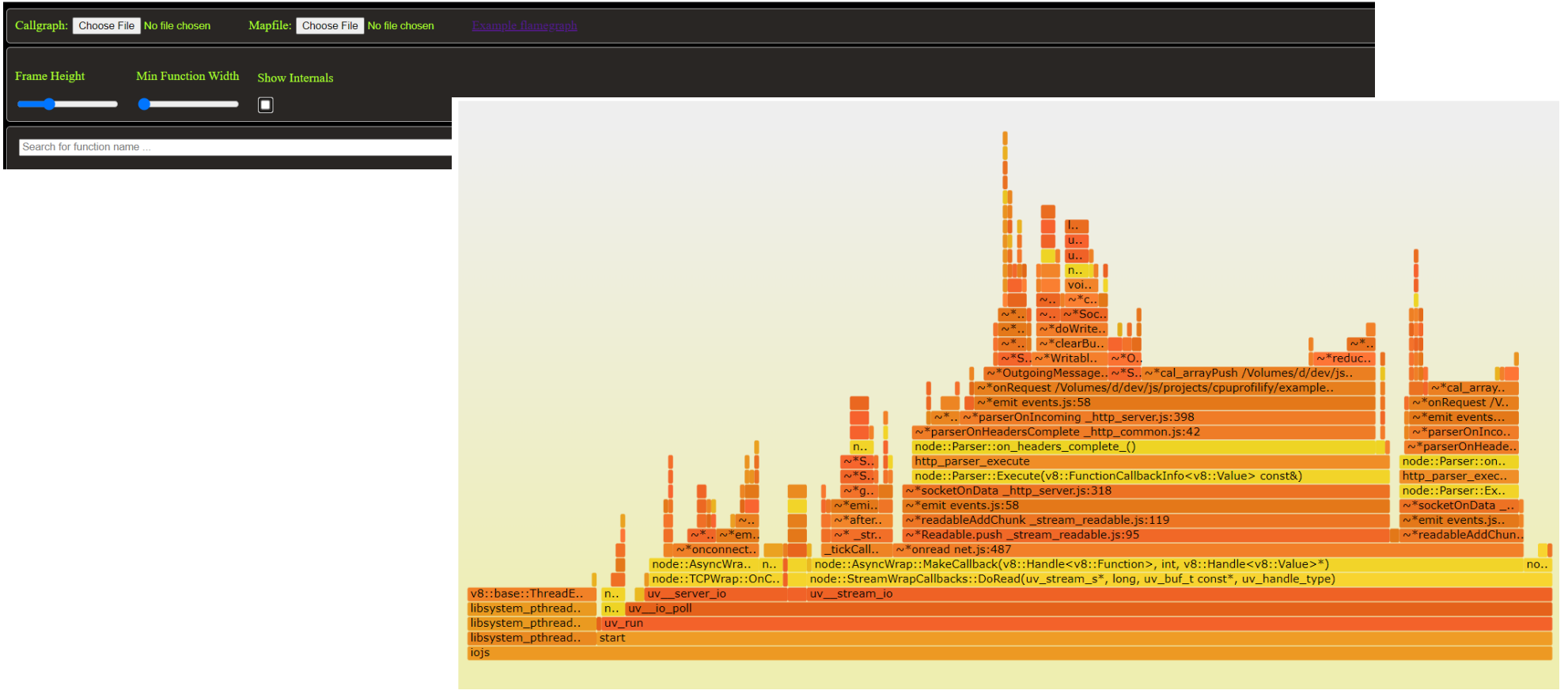
Source: https://github.com/proger/eflame (Volodymyr Ky)

Ruby: ruby-prof-flamegraph (2014)



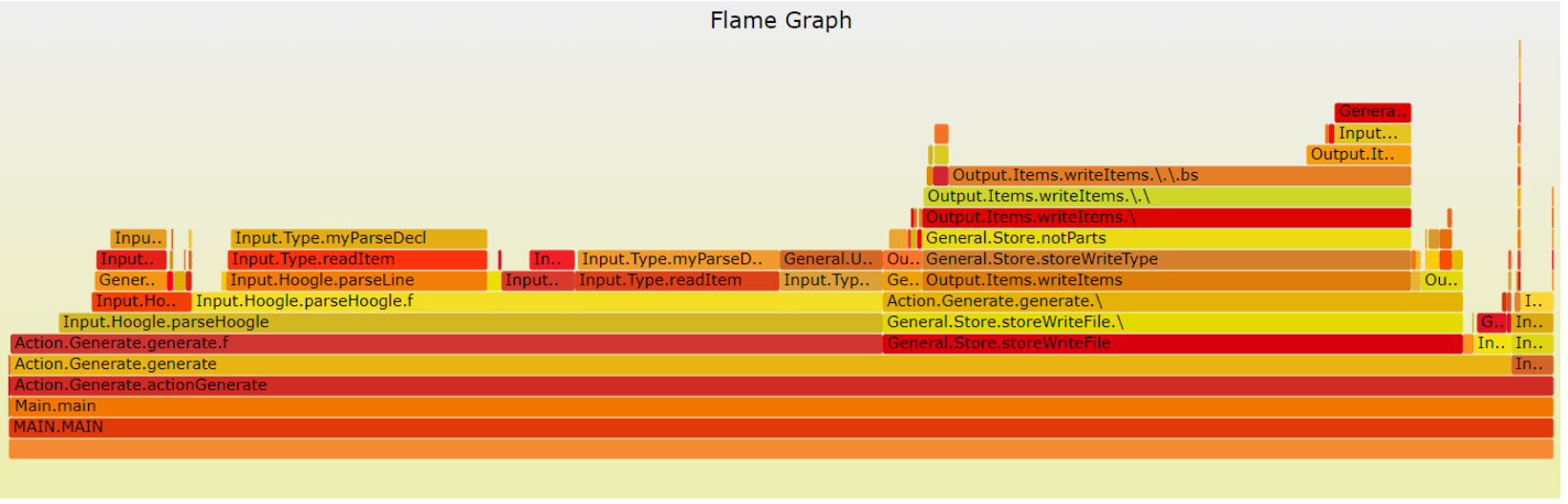
Source: https://github.com/oozou/ruby-prof-flamegraph

Node.js: flamegraph (2015)



Source: https://github.com/thlorenz/flamegraph (Thorsten Lorenz)

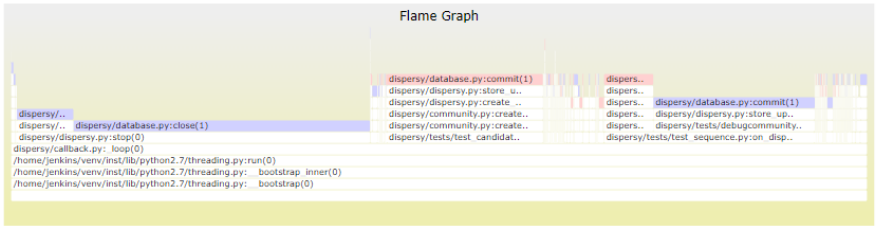
Haskell: ghc-prof-flamegraph (2015)



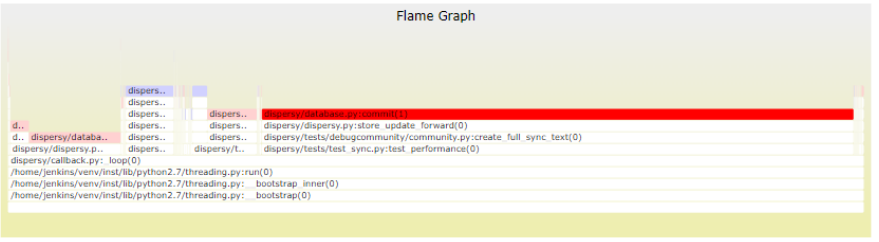
Source: https://www.fpcomplete.com/blog/2015/04/ghc-prof-flamegraph/ (Francesco Mazzoli)

Differentials: Flamegraphdiff (2015)

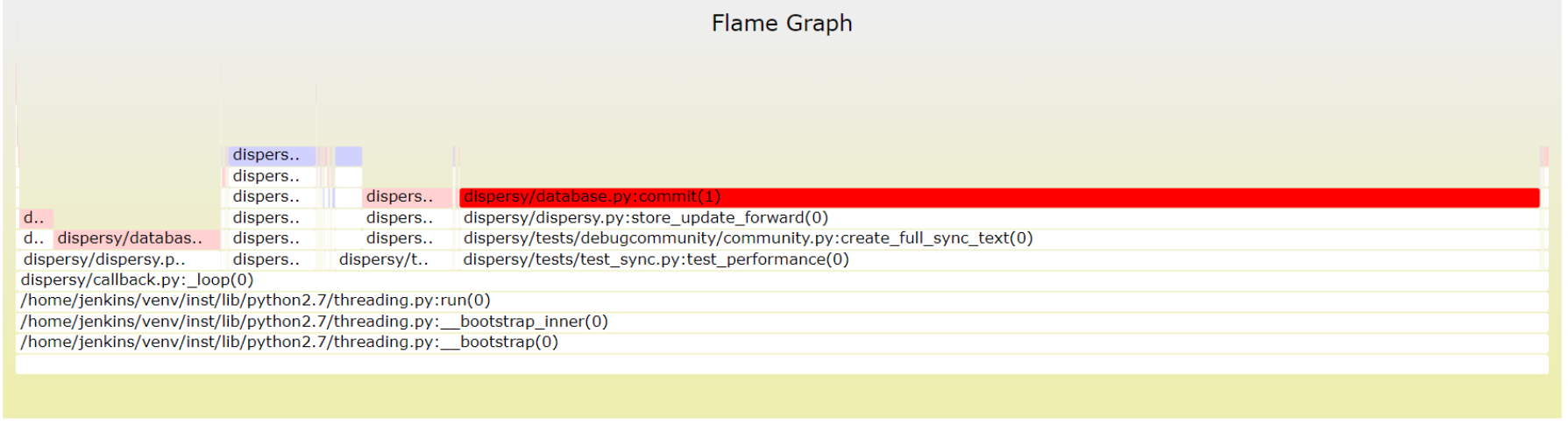
DFG1:



DFG2:

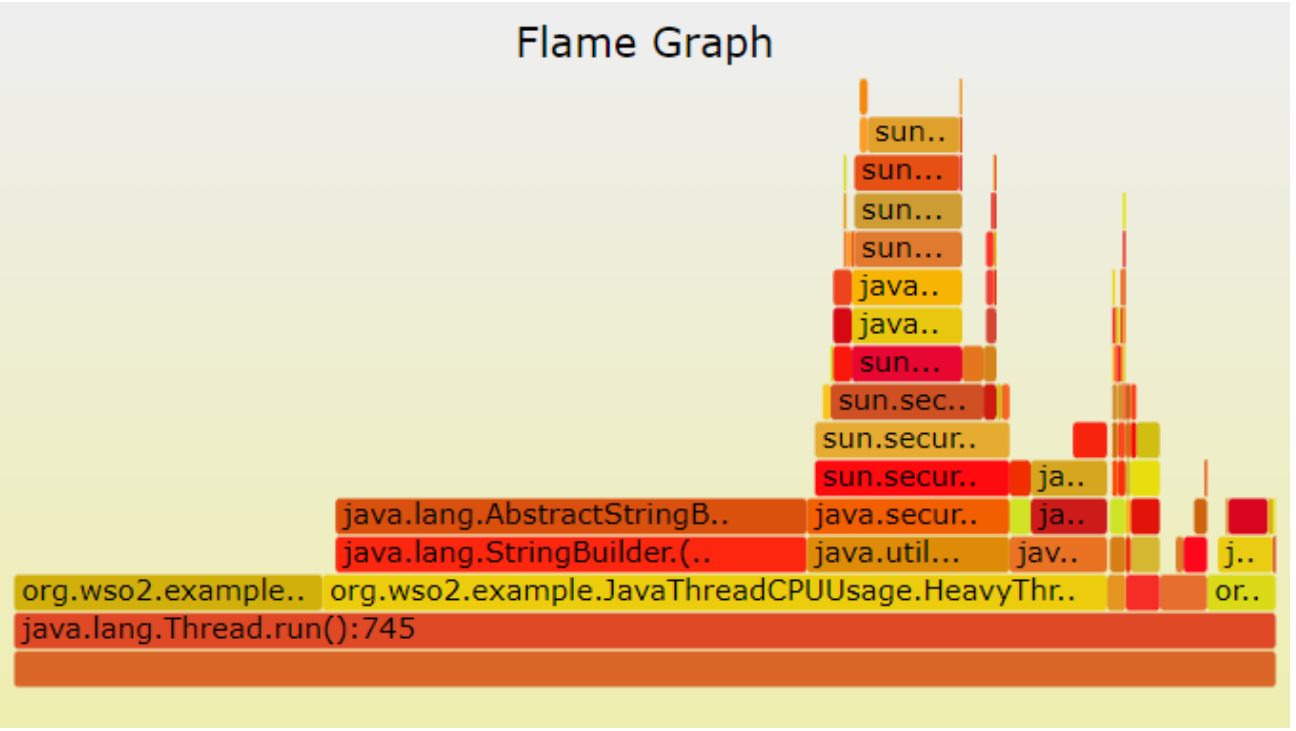


DFG_diff:



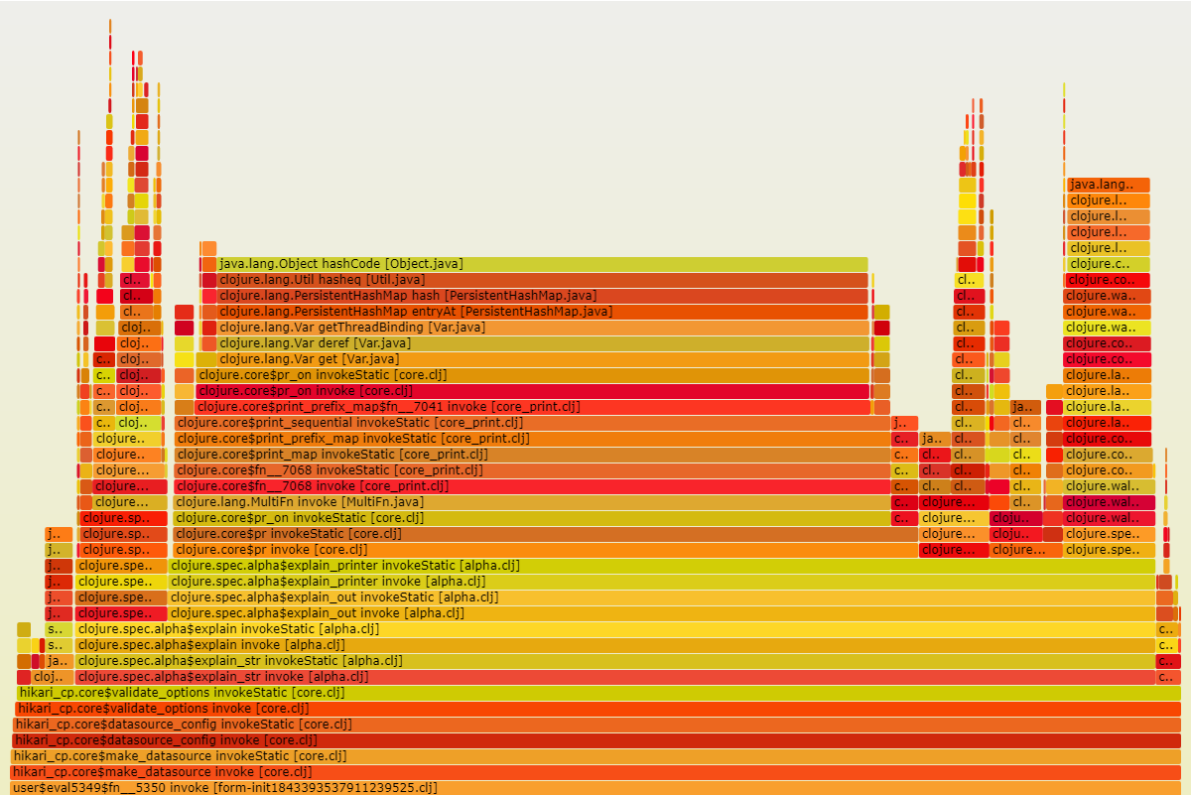
Source: http://corpaul.github.io/flamegraphdiff/ (Cor-Paul Bezemer)

Java: jfr-flame-graph (2015)



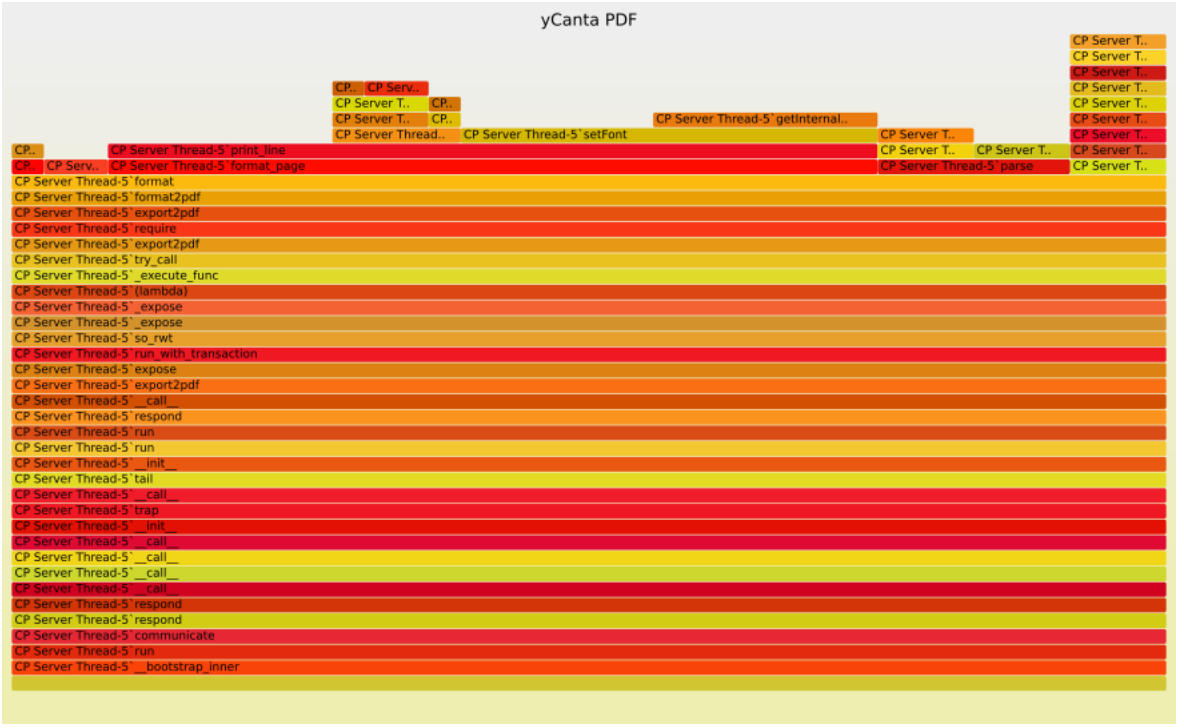
Source: http://isuru-perera.blogspot.com/2015/05/flame-graphs-with-java-flight-recordings.html
(M. Isuru Tharanga Chrishantha Perera)

Clojure: Flames (2015)



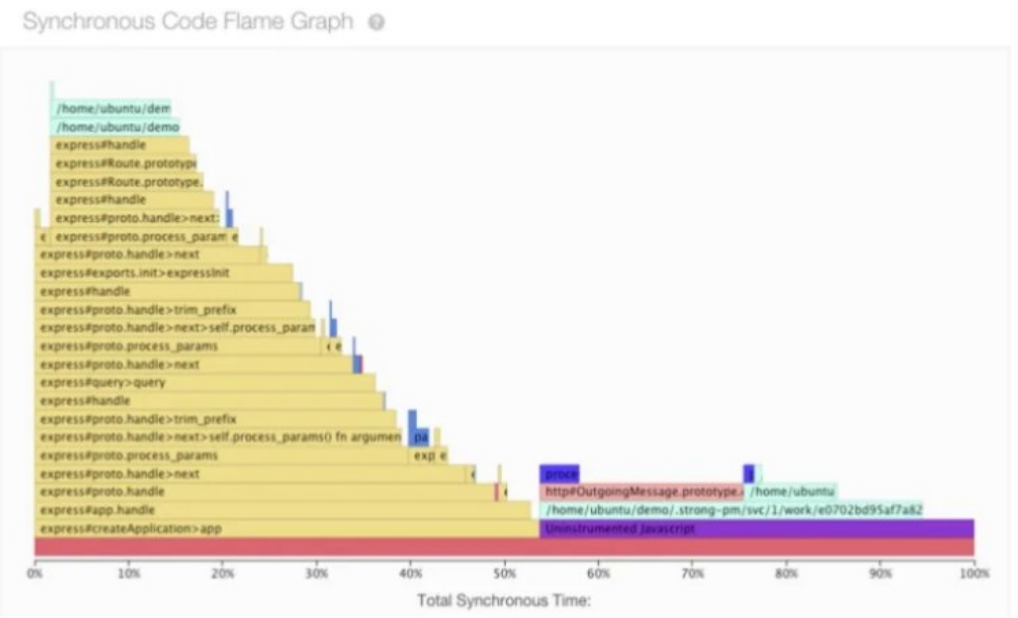
Source: https://github.com/jstepien/flames/ (Jan Stępień)

Python: python-flamegraph (2015)



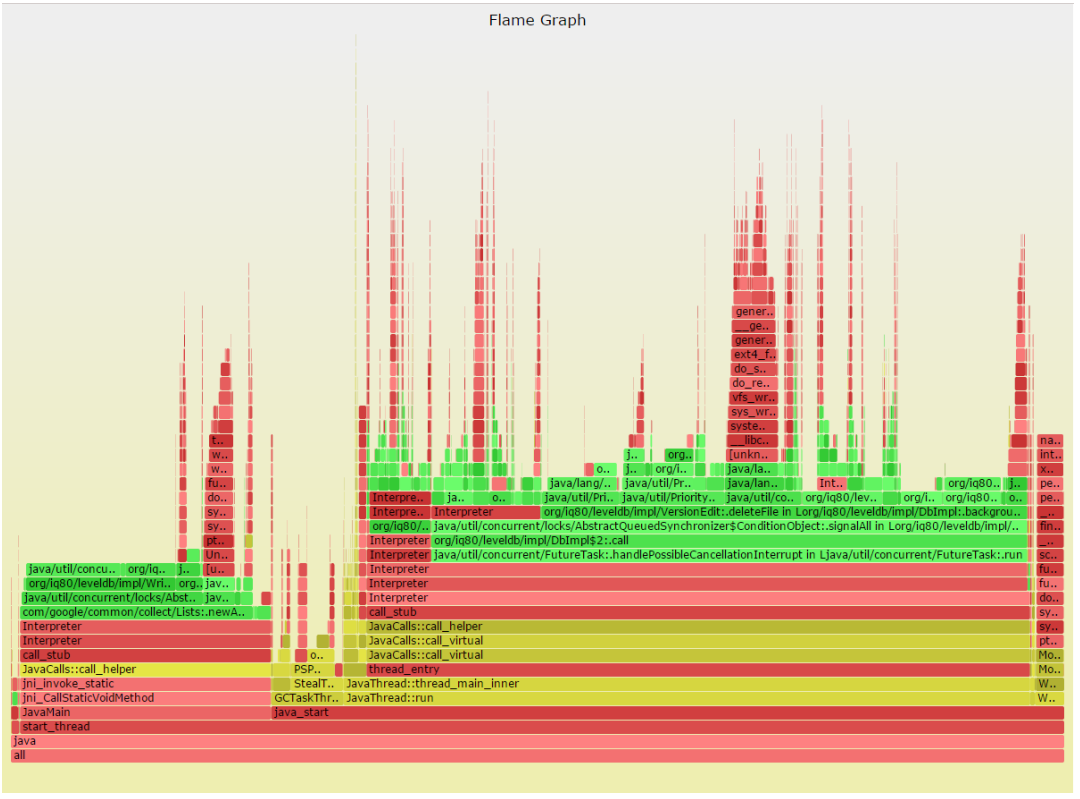
Source: https://github.com/evanhempel/python-flamegraph (Evan Hempel)

Strongloop: Arc (2015)



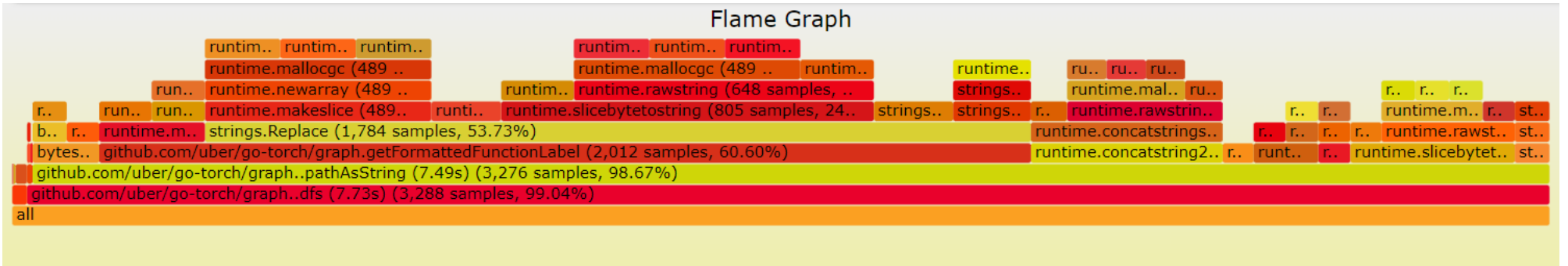
Source: https://es.slideshare.net/jguerrero999/nodejs-transaction-tracing-root-cause-analysis-with-strongloop-arc

Java: perfj (2015)



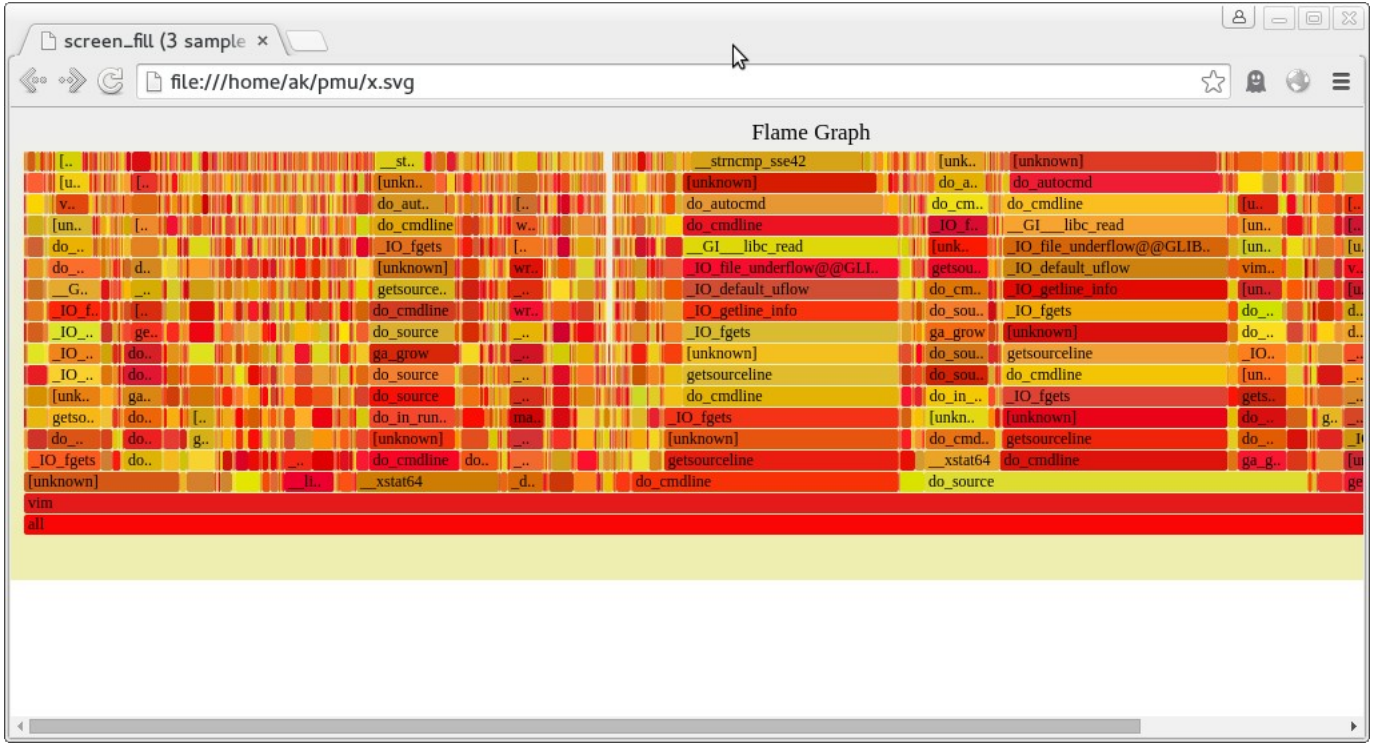
Source: https://github.com/coderplay/perfj (Min Zhou)

Golang: Uber go-torch (2015)



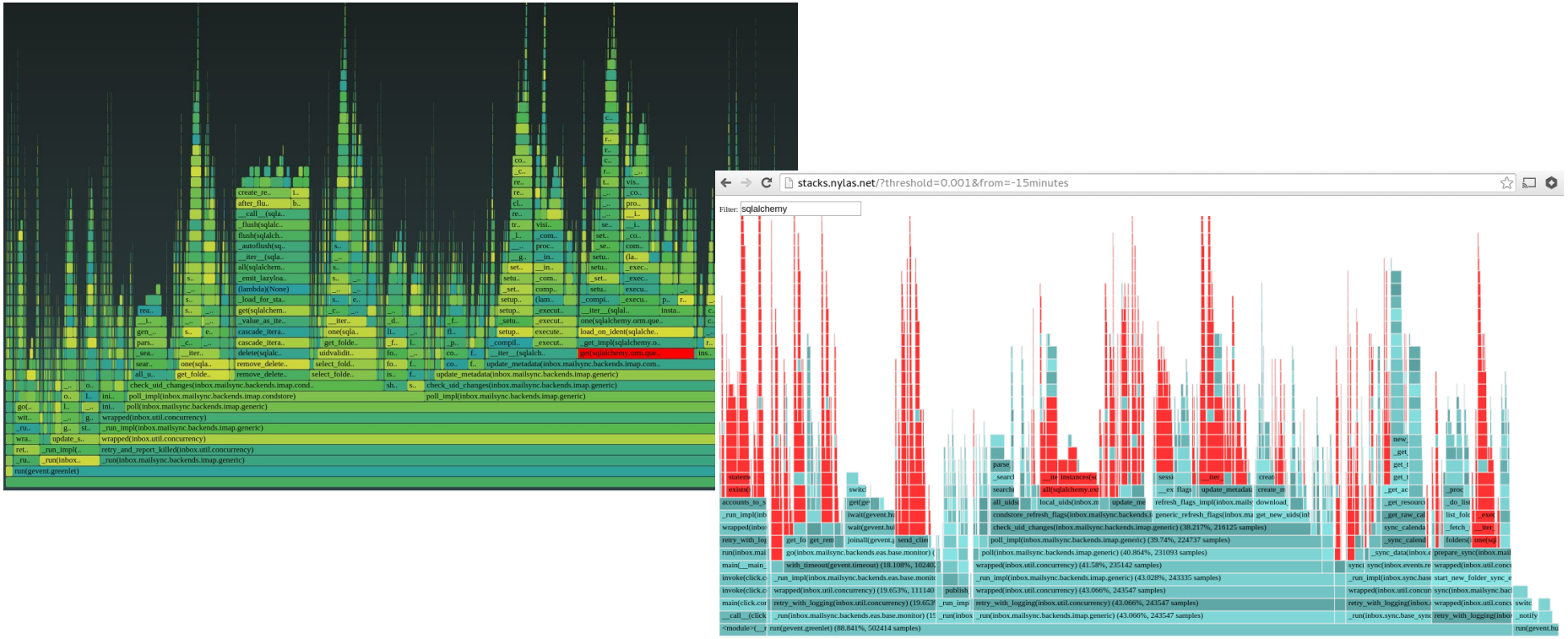
Source: https://github.com/uber-archive/go-torch

Intel: processor trace converter (2015)



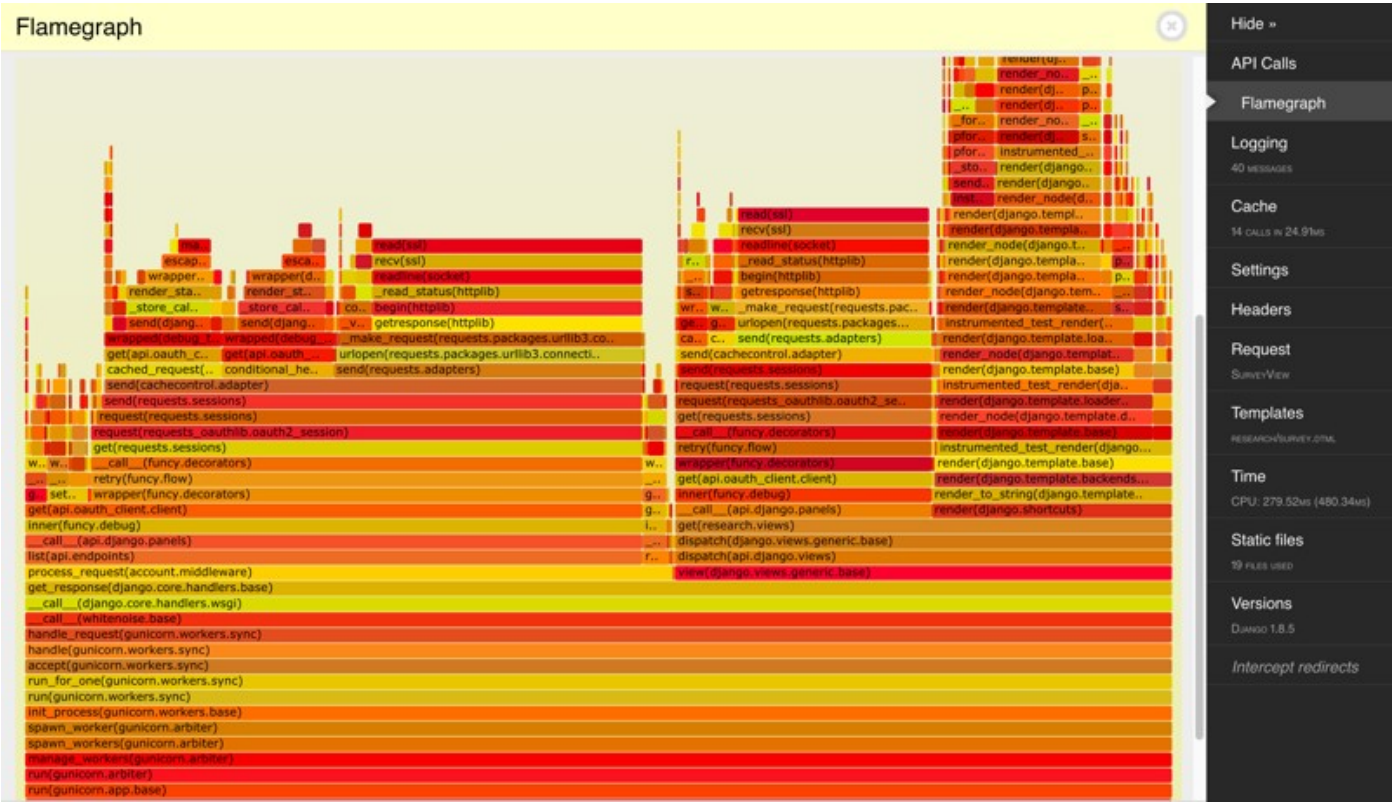
Source: http://halobates.de/blog/p/329 (Andi Kleen)

Nylas: perftools (2015)



Source: https://www.nylas.com/blog/performance/ (code by Eben Freeman)

Django: djdt-flamegraph (2015)

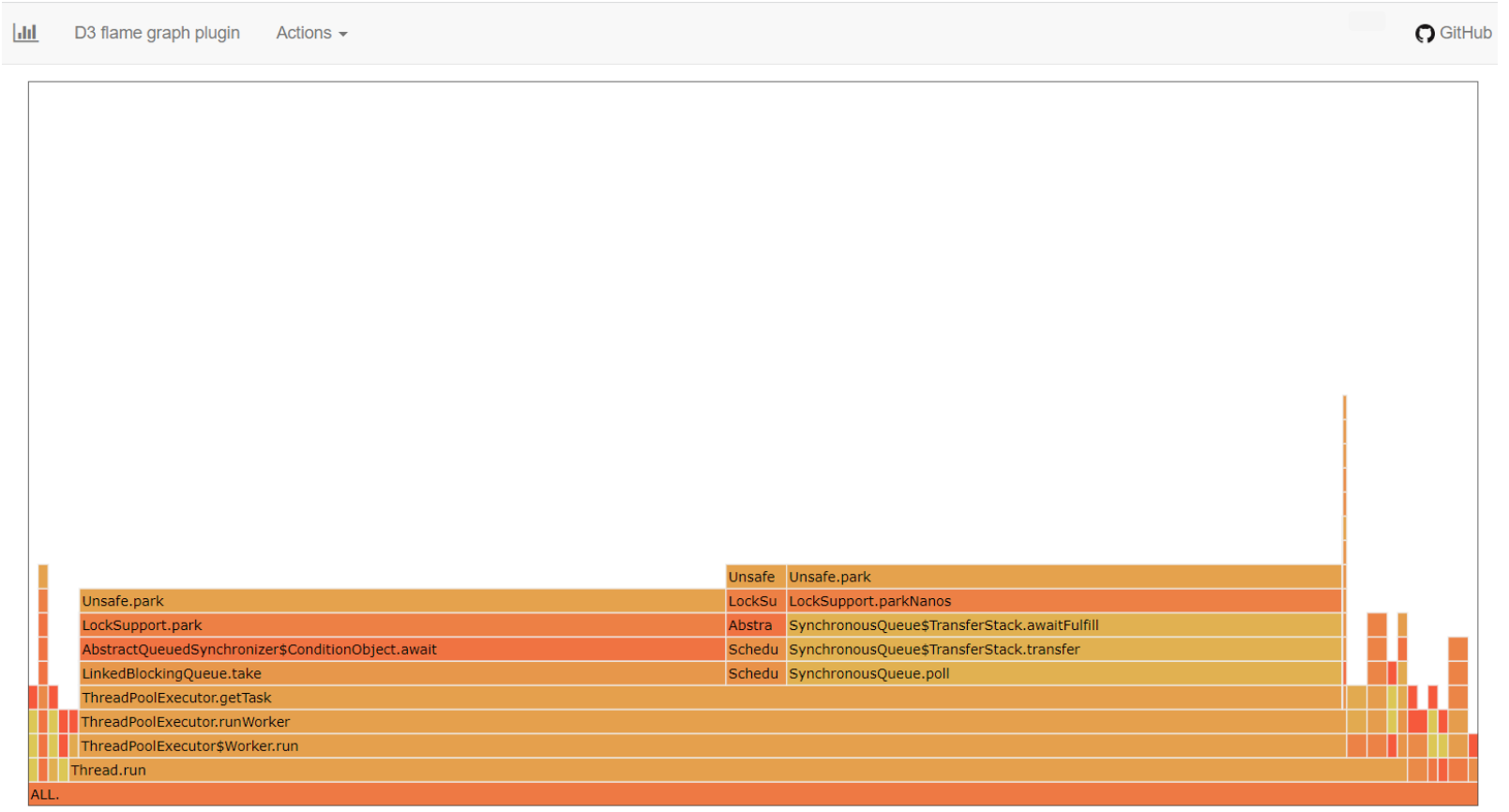


Source: https://github.com/blopker/djdt-flamegraph (Bo Lopker)

NodeSource: Nsolid (Node.js; 2015)

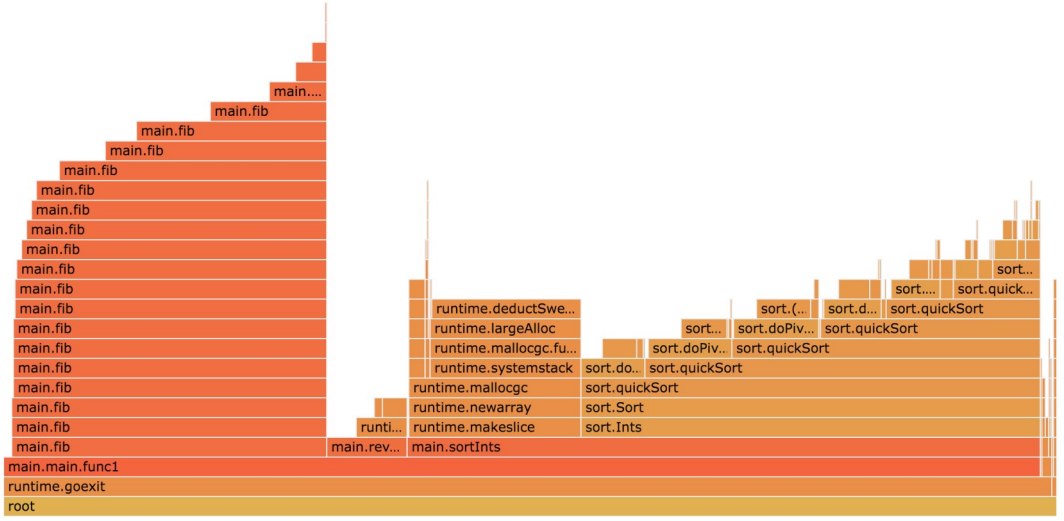
Source: https://nodesource.com/blog/understanding-cpu-flame-graphs

D3: d3-flame-graphs (2015)



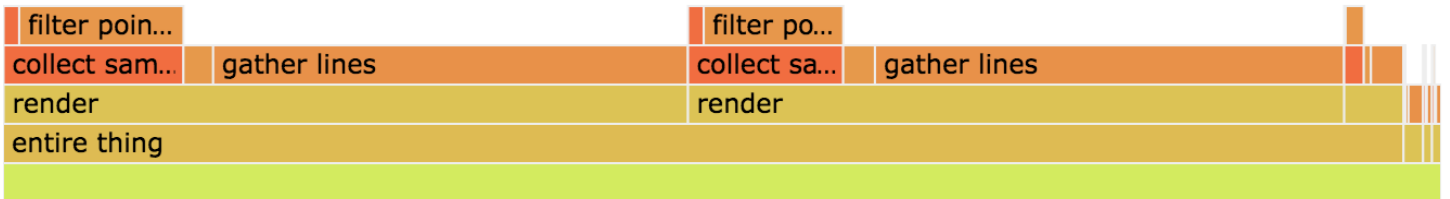
Source: https://cimi.io/d3-flame-graphs/ (Alex Ciminian)

Golang: Goprofui (2015)



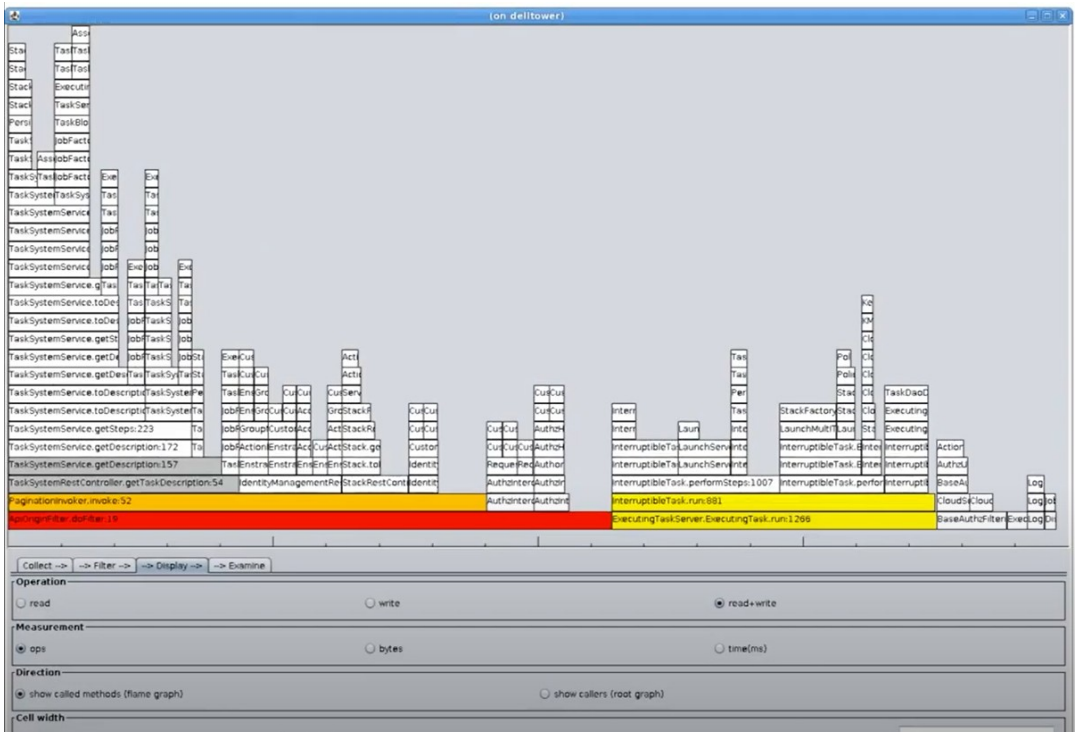
Source: https://github.com/wirelessregistry/goprofui (Srdjan Marinovic, Julia Allyce)

Rust: flame (2016)

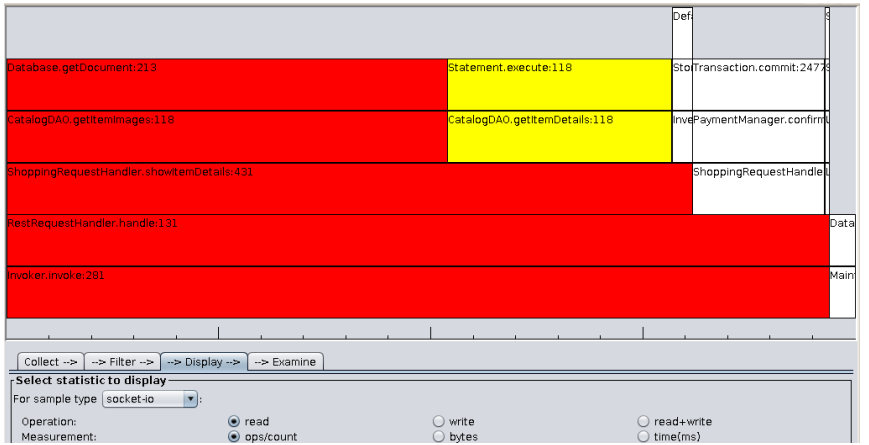


Source: https://github.com/llogiq/flame (Ty Overby)

Dell Cloud Manager: Gumshoe Load Investigator (2016)

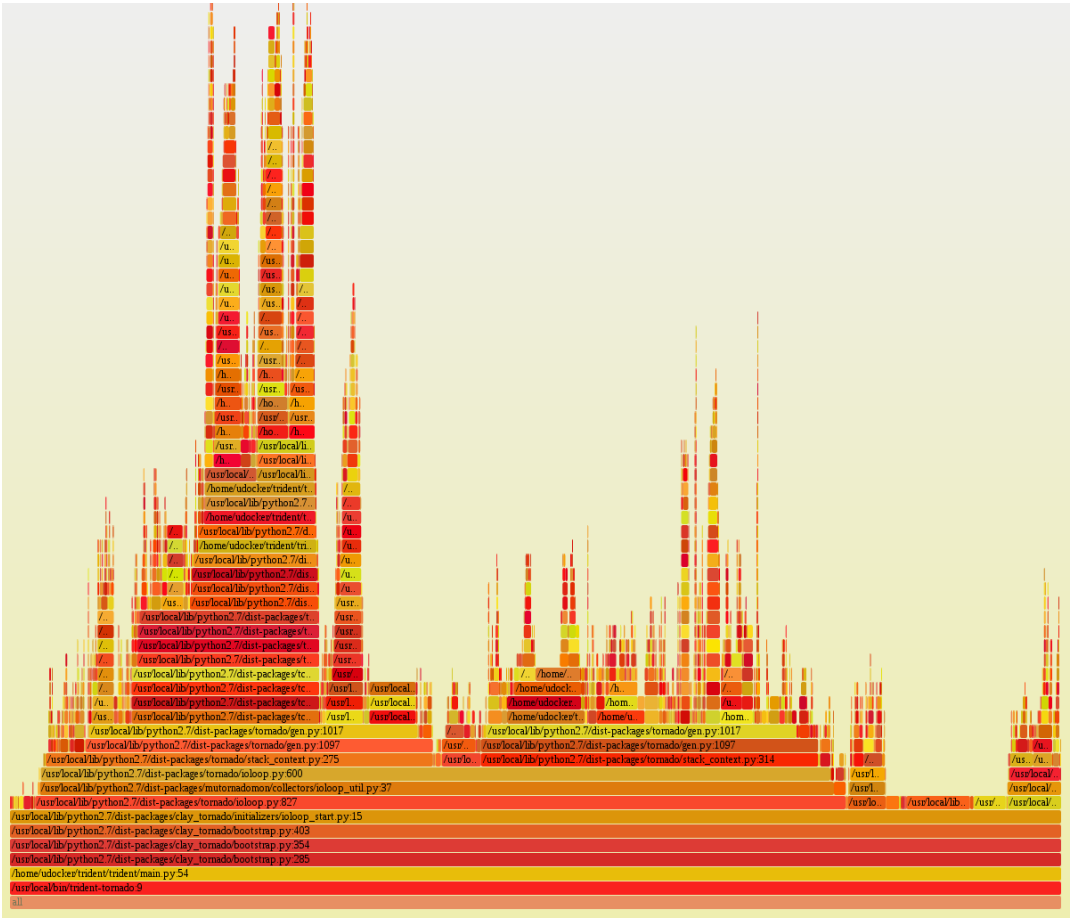


"This haystack is looking more like a needle every minute" -- source: https://youtu.be/GGJFZfwXJ44?t=225



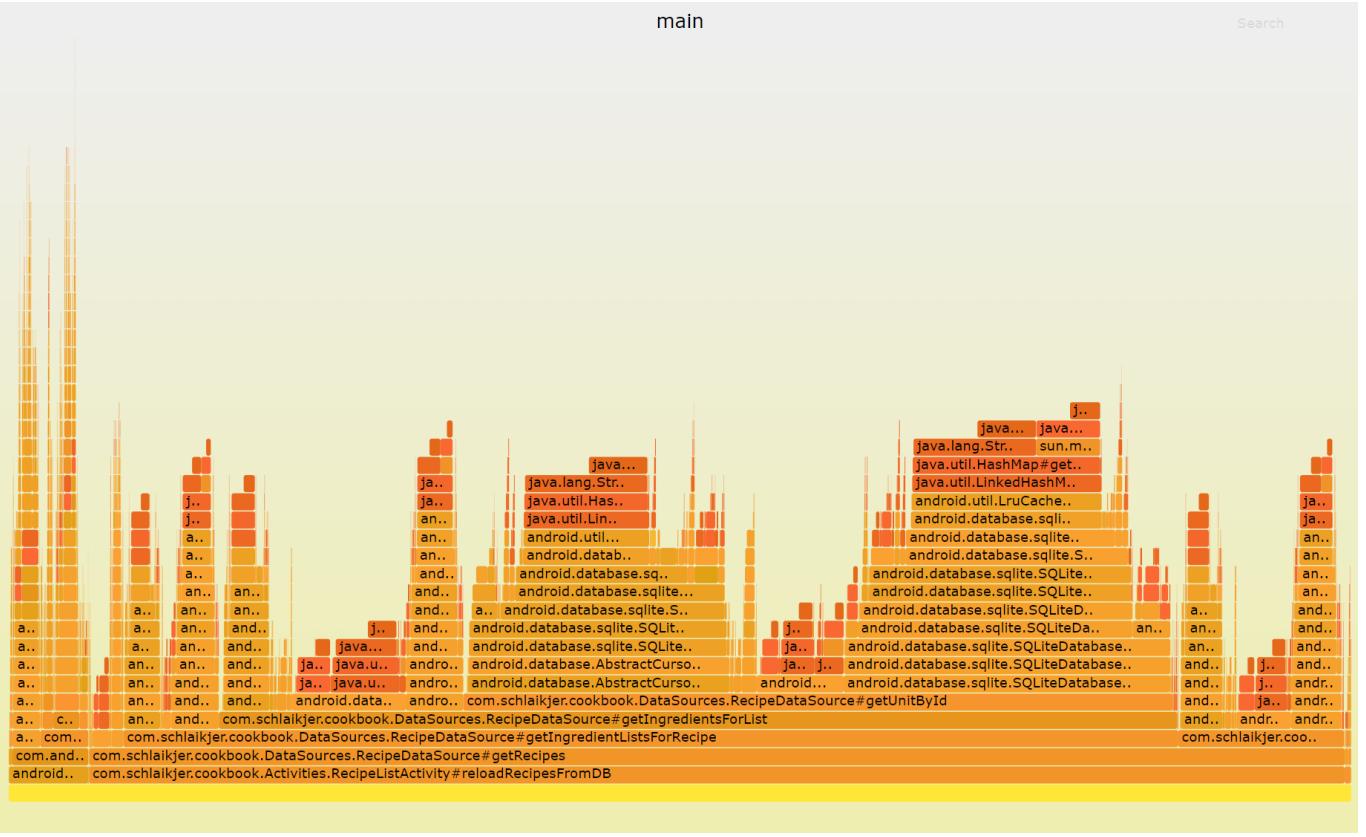
Source: https://github.com/worstcase/gumshoe (Jonathan Newbrough)

Uber: pyflame (Python; 2016)



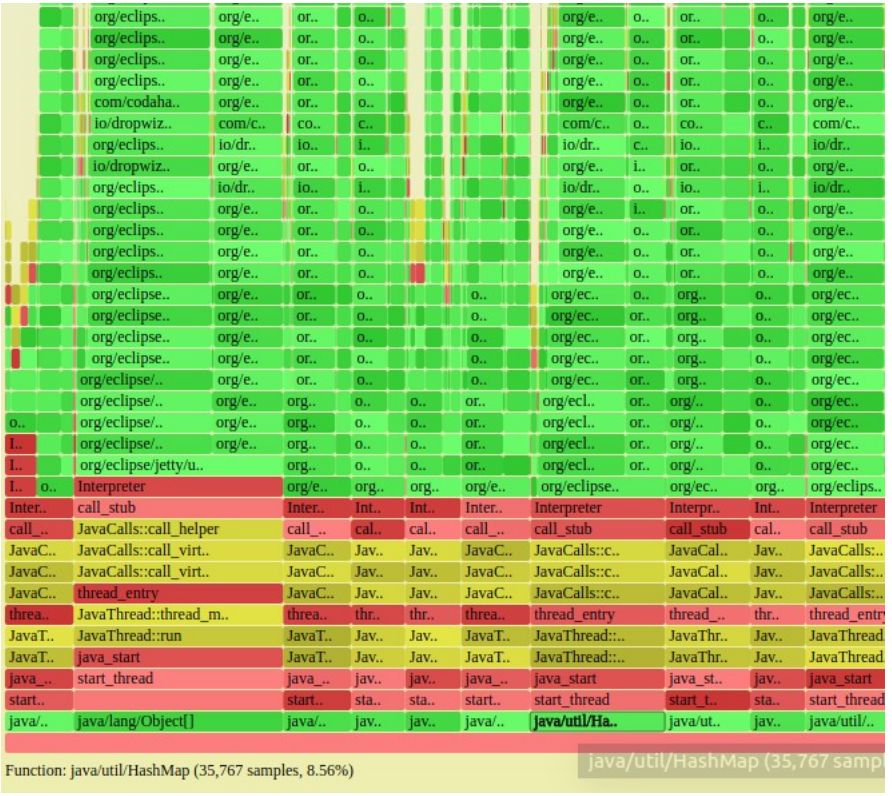
Source: https://www.uber.com/en-AU/blog/pyflame-python-profiler/

Android: erlang-atrace-flamegraphs (2017)



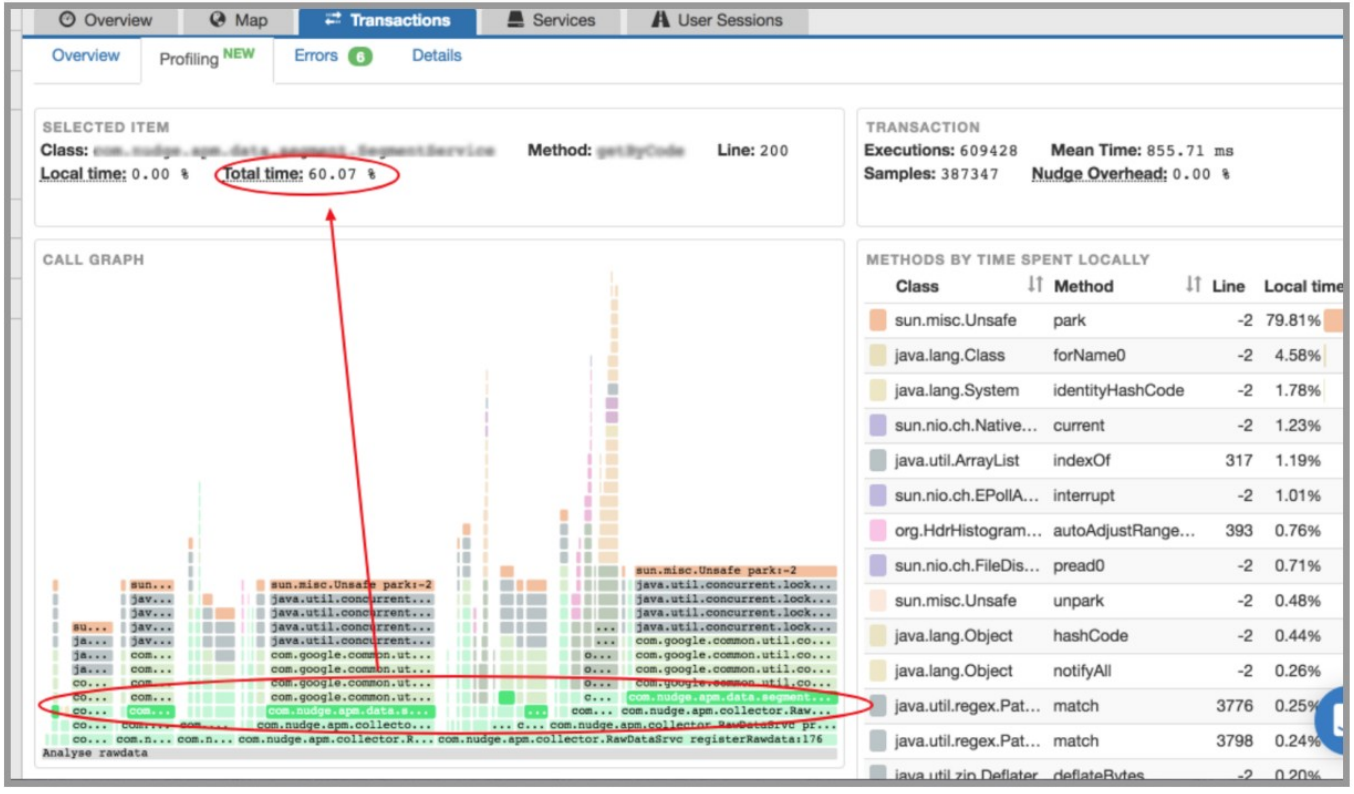
Source: https://blog.rhye.org/post/android-profiling-flamegraphs/ (Ross Schlaikjer)

Java: grav (heap allocations; 2017)



Source: https://epickrram.blogspot.com/2017/09/heap-allocation-flamegraphs.html (Mark Price)

Nudge: APM (for Java; 2017)

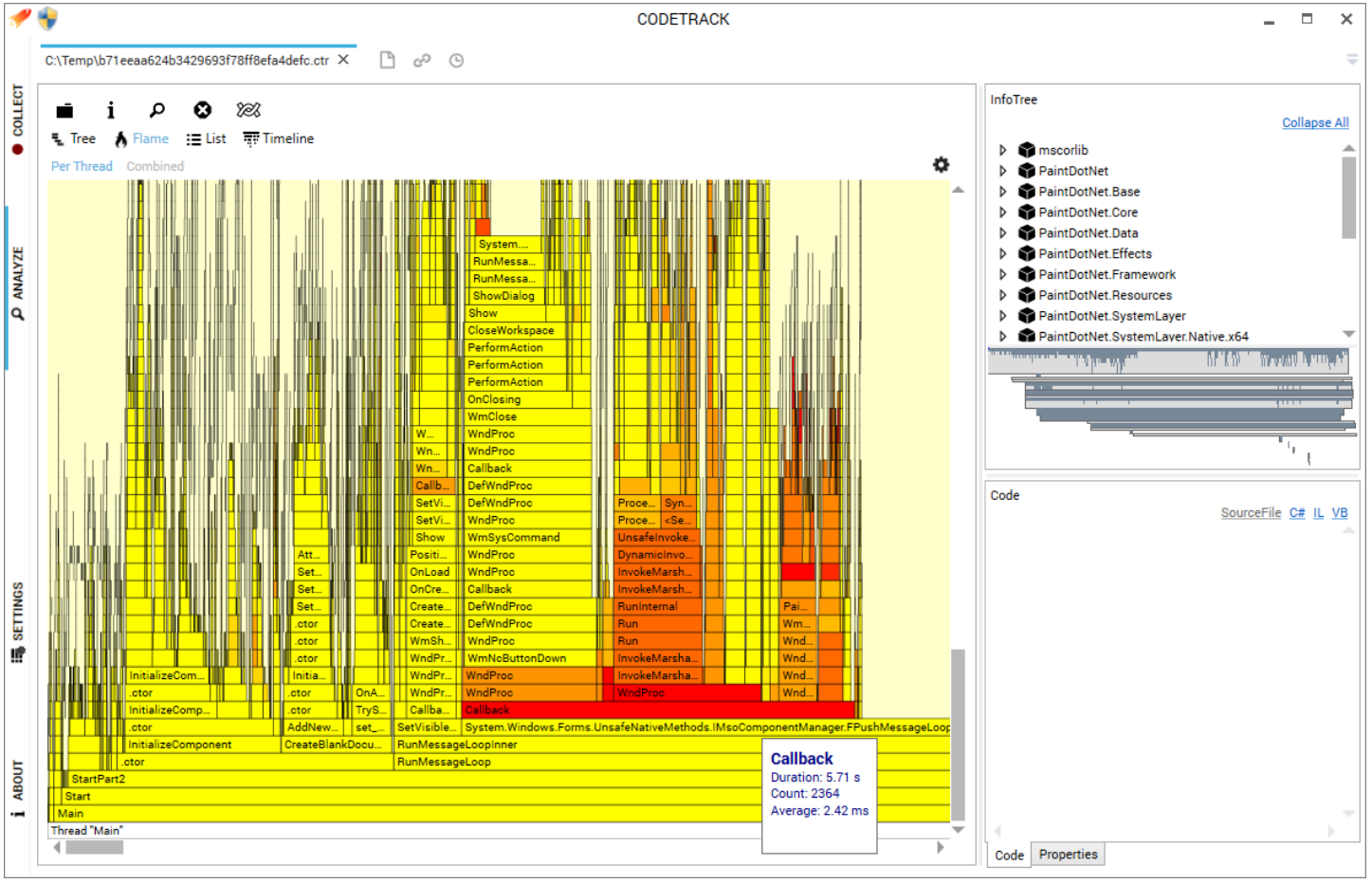


Source: https://nudge-apm.com/features/#profiling

Java: clj-async-profiler (2017)

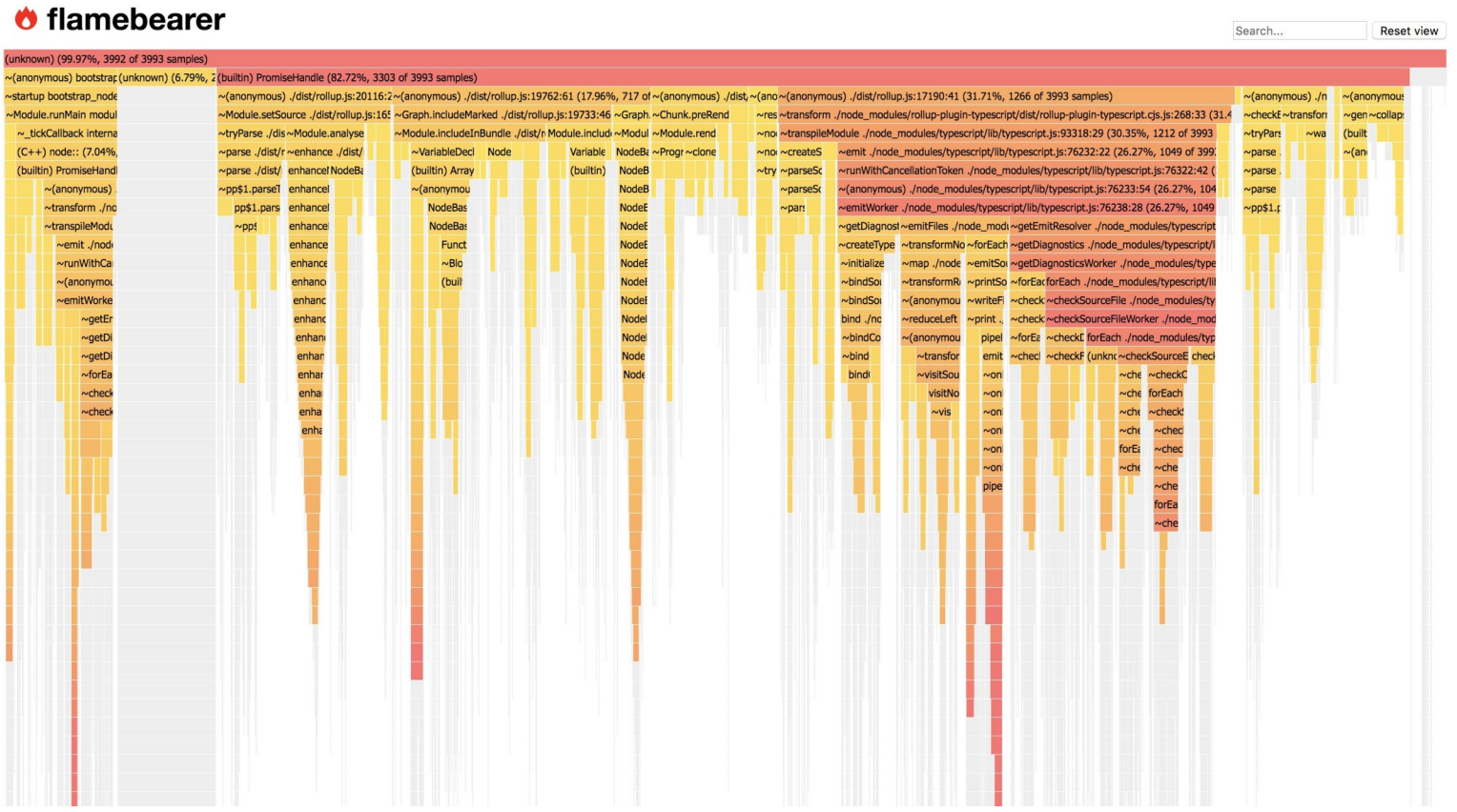
Source: http://clojure-goes-fast.com/blog/profiling-tool-async-profiler/ (Alexander Yakushev)

.NET: codetrack (2017)



Source: https://www.getcodetrack.com/

Node.js: Flamebearer (2018)

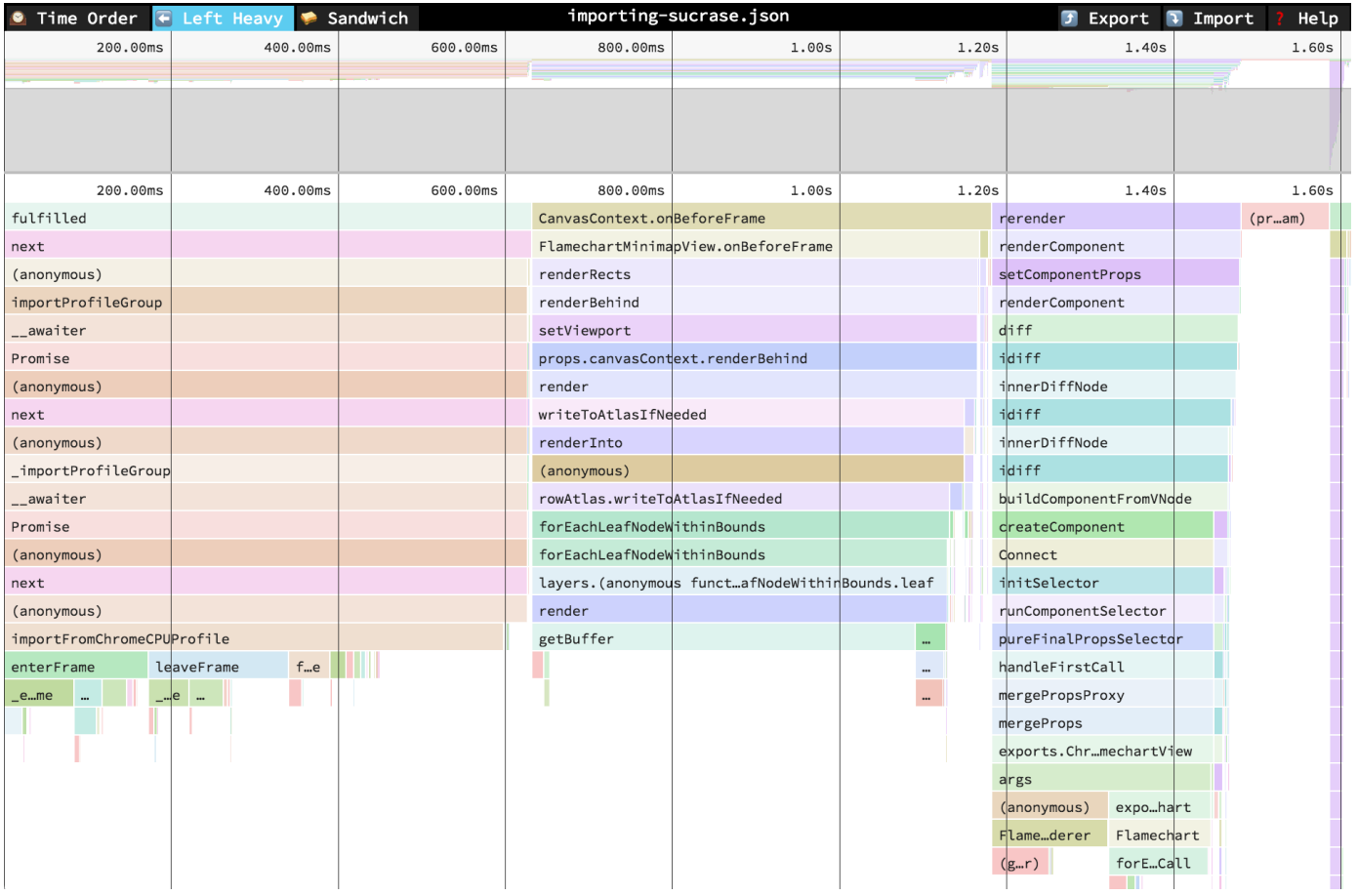


Source: https://github.com/mapbox/flamebearer (Volodymyr Agafonkin)

Opsian: always-on flame graphs (2018)

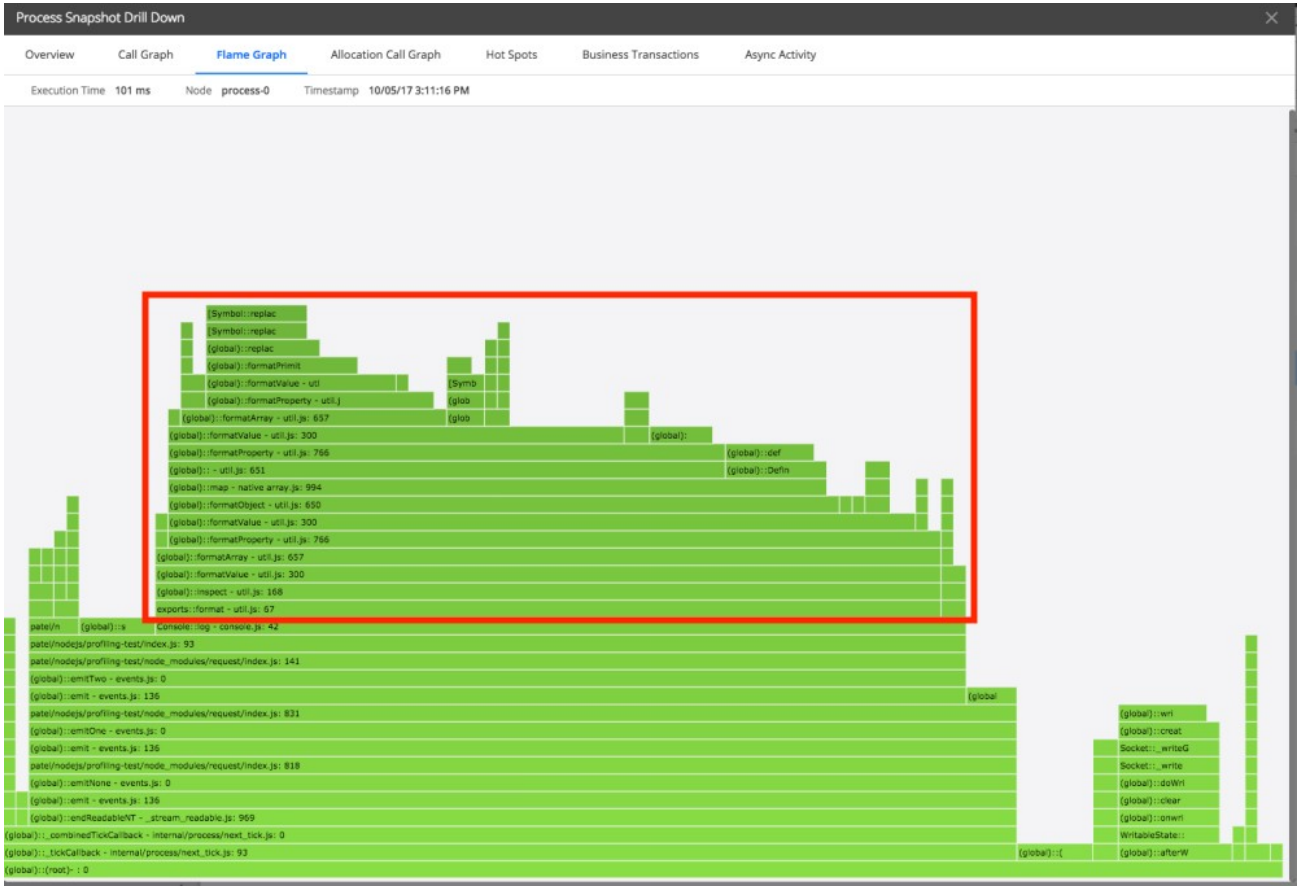
Source: https://www.opsian.com/blog/always-on-production-flame-graphs/

Speedscope: left heavy view (2018)



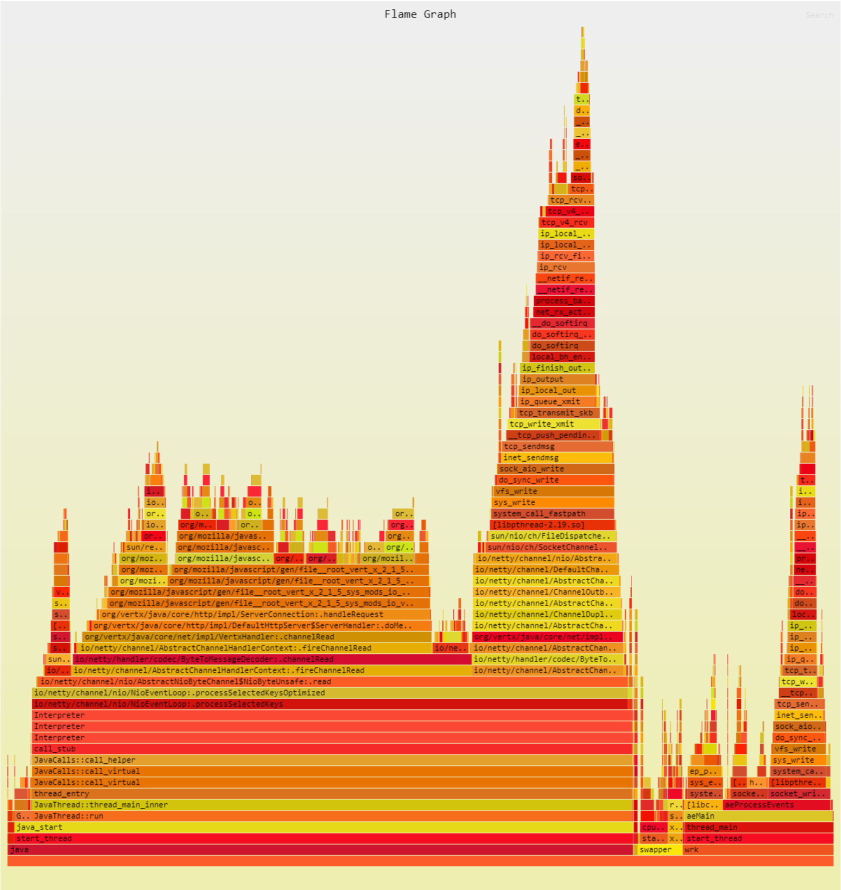
Source: https://jamie-wong.com/post/speedscope/ (Jamie Wong)

AppDynamics: flame graph (2018; now Cisco)



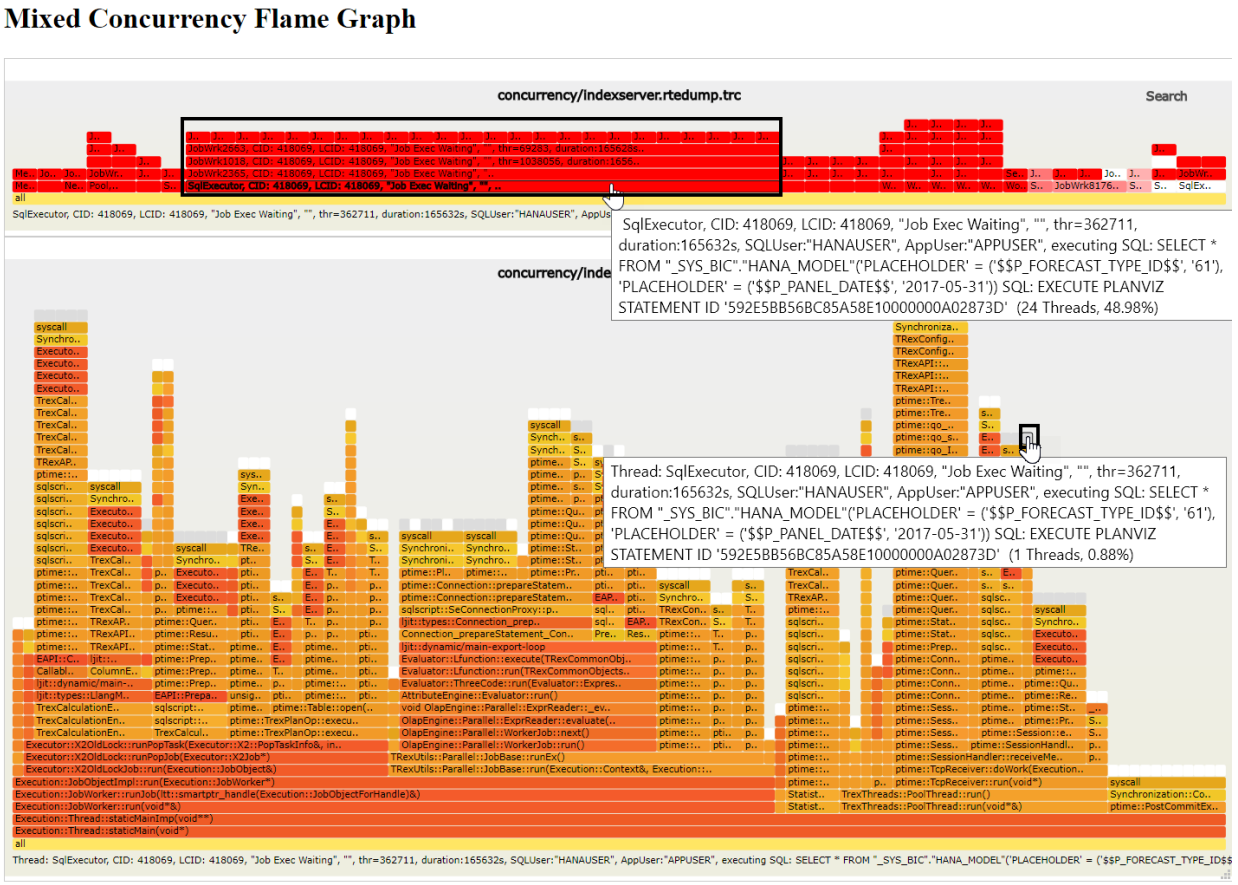
Source: https://docs.appdynamics.com/appd/20.x/en/application-monitoring/troubleshooting-applications/event-loop-blocking-in-node-js#EventLoopBlockinginNode.js-FlameGraph

Inferno: flame graph (Rust port; 2019)



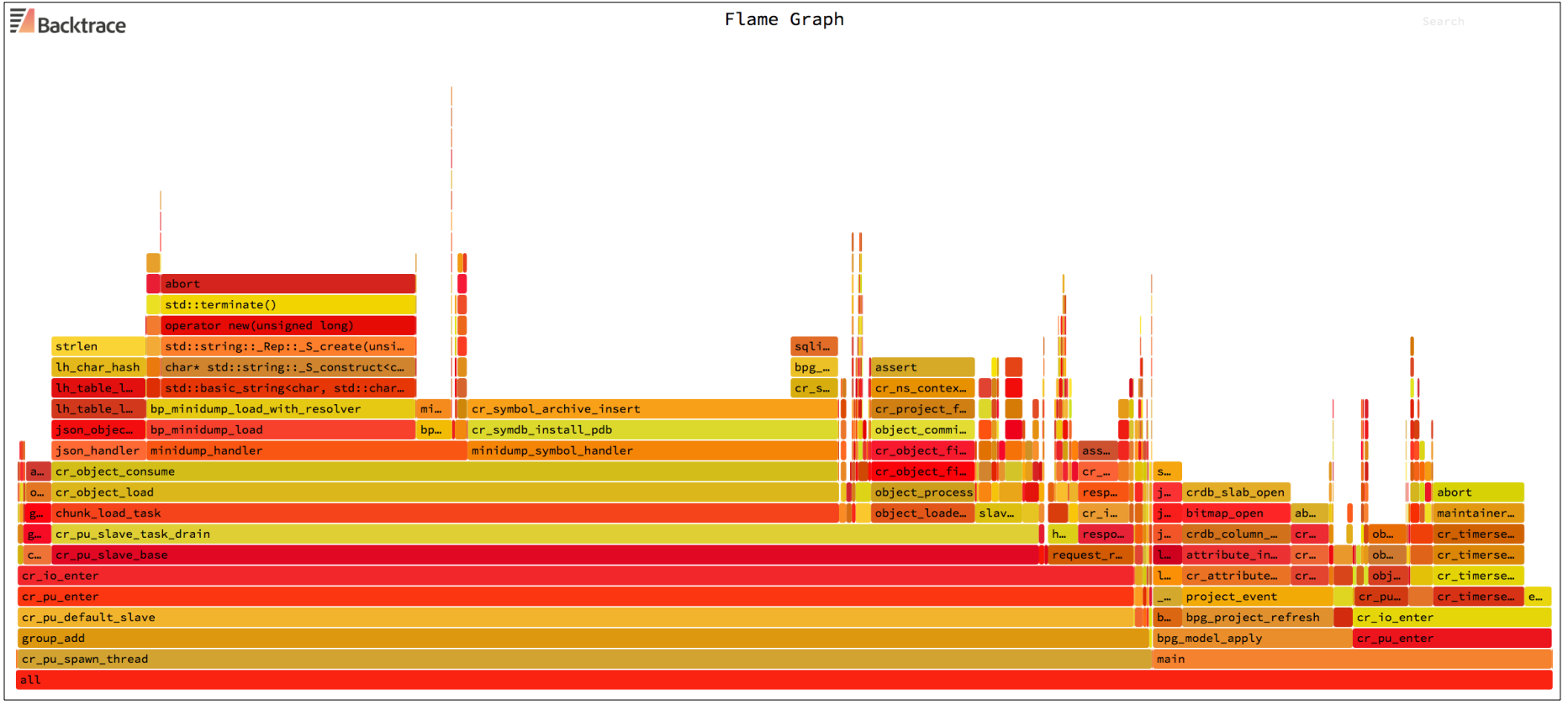
Source: https://github.com/jonhoo/inferno (Jon Gjengset)

SAP: HANA Dump Analyzer (2019)



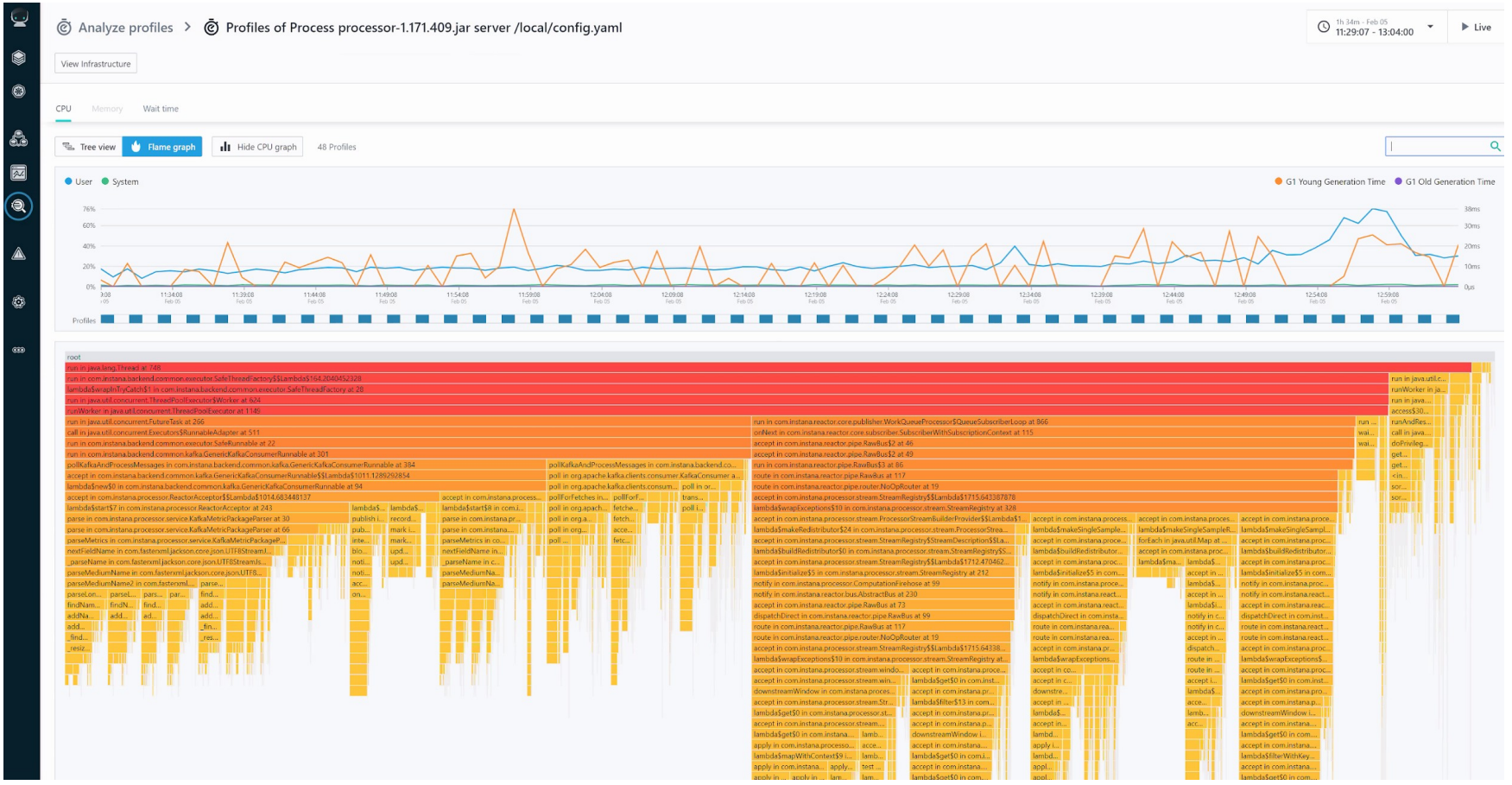
Source: https://blogs.sap.com/2019/04/22/visualizing-olap-requests-on-sap-hana-system-with-concurrency-flame-graph-using-sap-hana-dump-analyzer/

Backtrace: flame graph (2019)



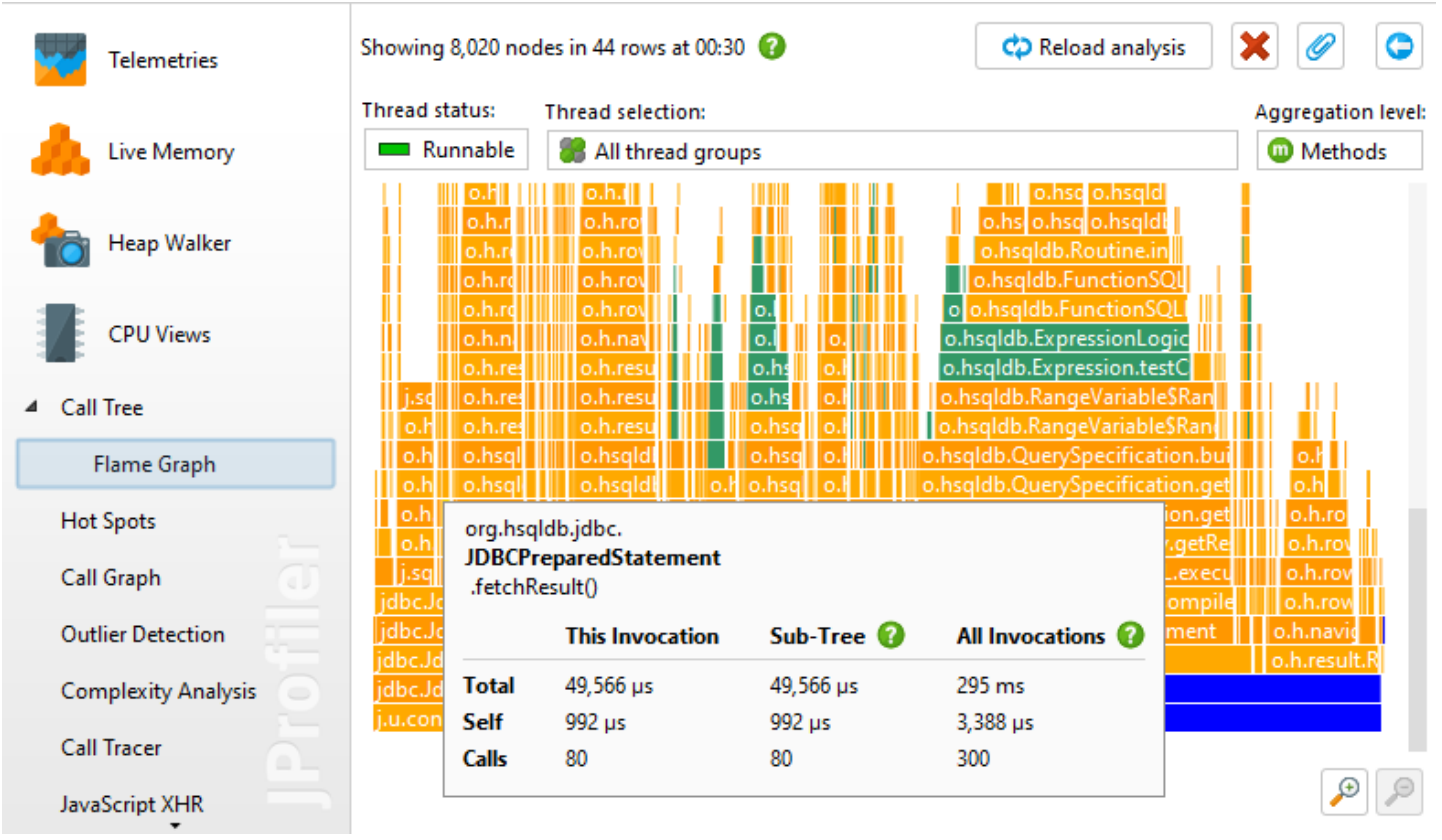
Source: https://support.backtrace.io/hc/en-us/articles/360040515971-Flame-graphs

Instana: flame graph (2020; now IBM)



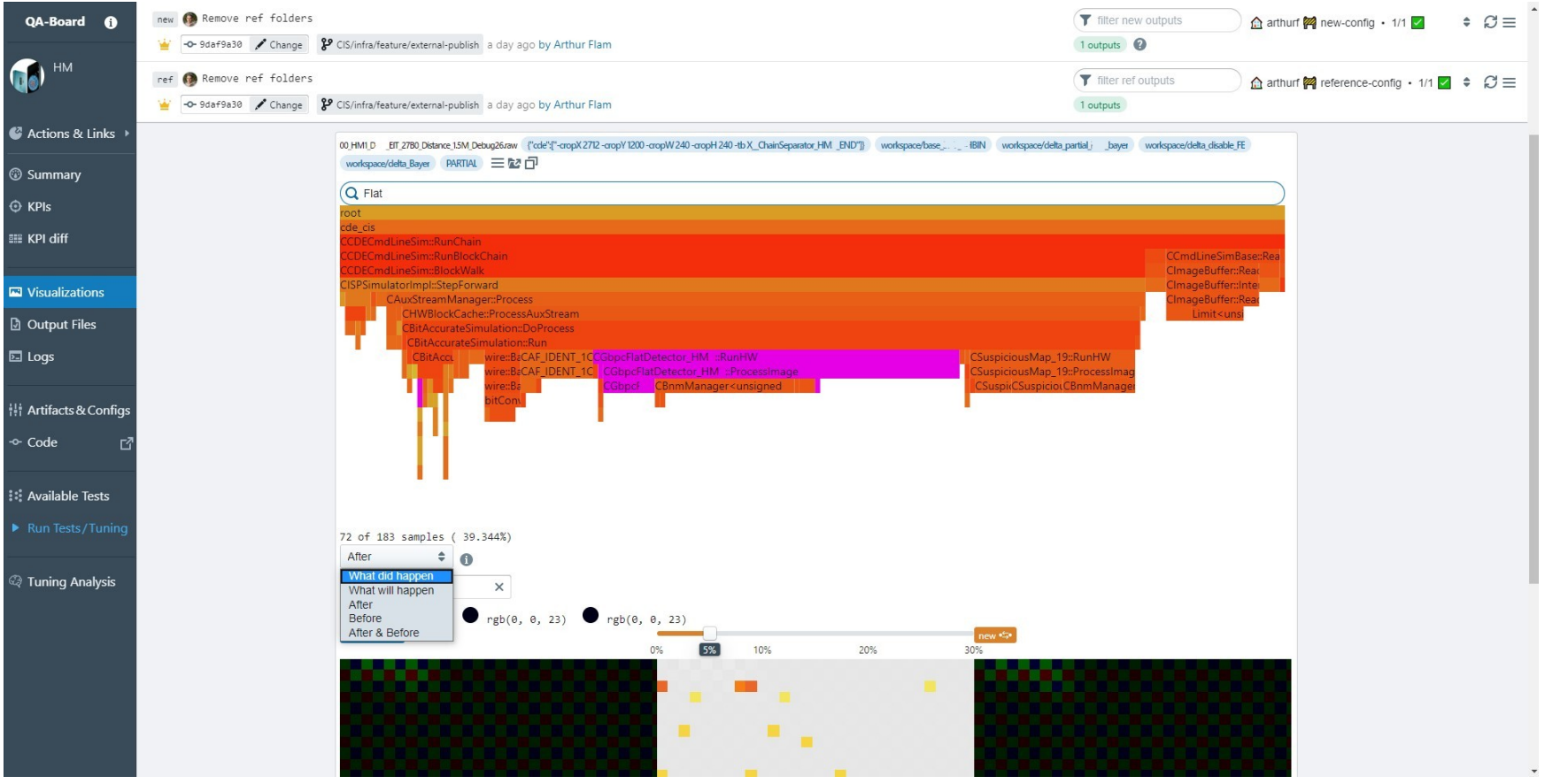
Source: https://www.ibm.com/docs/en/instana-observability/current?topic=processes-analyzing-profiles

ej-technologies: JProfiler Flame Graph (for Java; 2020)



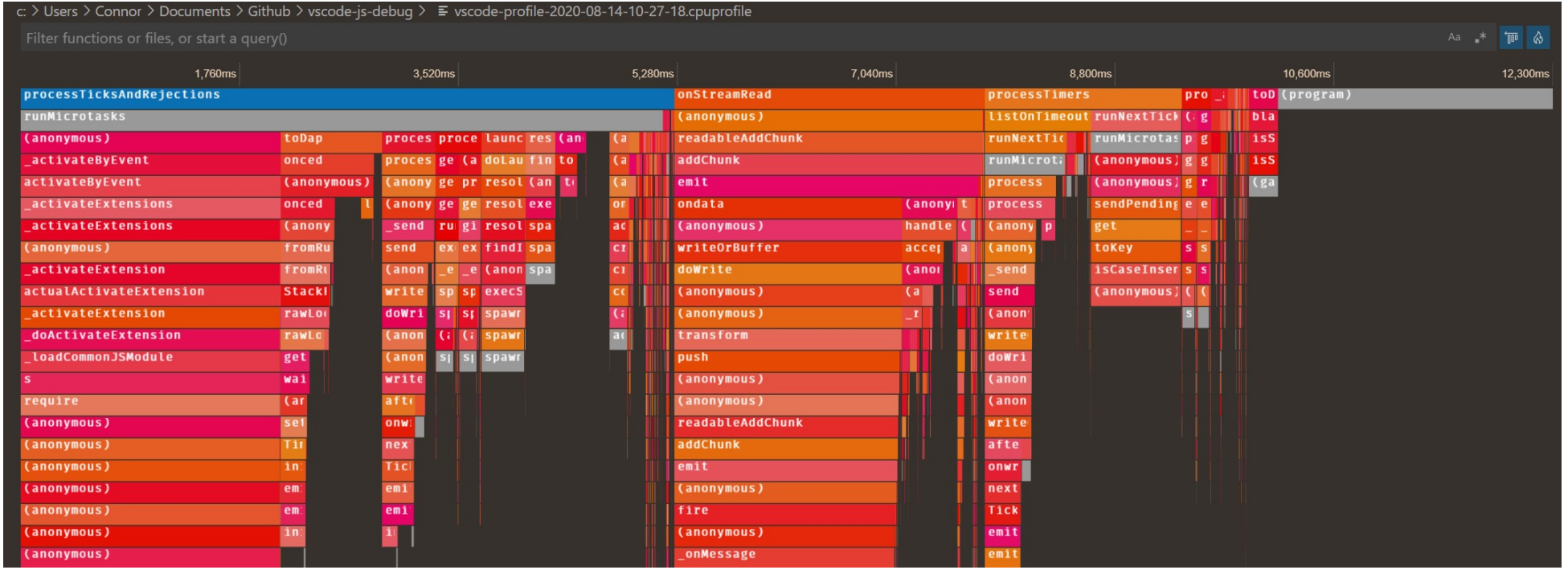
Source: https://www.ej-technologies.com/resources/jprofiler/help/doc/main/cpu.html

Samsung: QA-Board (2020)



Source: https://samsung.github.io/qaboard/blog/2020/06/24/flame-graphs

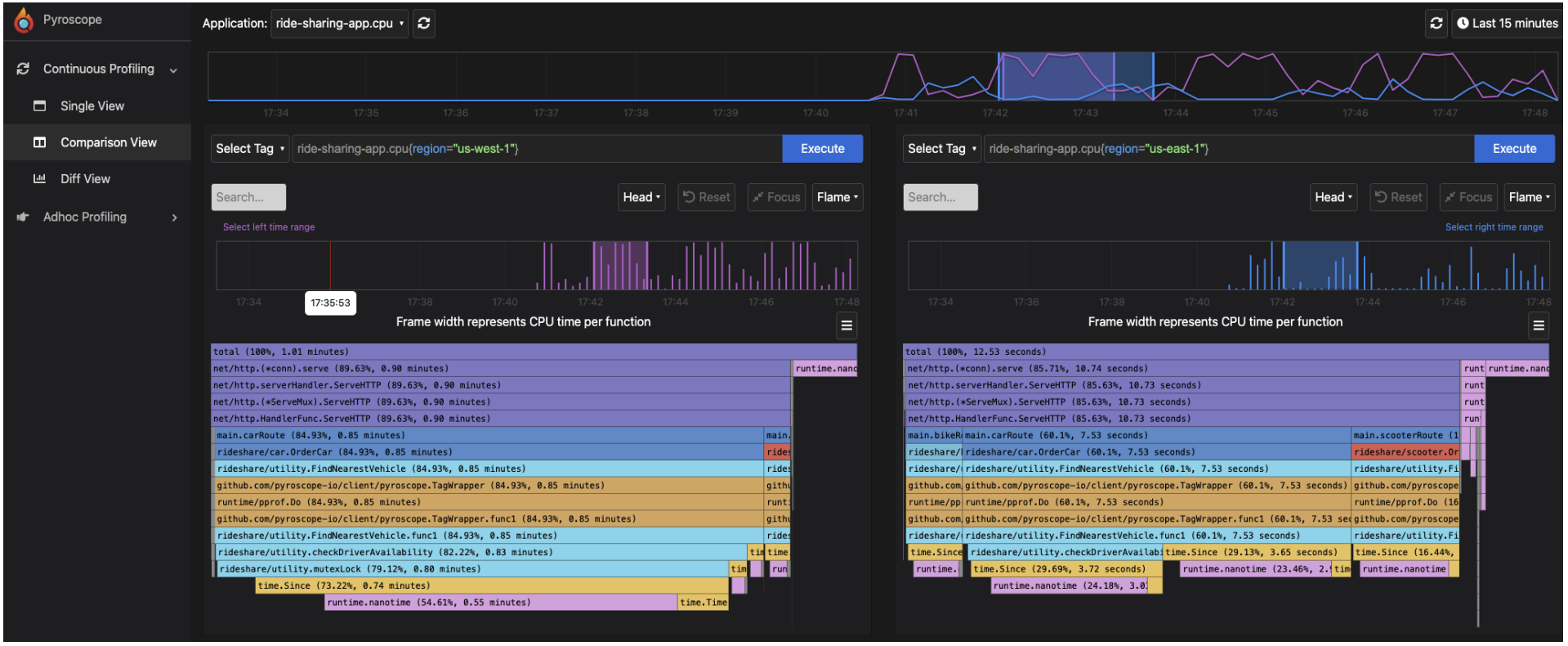
Microsoft Visual Studio: vscode-js-profile-flame (for JavaScript; 2020)



Left Heavy view

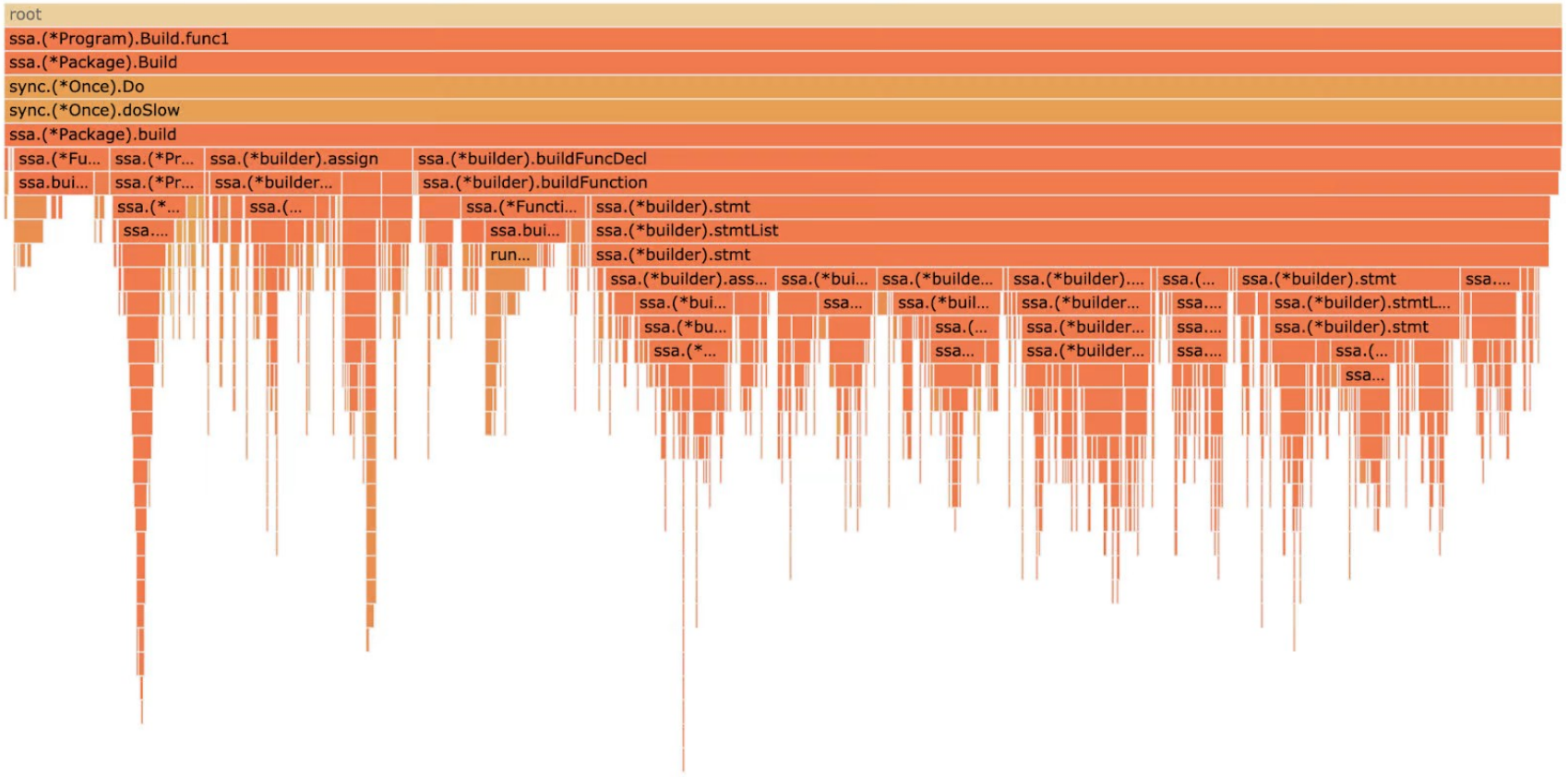
Source: https://marketplace.visualstudio.com/items?itemName=ms-vscode.vscode-js-profile-flame

Pyroscope: flame graph (2020)



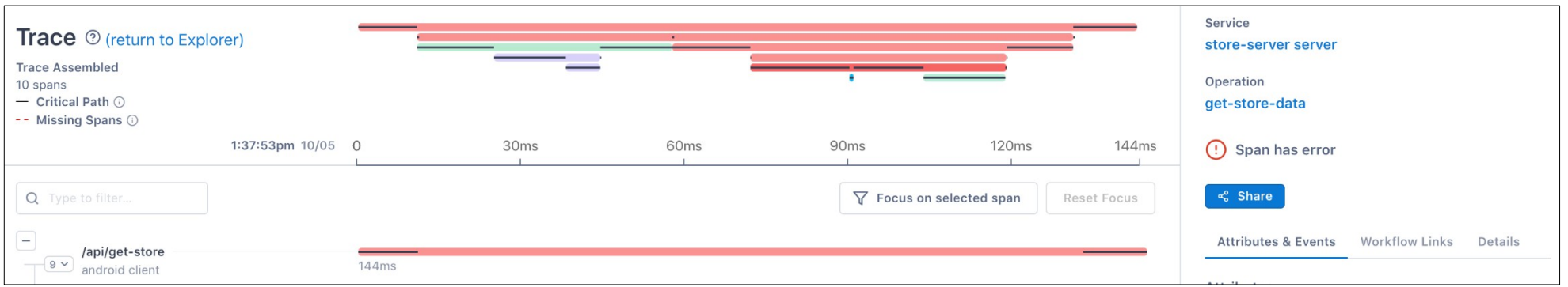
Source: https://pyroscope.io/blog/what-is-a-flamegraph/

Uber: pprof++ (2021; for Golang)



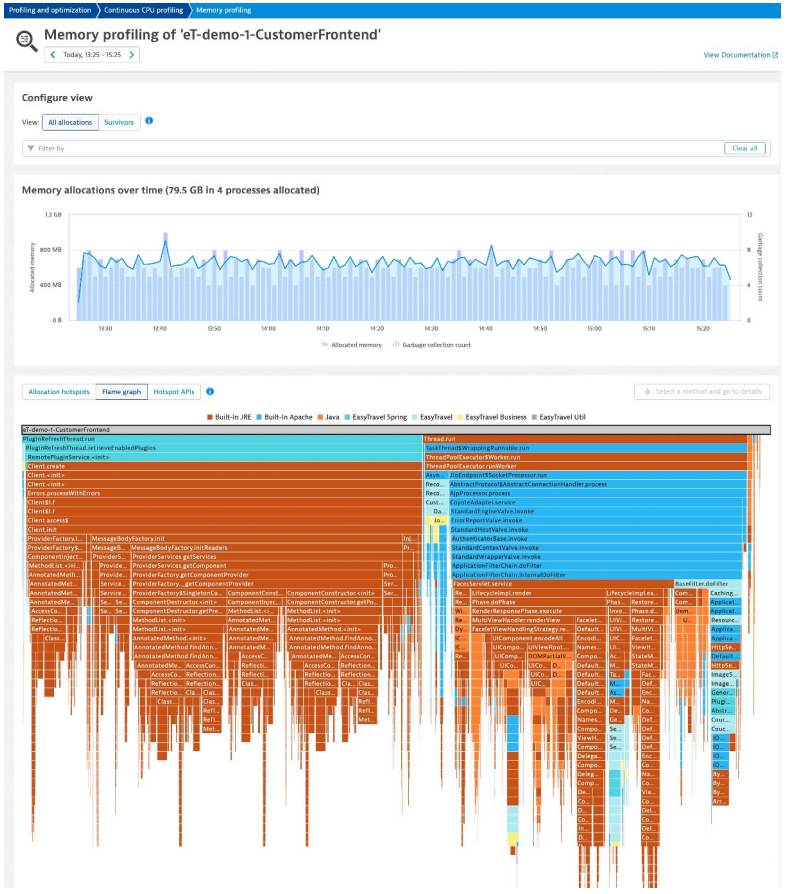
Source: https://www.uber.com/en-AU/blog/pprof-go-profiler/ (Pengfei Su)

Lightstep: flame graph (2021)



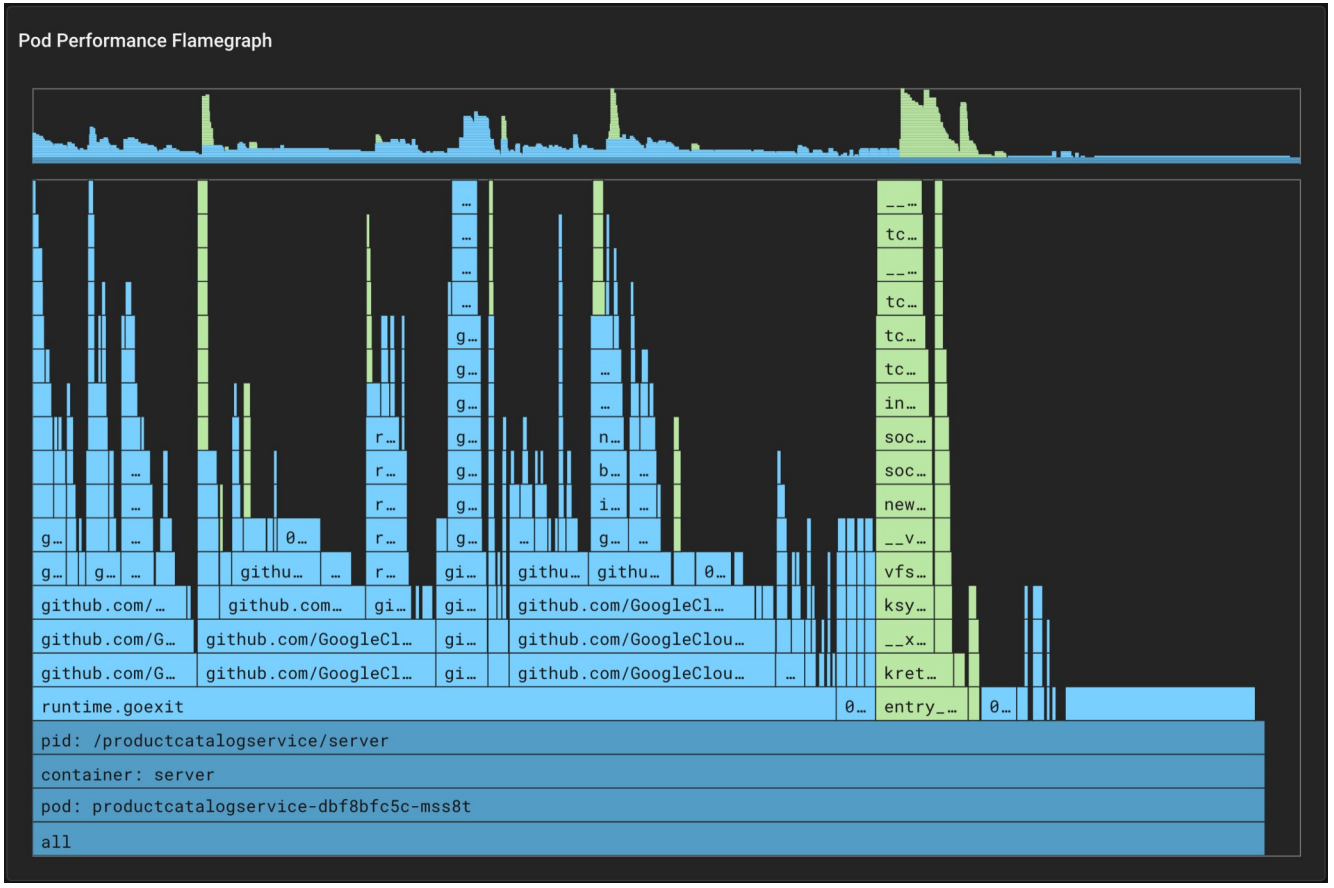
Source: https://www.instana.com/blog/instana-announces-the-industrys-first-commercial-continuous-production-profiler/

Dynatrace: allocation flame graph (2021)



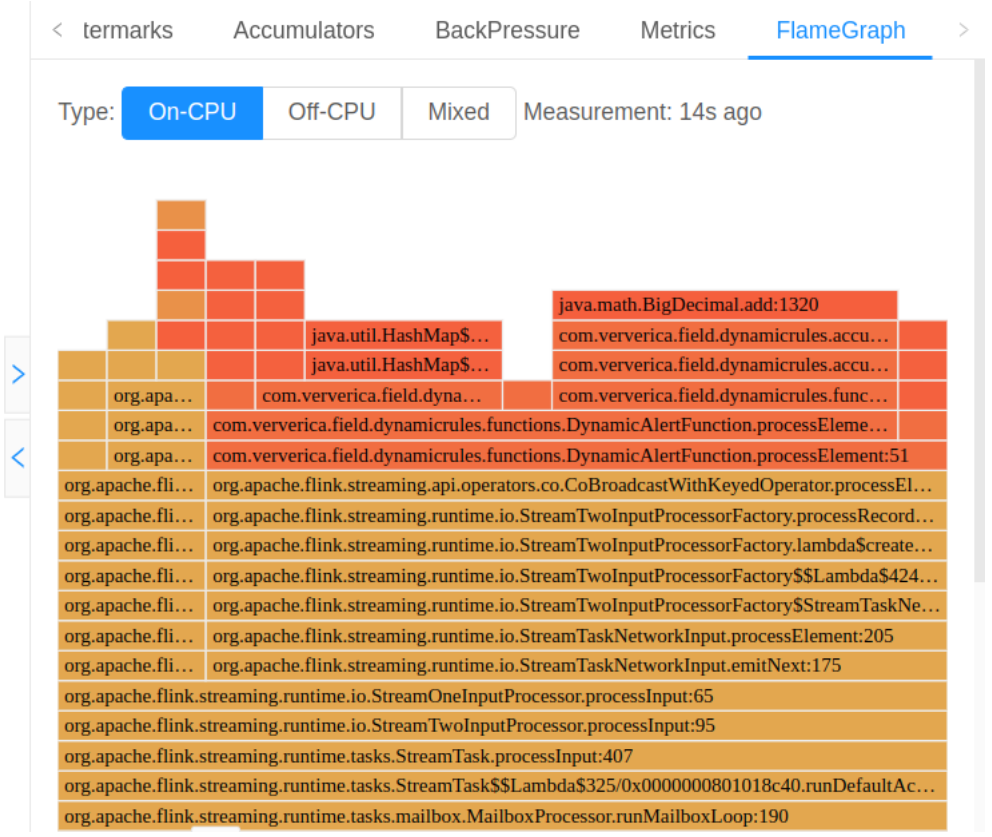
Source: https://www.dynatrace.com/support/help/how-to-use-dynatrace/diagnostics/memory-profiling

Pixie Labs: pod performance flamegraph (2021)



Source: https://docs.pixielabs.ai/tutorials/pixie-101/profiler/

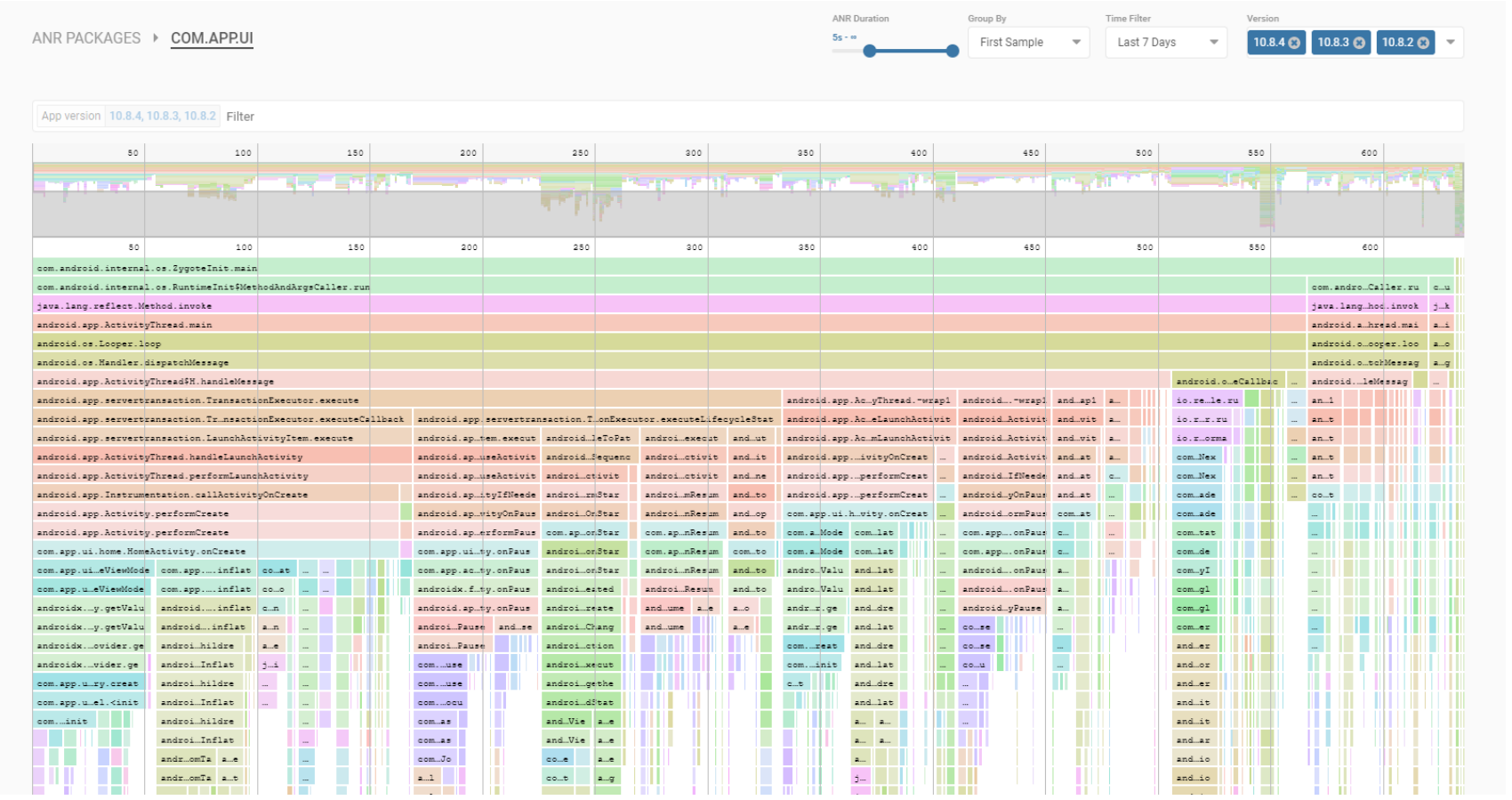
Apache Flink: flame graphs (2021)



off-CPU

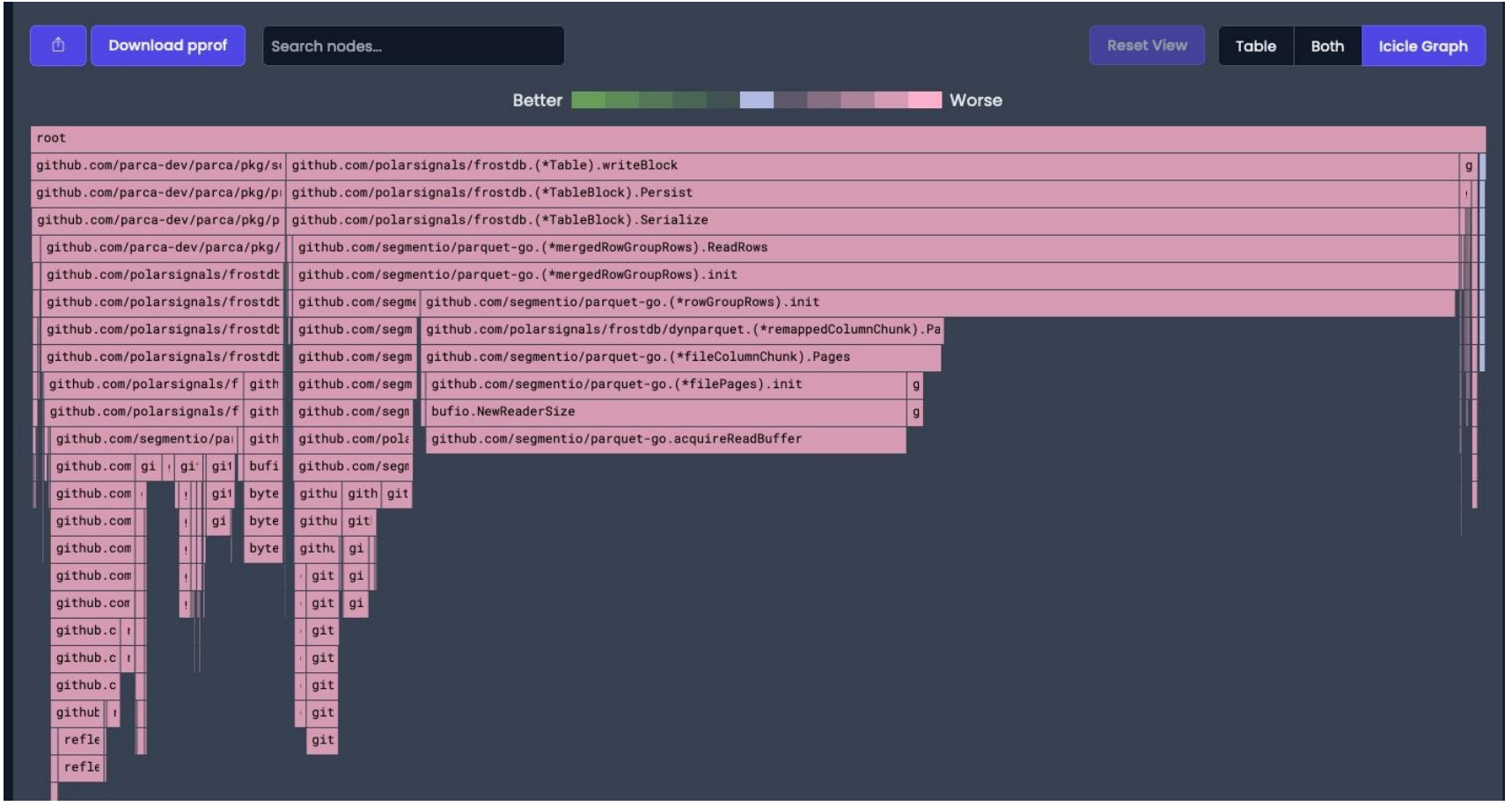
Source: https://nightlies.apache.org/flink/flink-docs-master/docs/ops/debugging/flame_graphs/

Embrace: Application-Not-Responding flame graph (2021)



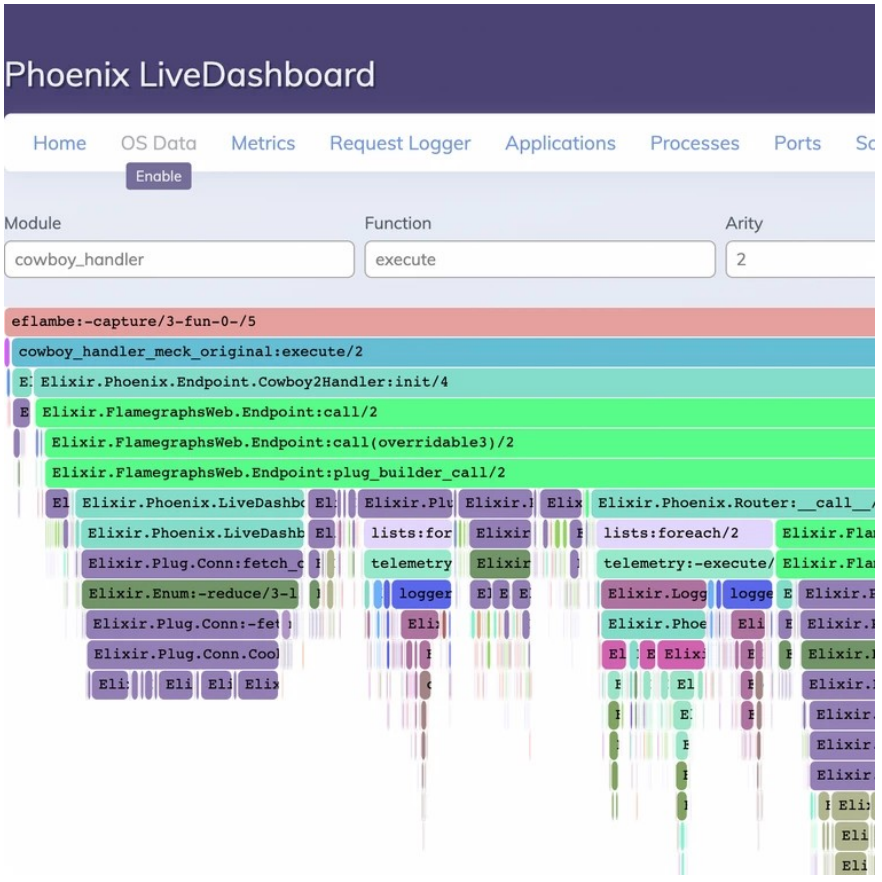
Source: https://blog.embrace.io/solve-anrs-with-flame-graphs/

Polar Signals: parca Continuous Profiling (2021)



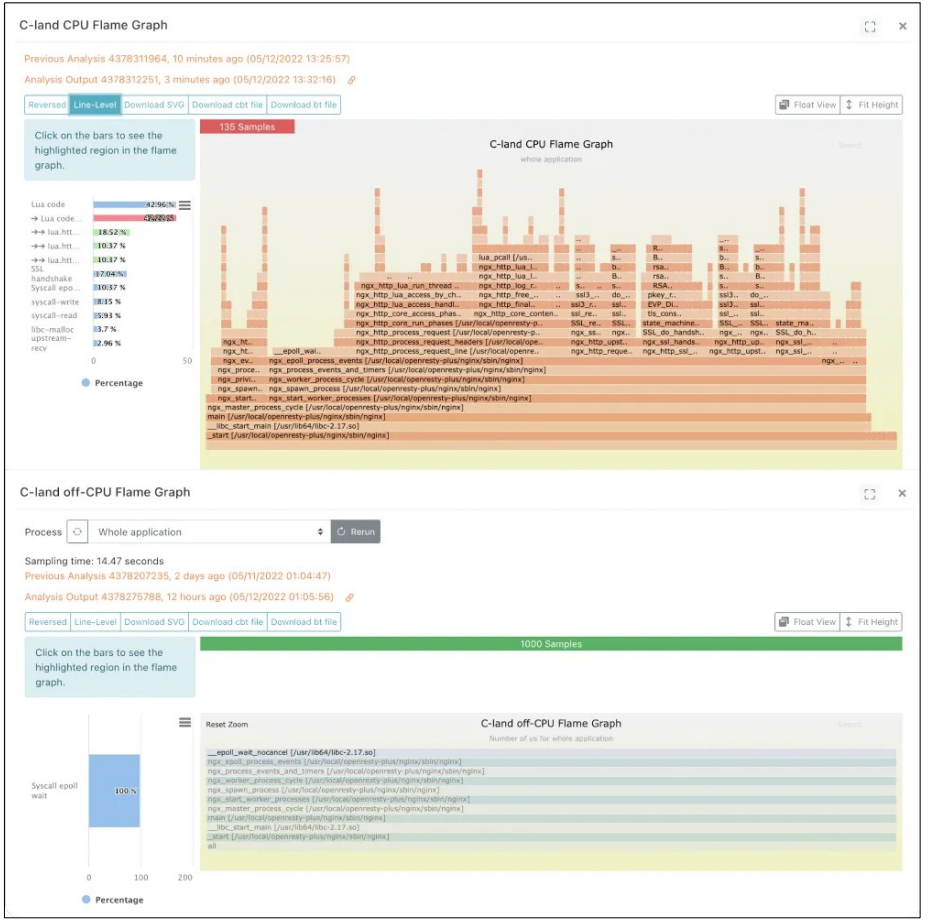
Source: https://www.polarsignals.com/blog/posts/2022/08/30/optimizing-with-continuous-profiling/

Dockyard: Flame On (for Elixir apps; 2022)



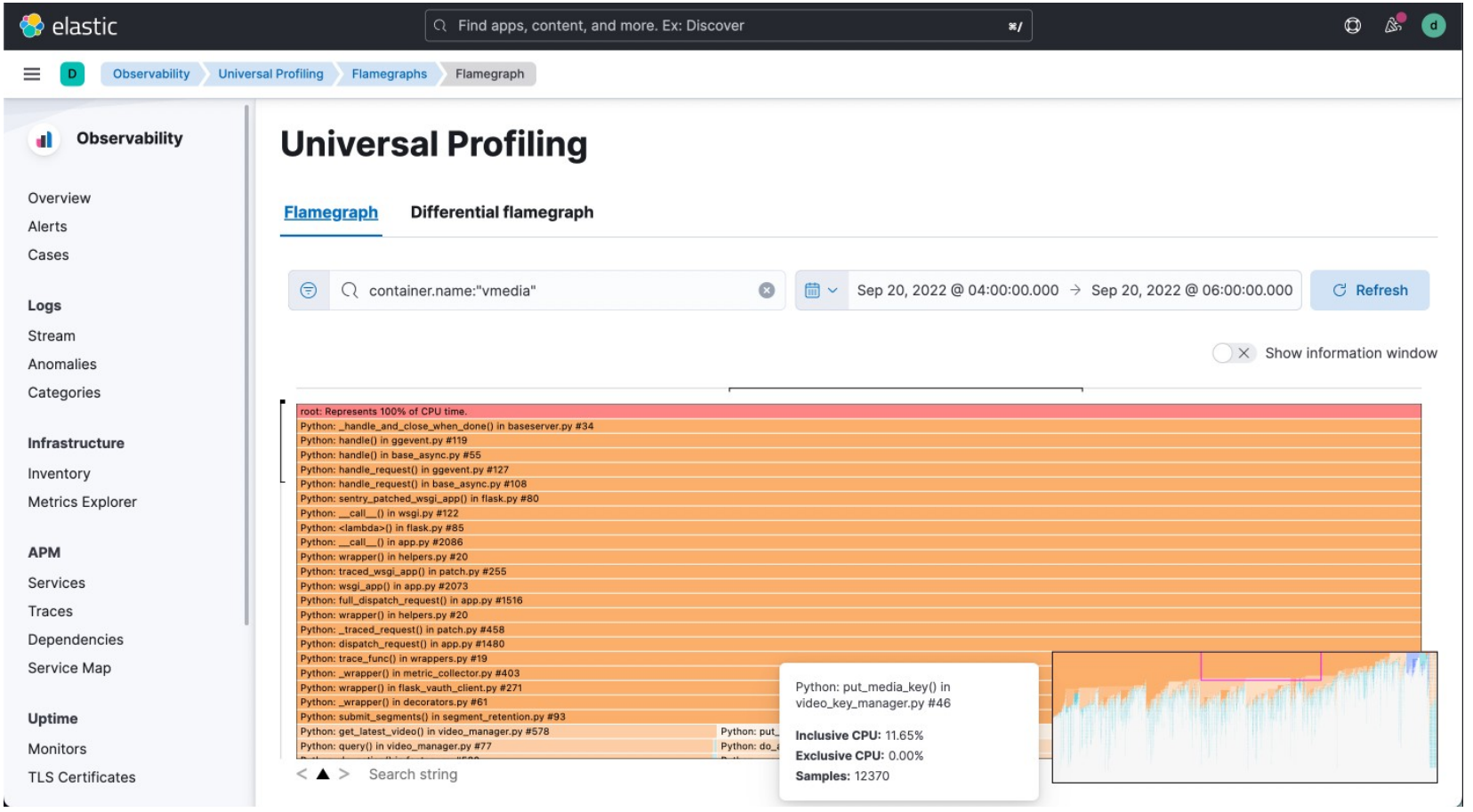
Source: https://dockyard.com/blog/2022/02/22/profiling-elixir-applications-with-flame-graphs-and-flame-on (Mike Binns)

OpenResty: Xray (2022)



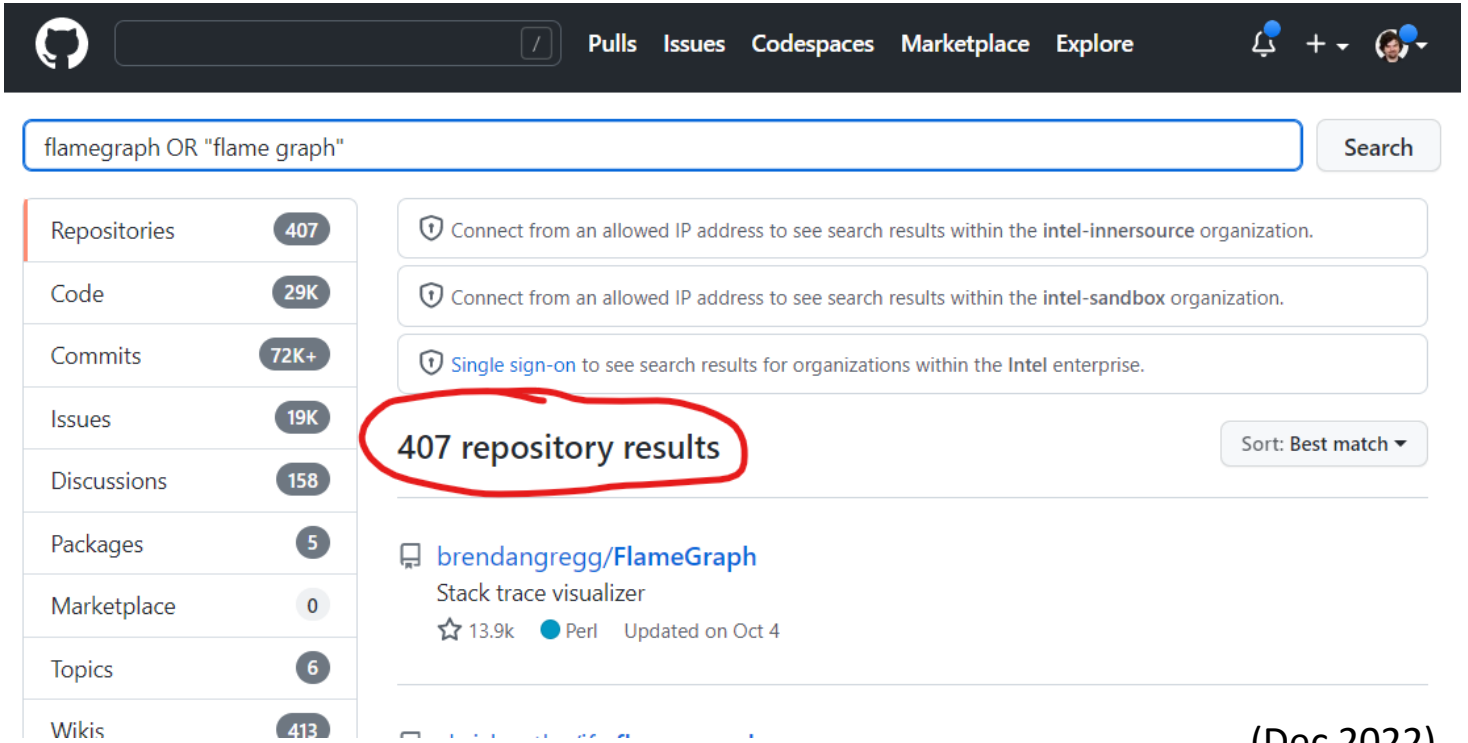
Source: https://openresty.com/en/xray (Yichun Zhang)

Elastic: universal profiling (2022)



Source: https://www.elastic.co/observability/universal-profiling

… and more



(Dec 2022)

Thanks for all the open source contributions!