2017 USENIX Annual Technical Conference

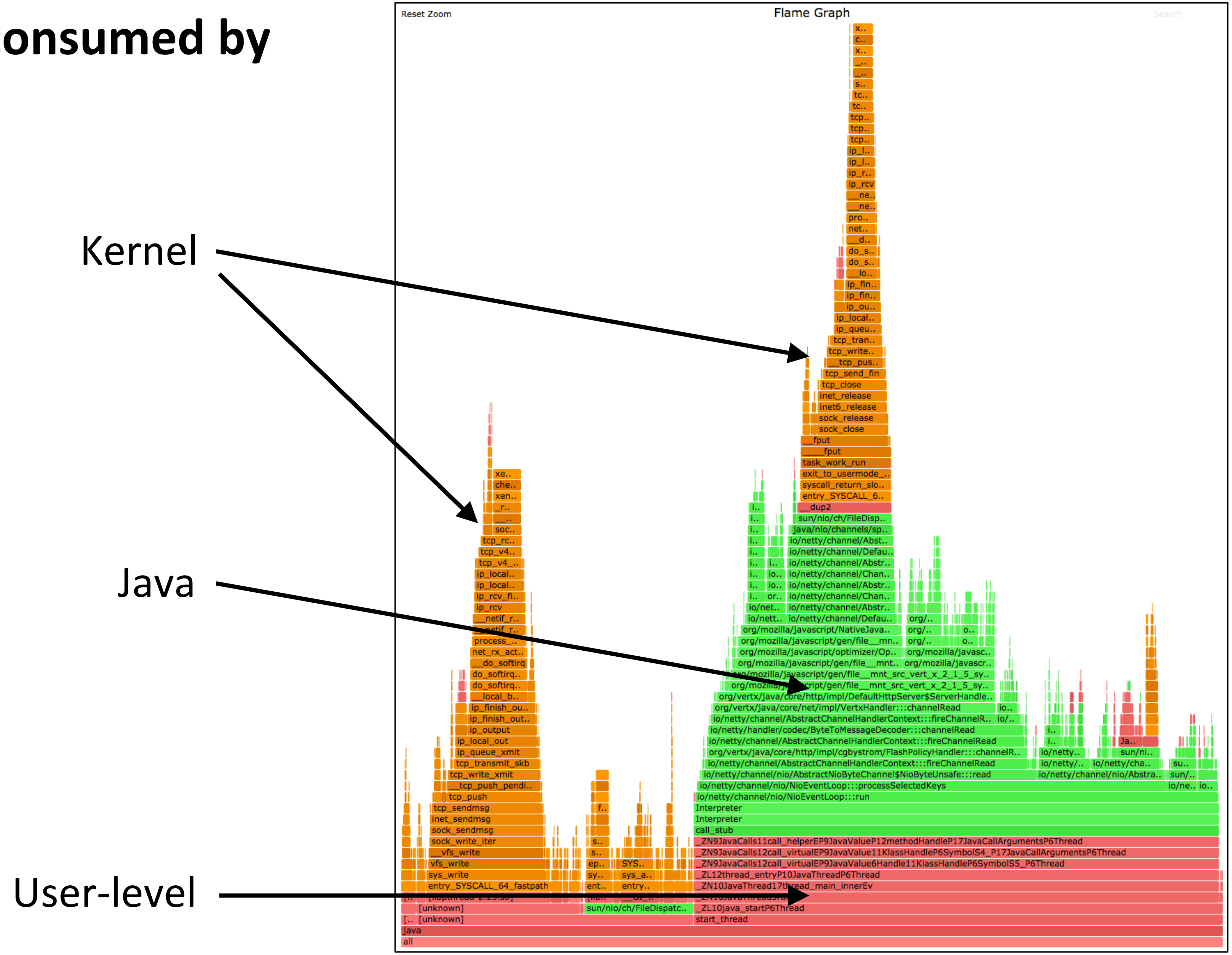
Visualizing Performance with Flame Graphs

Brendan Gregg

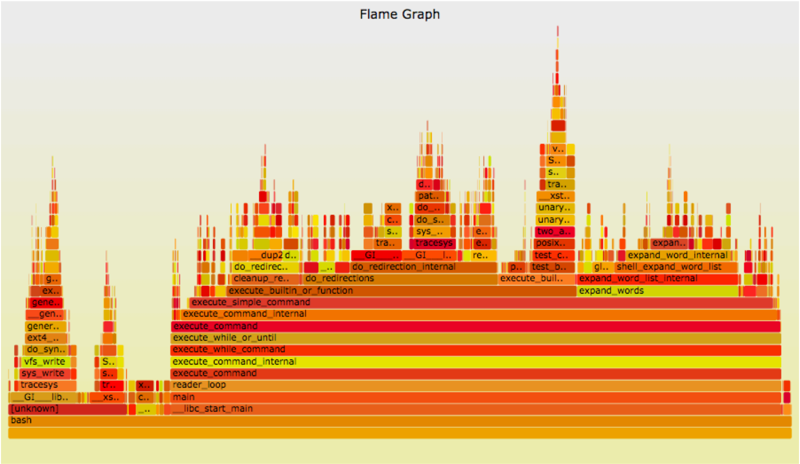
Senior Performance Architect

Jul 2017

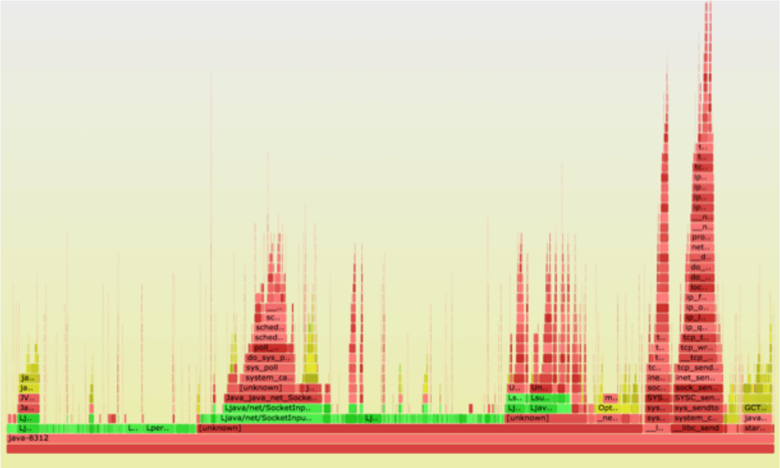
Visualize CPU time consumed by all software



Agenda



1. CPU Flame graphs



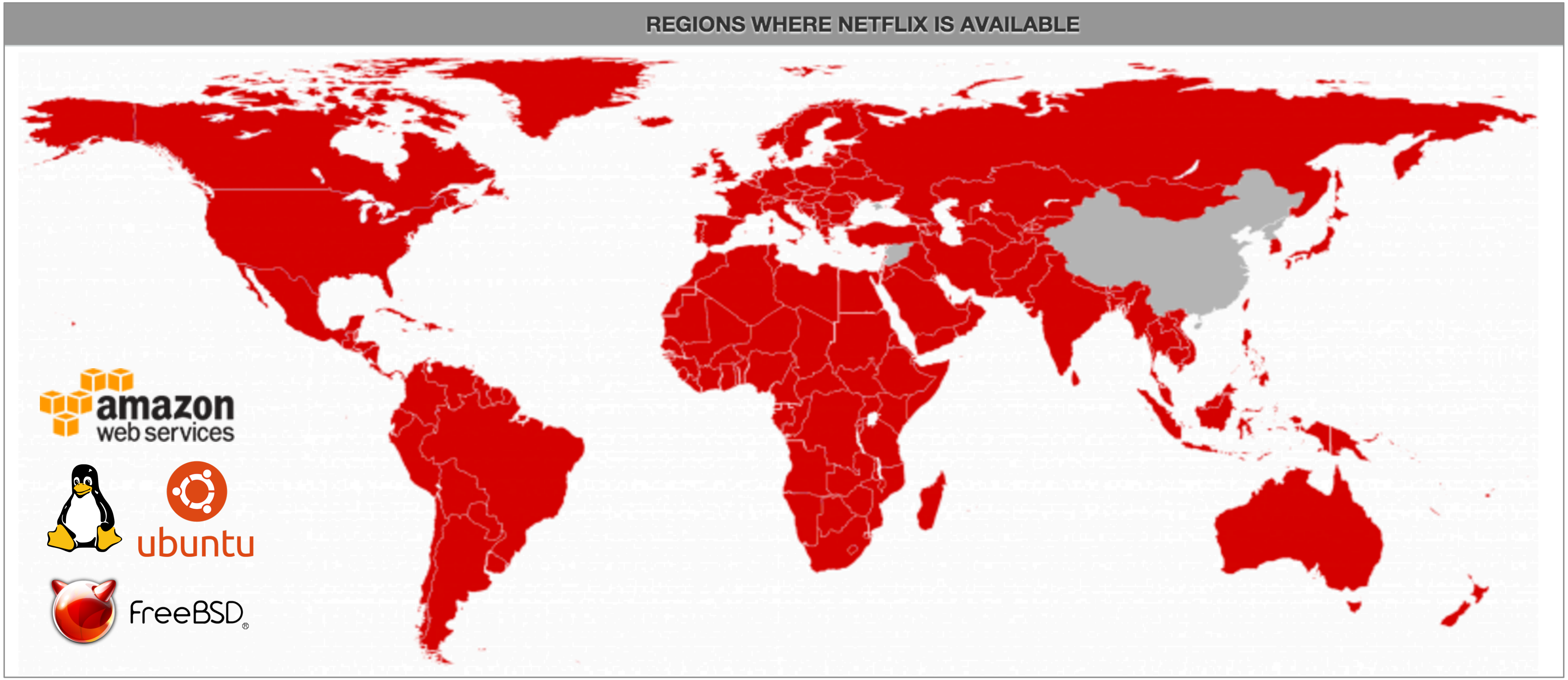
2. Fixing Stacks & Symbols

3. Advanced flame graphs

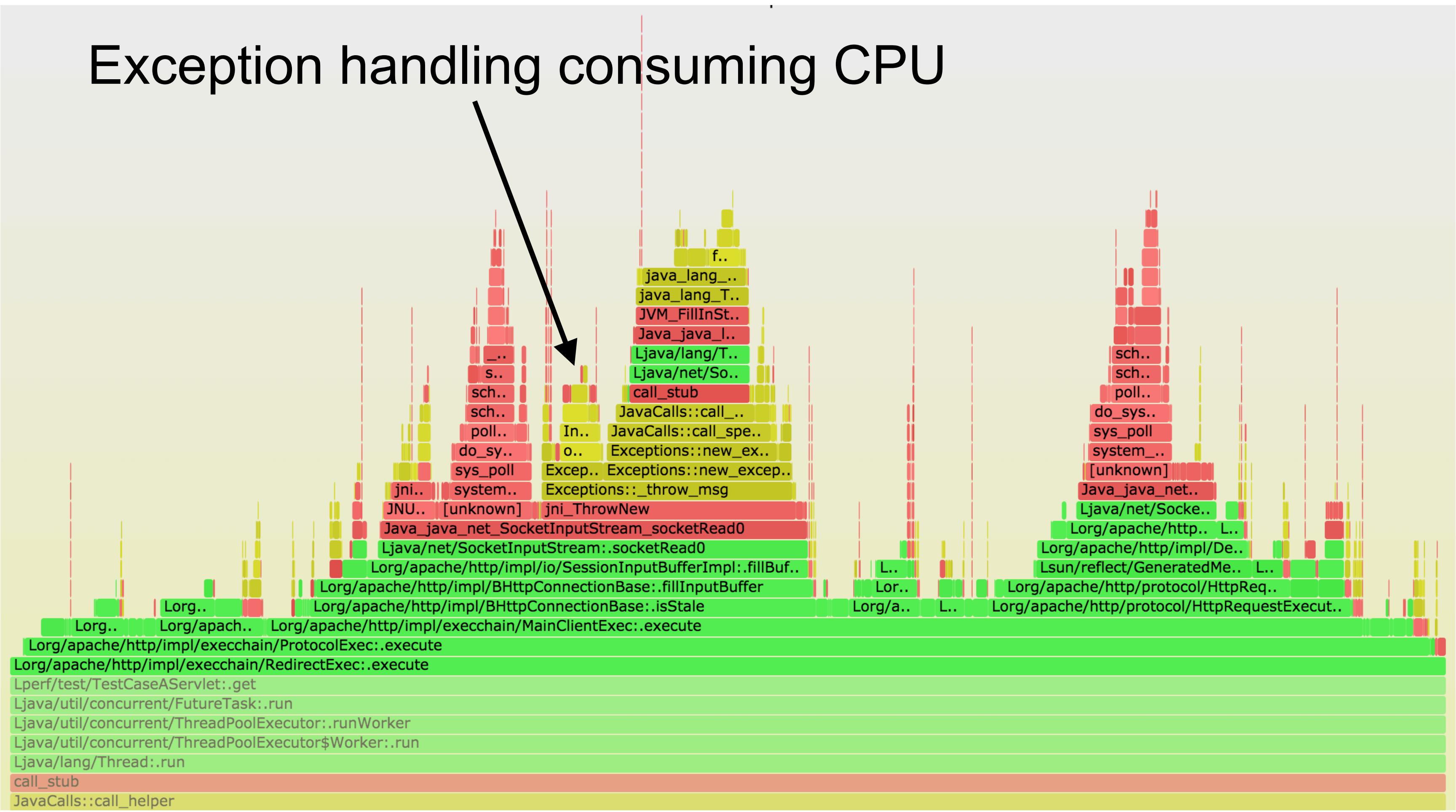
Take aways

1. Interpret CPU flame graphs
2. Understand pitfalls with stack traces and symbols
3. Discover opportunities for future development

NETFLIX


REGIONS WHERE NETFLIX IS AVAILABLE
amazon
web services
ubuntu
FreeBSD

Case Study



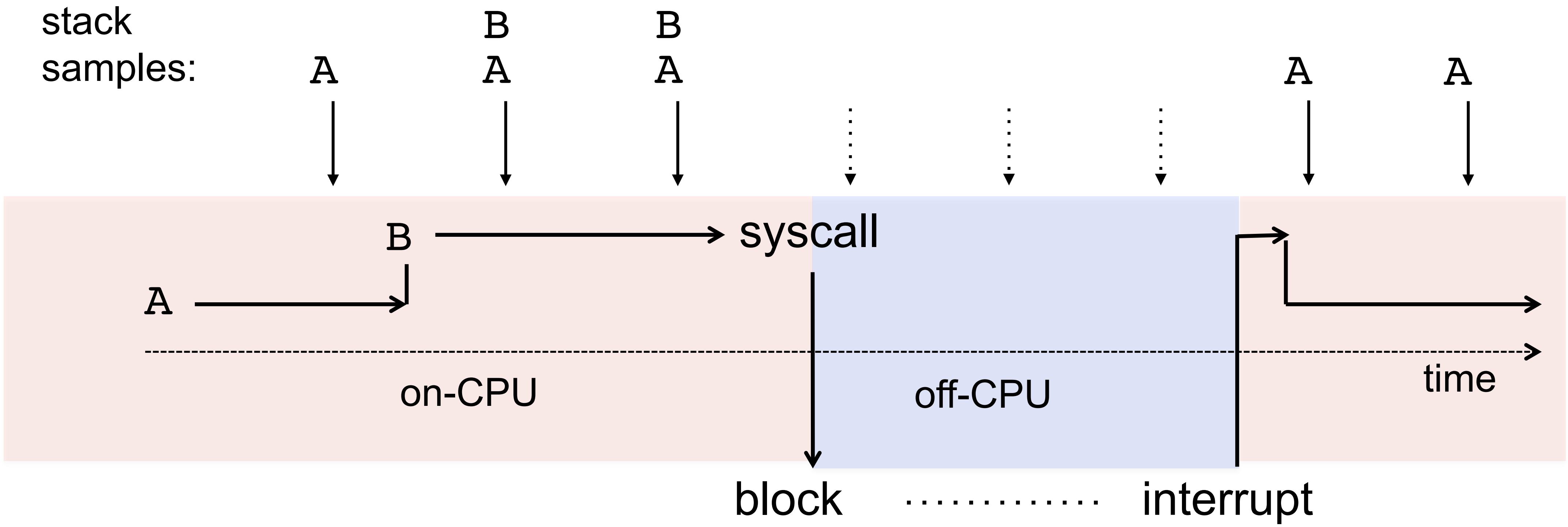
Exception handling consuming CPU
f..
java_lang_..
java_lang_T..
JVM_FillInSt..
Java_java_l..
Ljava/lang/T..
Ljava/net/So..
call_stub
JavaCalls::call_..
JavaCalls::call_spe..
Exceptions::new_ex..
Excep..
Exceptions::new_excep..
Exceptions::_throw_msg
jni_ThrowNew
In..
o..
_..
s..
sch..
sch..
poll..
do_sy..
sys_poll
system..
[unknown]
jni..
JNU..
Java_java_net_SocketInputStream_socketRead0
Ljava/net/SocketInputStream:.socketRead0
Lorg/apache/http/impl/io/SessionInputBufferImpl:.fillBuf..
Lorg/apache/http/impl/BHttpConnectionBase:.fillInputBuffer
Lorg/apache/http/impl/BHttpConnectionBase:.isStale
Lorg..
Lorg..
Lorg/apach..
Lorg/apache/http/impl/execchain/MainClientExec:.execute
Lorg/apache/http/impl/execchain/ProtocolExec:.execute
Lorg/apache/http/impl/execchain/RedirectExec:.execute
Lperf/test/TestCaseAServlet:.get
Ljava/util/concurrent/FutureTask:.run
Ljava/util/concurrent/ThreadPoolExecutor:.runWorker
Ljava/util/concurrent/ThreadPoolExecutor$Worker:.run
Ljava/lang/Thread:.run
call_stub
JavaCalls::call_helper
sch..
sch..
poll..
do_sys..
sys_poll
system_..
[unknown]
Java_java_net..
Ljava/net/Socke..
Lorg/apache/http..
Lorg/apache/http/impl/De..
Lsun/reflect/GeneratedMe..
Lorg/apache/http/protocol/HttpReq..
Lorg/apache/http/protocol/HttpRequestExecut..
L..
Lor..
Lorg/a..

Summary

CPU PROFILING

CPU Profiling

- Record stacks at a timed interval: simple and effective
  - Pros: Low (deterministic) overhead
  - Cons: Coarse accuracy, but usually sufficient



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

System Profilers

- Linux
  - perf_events (aka "perf")
- Oracle Solaris
  - DTrace
- OS X
  - Instruments
- Windows
  - XPerf, WPA (which now has flame graphs!)
- And many others…

Linux perf_events

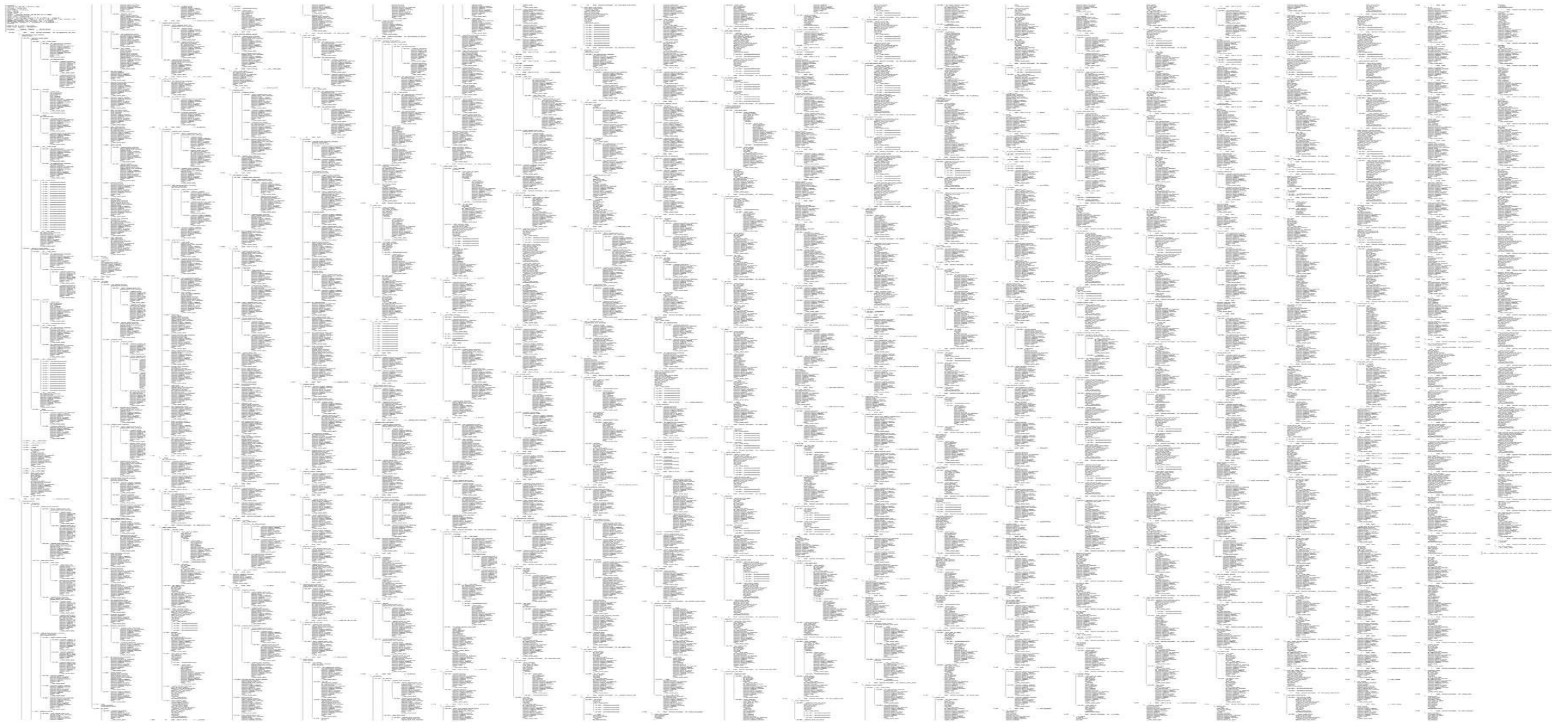
- Standard Linux profiler
  - Provides the `perf` command (multi-tool)
  - Usually pkg added by linux-tools-common, etc.
- Many event sources:
  - Timer-based sampling
  - Hardware events
  - Tracepoints
  - Dynamic tracing
- Can sample stacks of (almost) everything on CPU
  - Can miss hard interrupt ISRs, but these should be near-zero. They can be measured if needed (I wrote my own tools).

perf Profiling

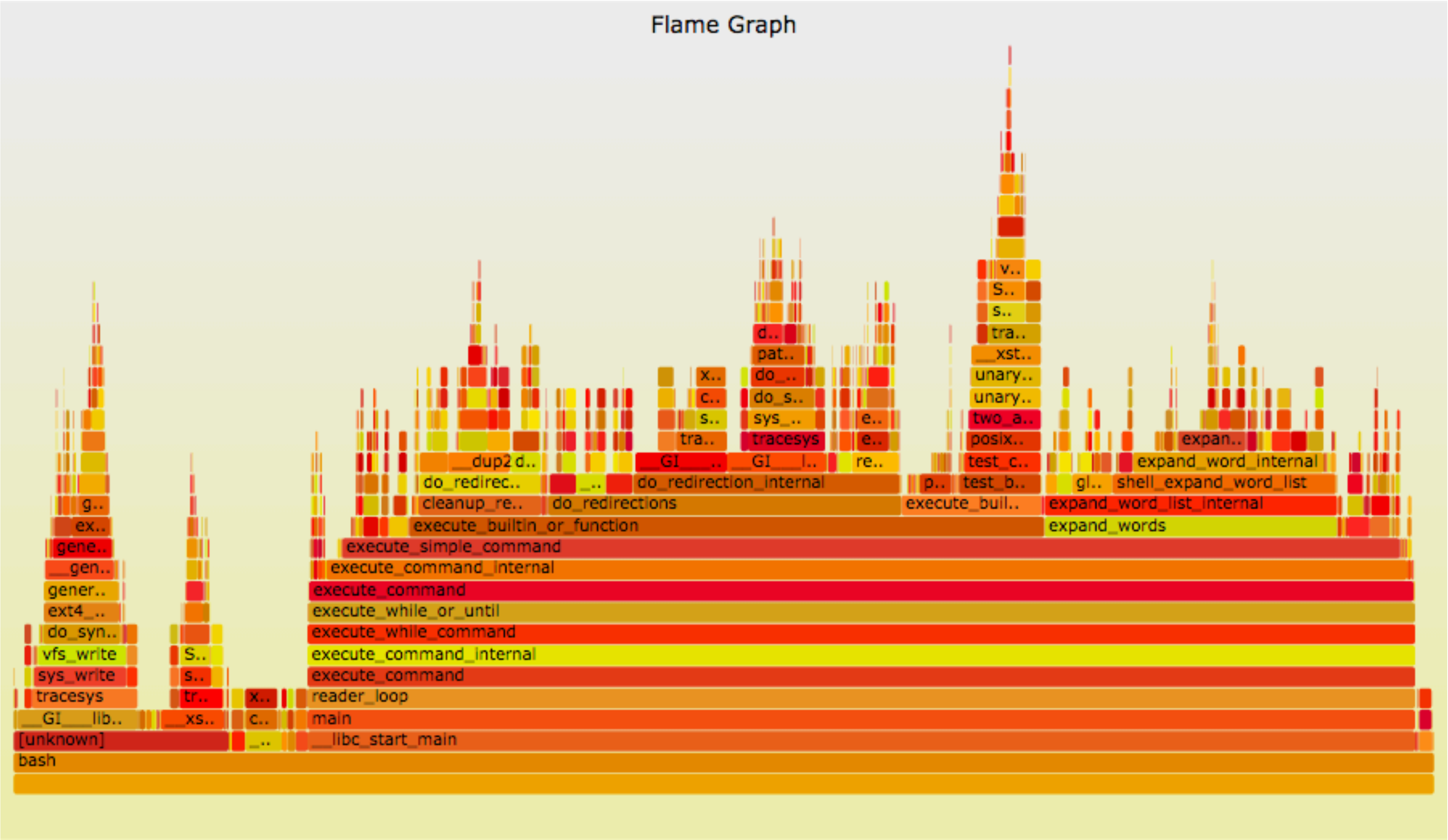
```
# perf record -F 99 -ag -- sleep 30
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 2.745 MB perf.data (~119930 samples) ]
# perf report -n -stdio
[…]
# Overhead       Samples  Command      Shared Object                         Symbol
# ........  ............  .......  .................  ..............................
#
    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 …]
```

call tree summary

Full perf report Output

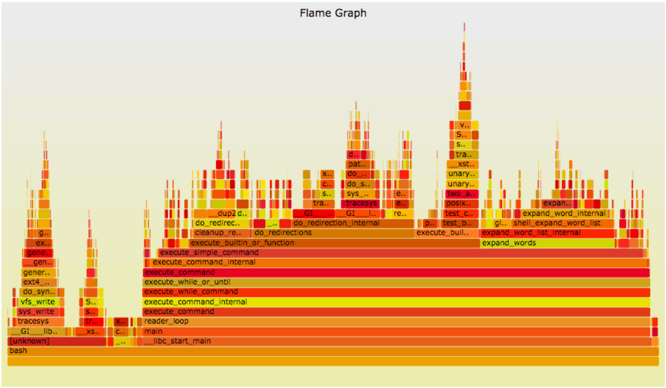


… as a Flame Graph

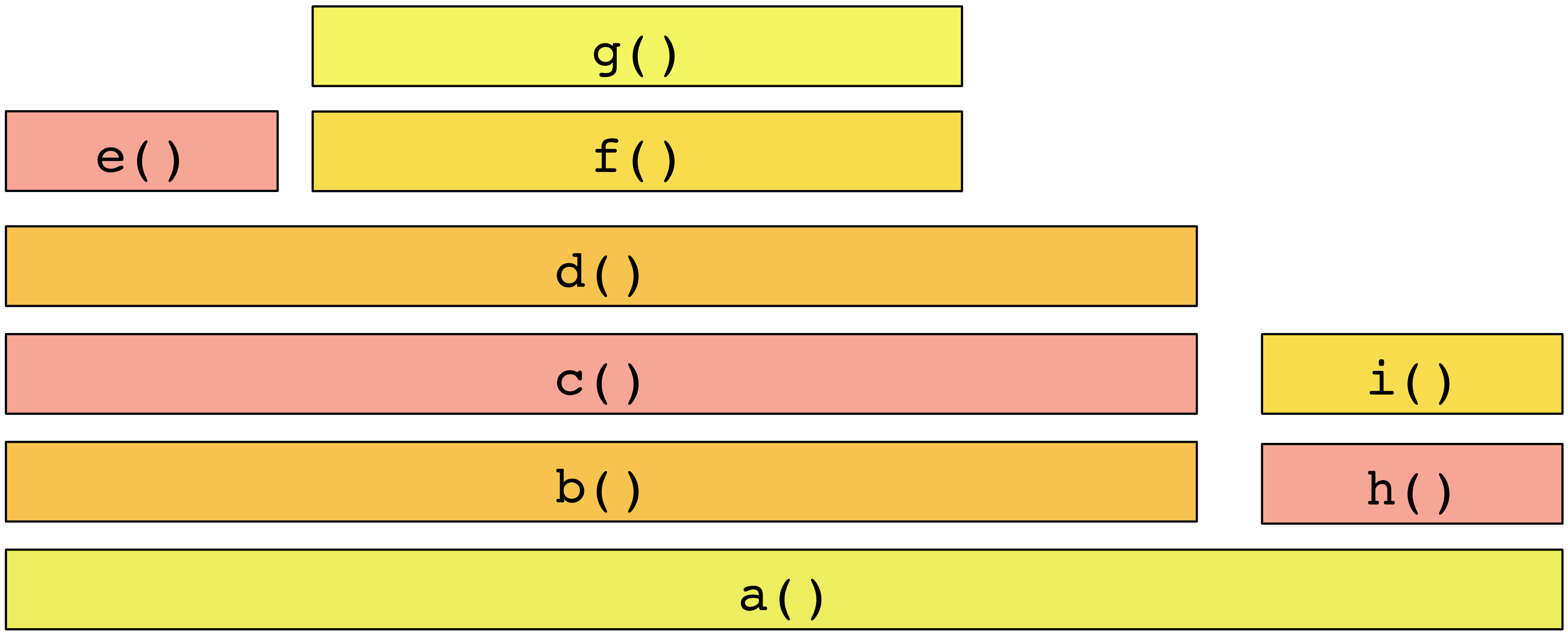


Flame Graph Summary

- Visualizes a collection of stack traces
  - **x-axis**: alphabetical stack sort, to maximize merging
  - **y-axis**: stack depth
  - **color**: random (default), or a dimension
- Currently made from Perl + SVG + JavaScript
  - **https://github.com/brendangregg/FlameGraph**
  - Takes input from many different profilers
  - Multiple d3 versions are being developed
- References:
  - http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
  - http://queue.acm.org/detail.cfm?id=2927301
  - "The Flame Graph" CACM, June 2016



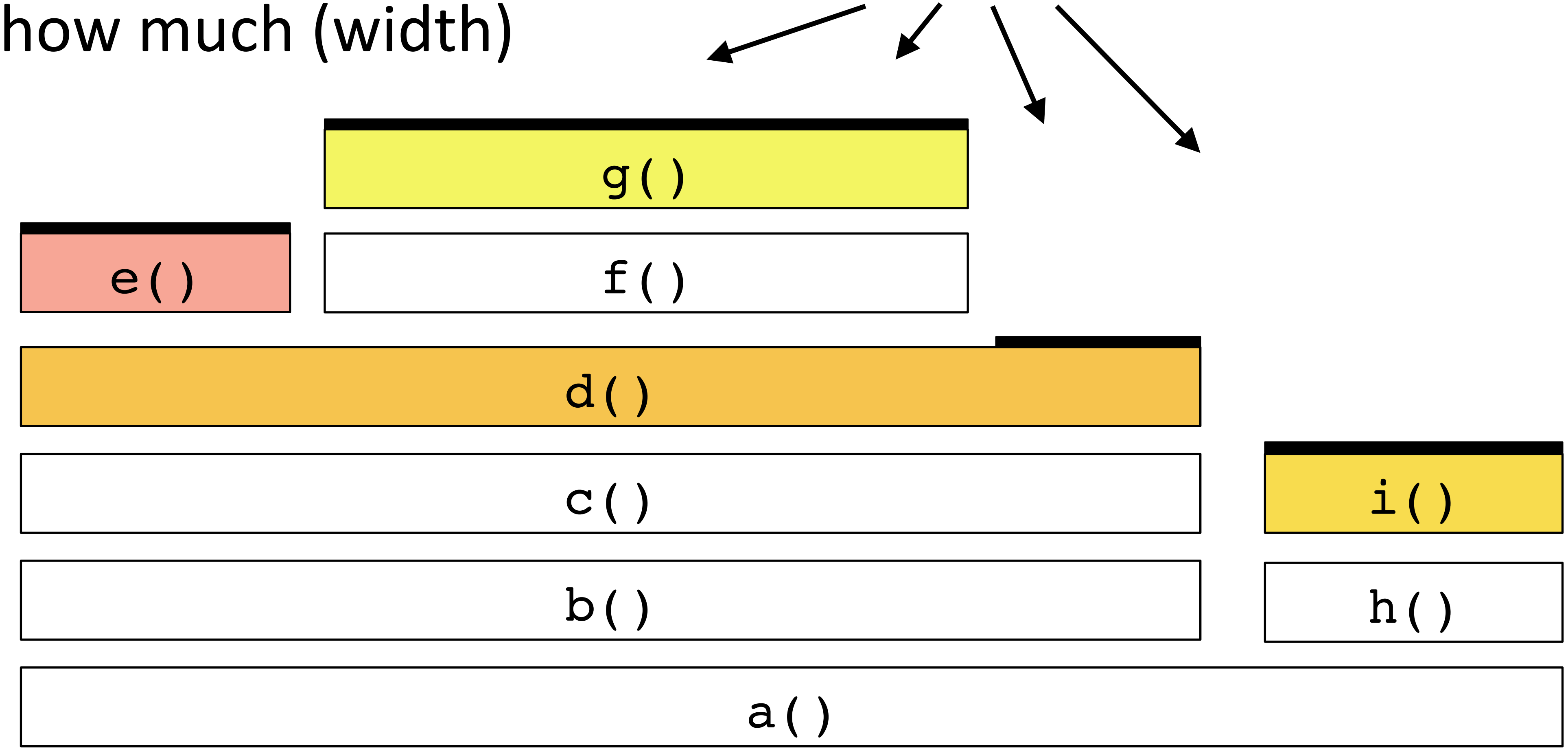
Flame Graph Interpretation



Flame Graph Interpretation (1/3)

Top edge shows who is running on-CPU,

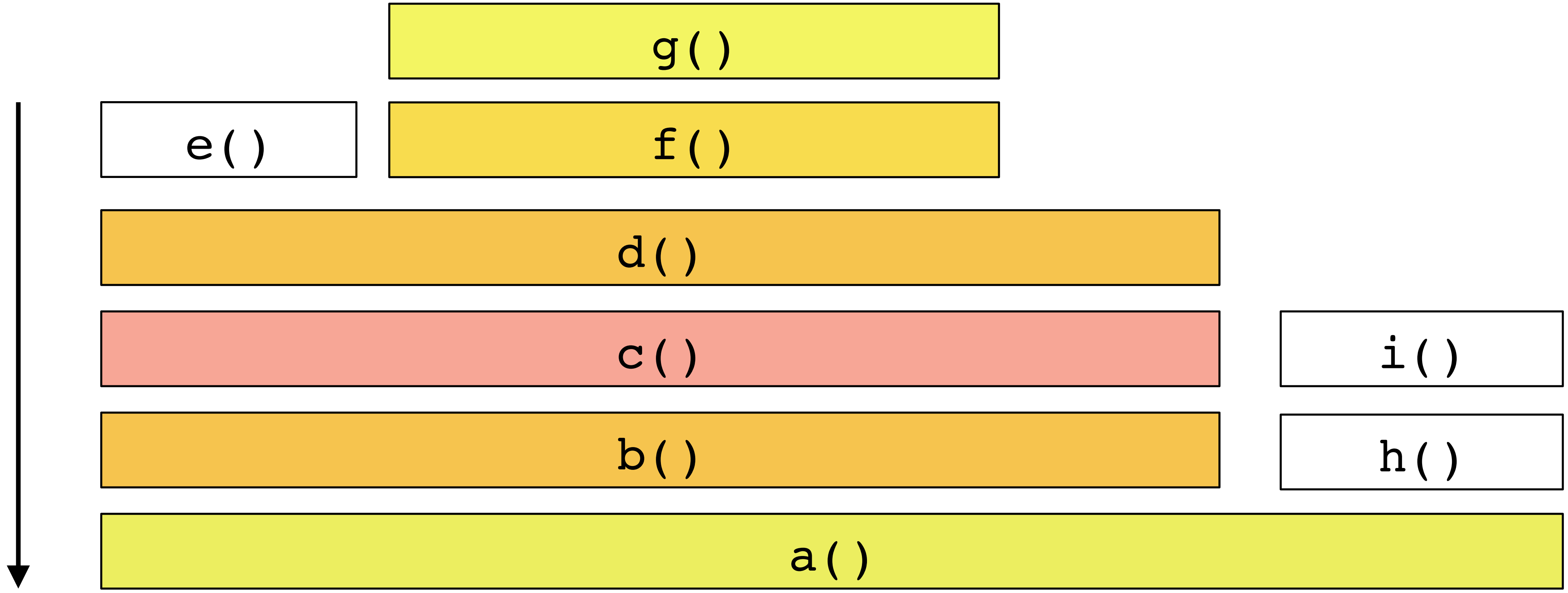
and how much (width)



Flame Graph Interpretation (2/3)

Top-down shows ancestry

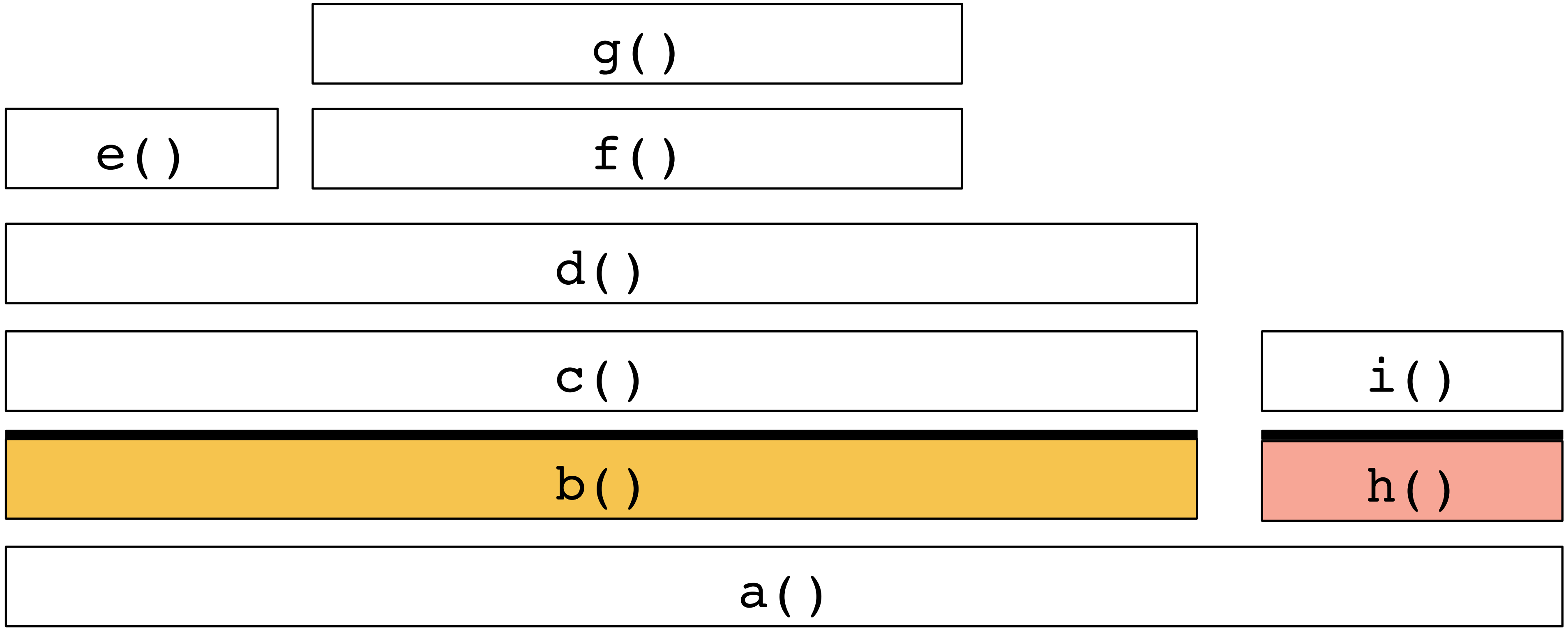
e.g., from g():



Flame Graph Interpretation (3/3)

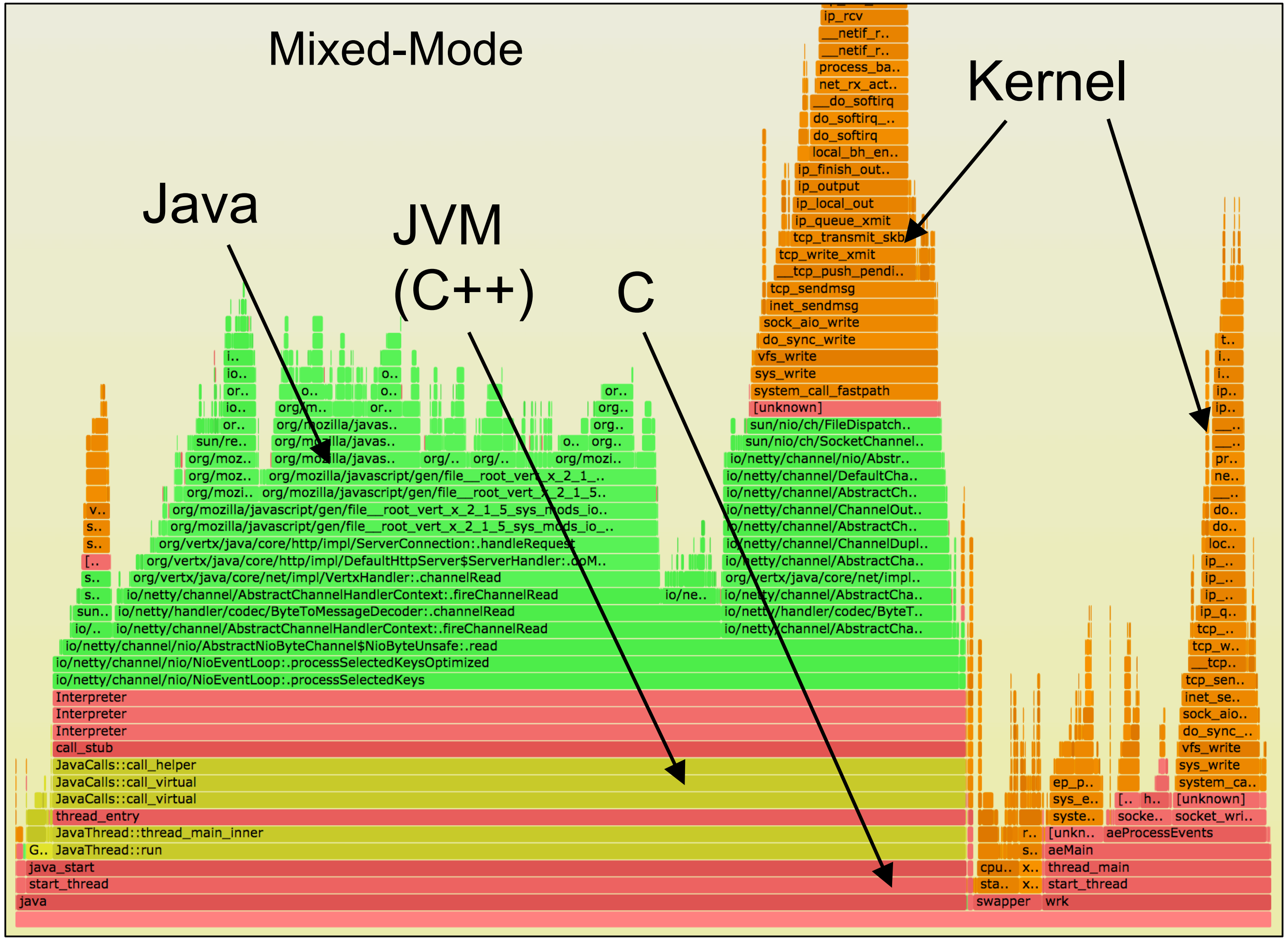
Widths are proportional to presence in samples

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



Mixed-Mode Flame Graphs

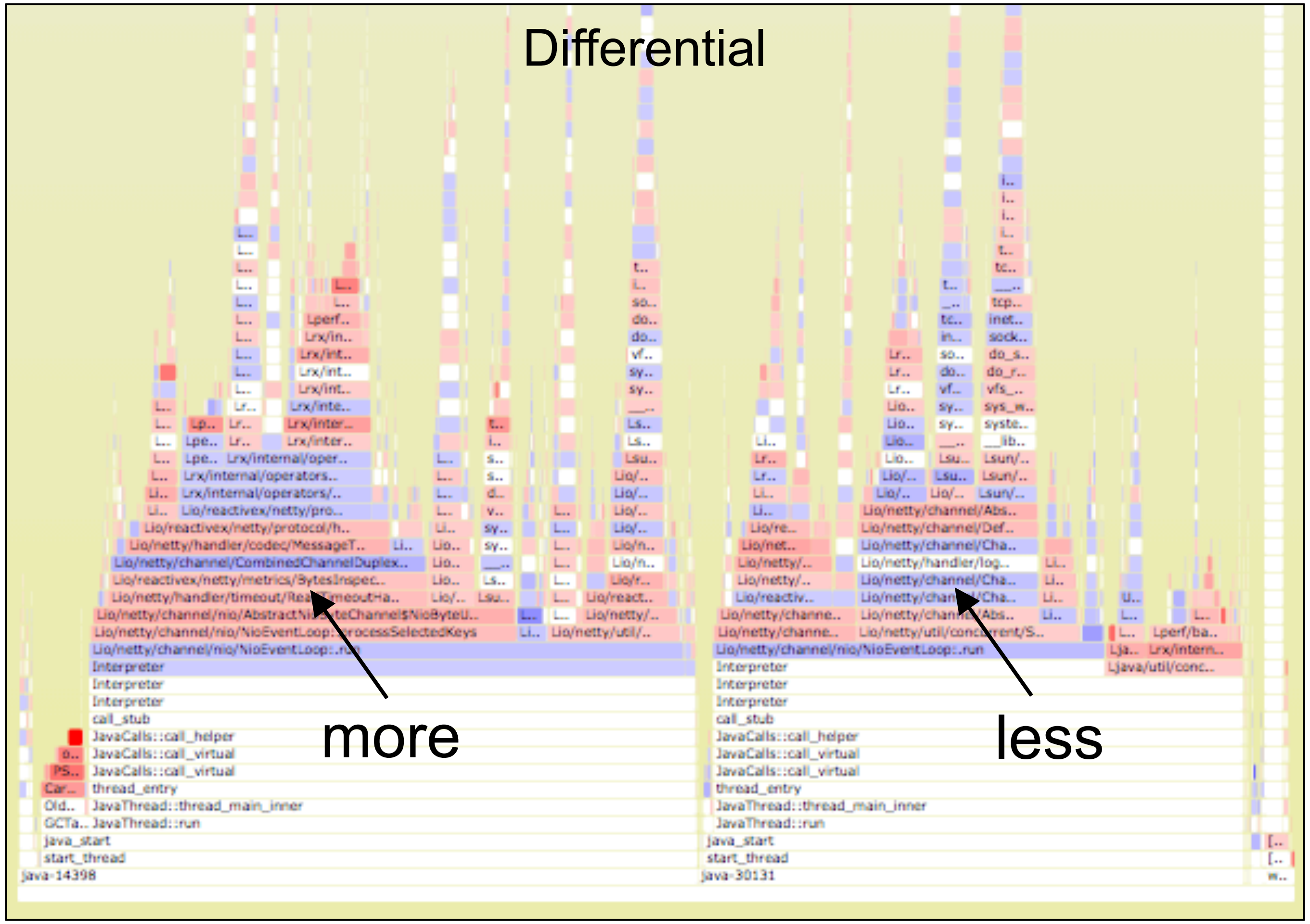
- Hues:
  - green == JIT (eg, Java)
  - aqua == inlined
    - if included
  - red == user-level*
  - orange == kernel
  - yellow == C++
- Intensity:
  - Randomized to differentiate frames
  - Or hashed on function name



* new palette uses red for kernel modules too

Differential Flame Graphs

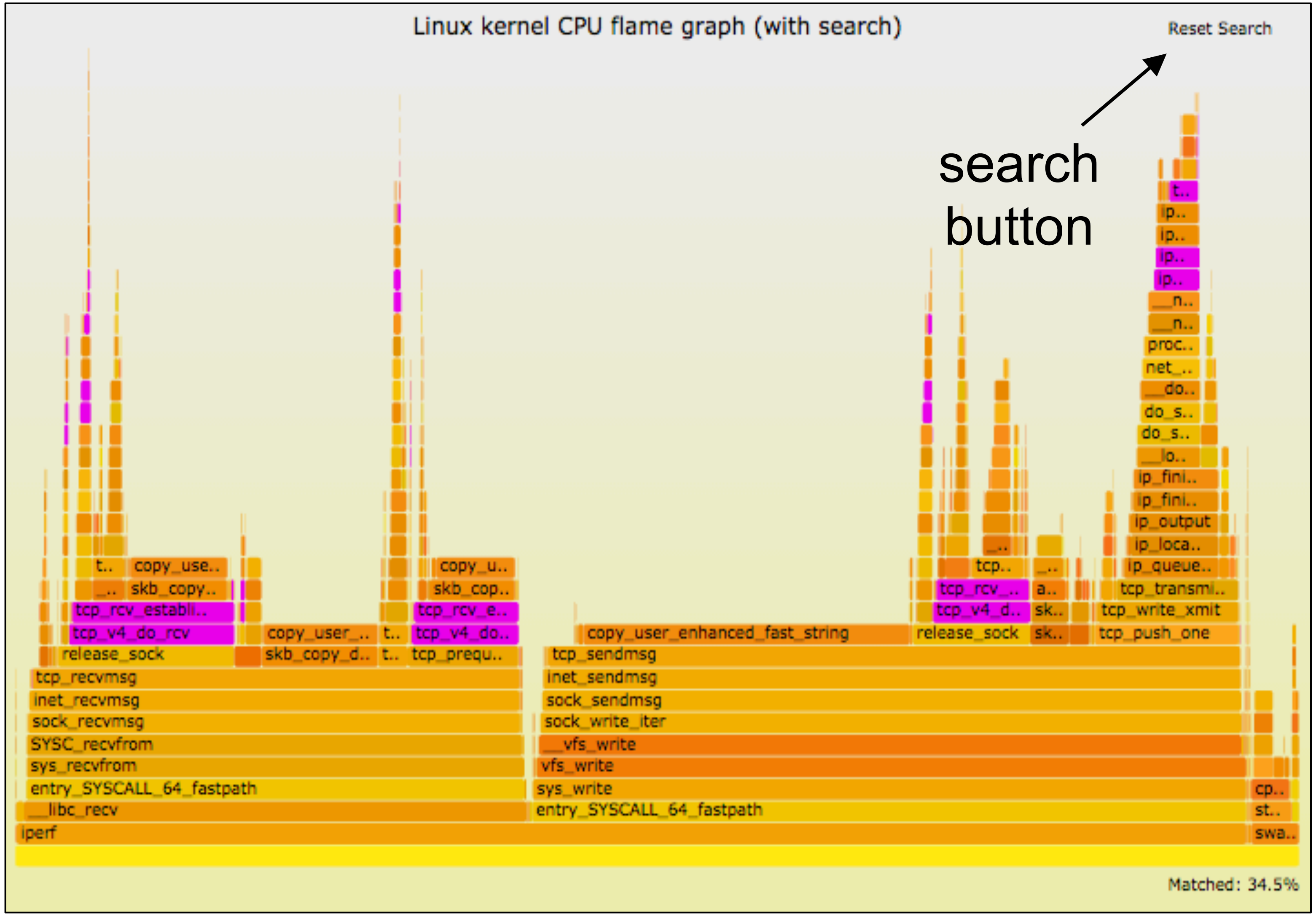
- Hues:
  - red == more samples
  - blue == less samples
- Intensity:
  - Degree of difference
- Compares two profiles
- Can show other metrics: e.g., CPI
- Other types exist
  - flamegraphdiff



Icicle Graph

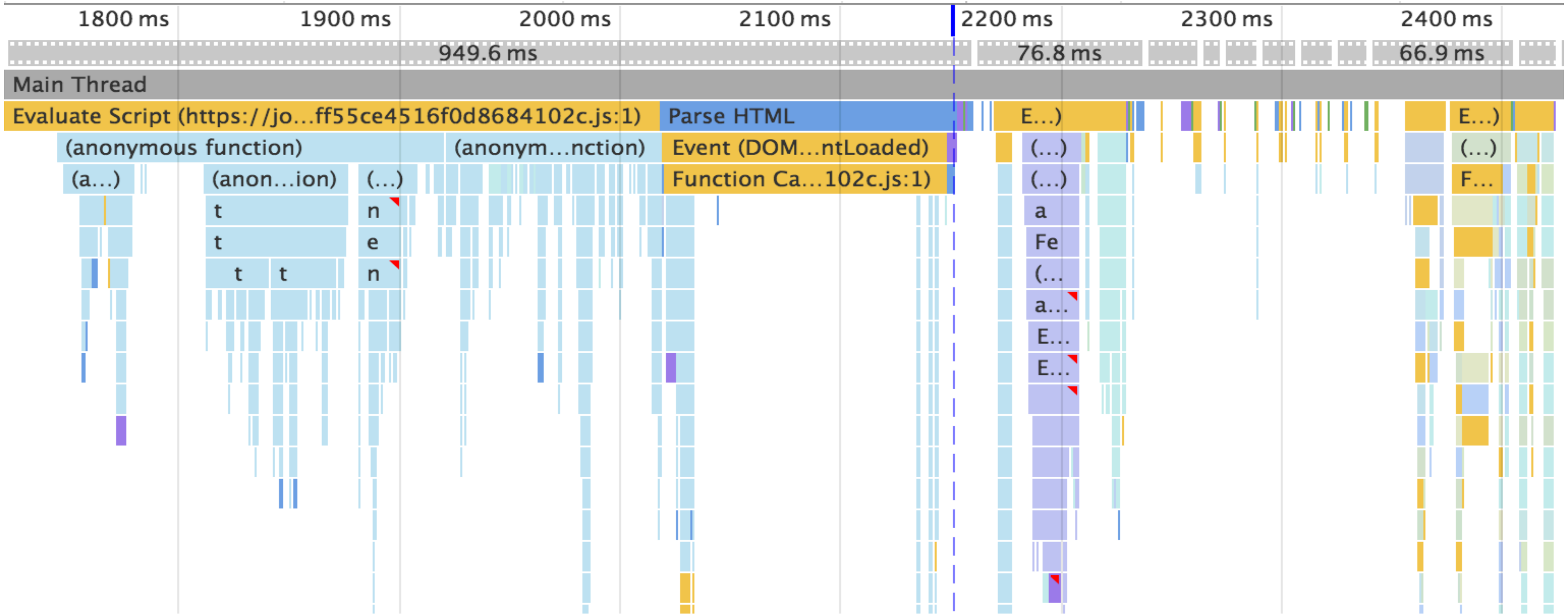
Flame Graph Search

- Color: magenta to show matched frames



Flame Charts

- Final note: these are useful, but are not flame *graphs*



from Chrome dev tools

- Flame **charts**: x-axis is time
- Flame **graphs**: x-axis is population (maximize merging)

Pitfalls and fixes

STACK TRACING

Broken Stack Traces are Common

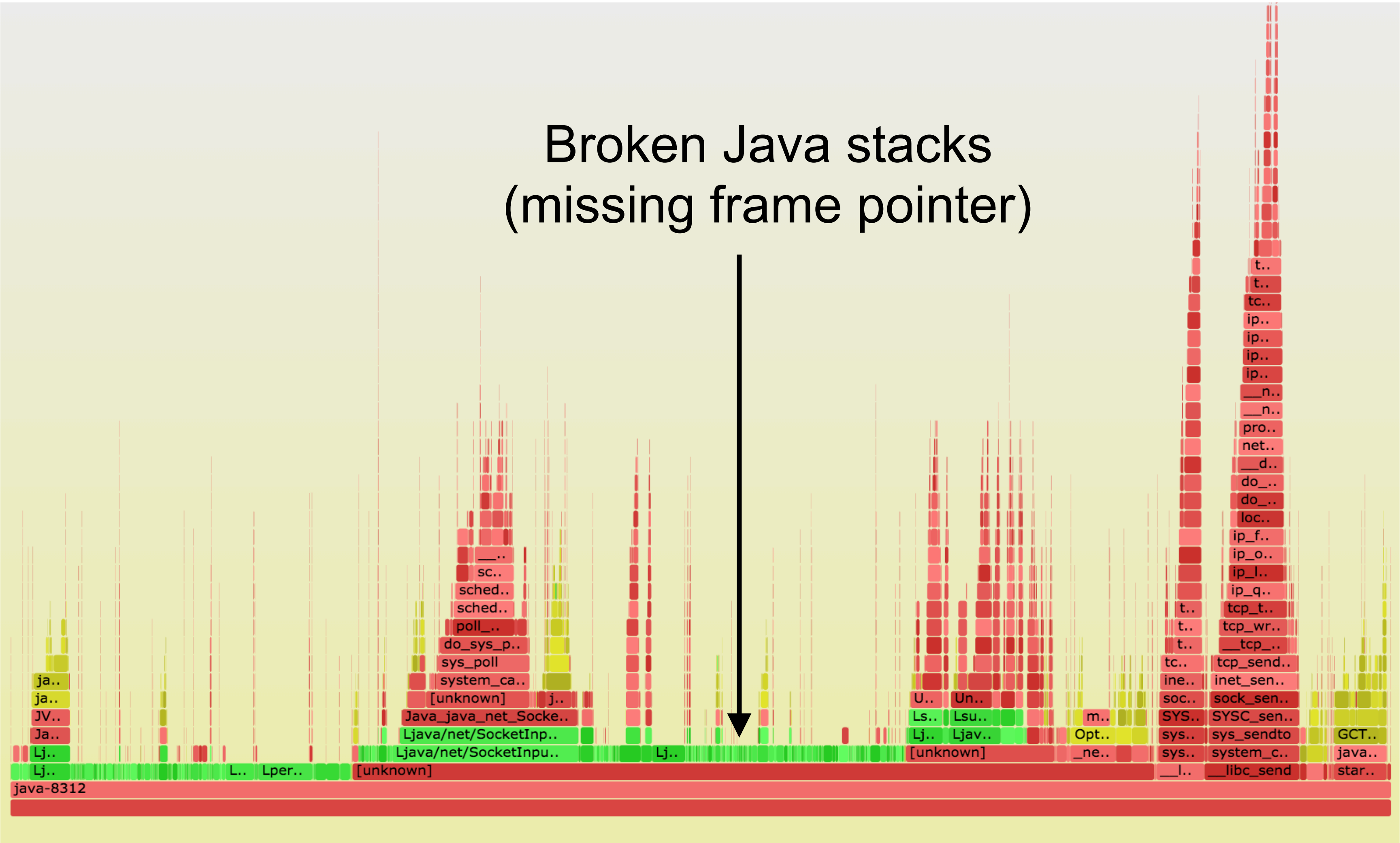
Because:

A. Profilers use frame pointer walking by default
B. Compilers reuse the frame pointer register as a general purpose register: a (usually very small) performance optimization.

```
# 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])
[…]
```

… 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
B. Debug info (DWARF) walking
   - Cons: costs disk space, and not supported by all profilers. Even possible with JIT?
C. JIT runtime walkers
   - Pros: include more internals, such as inlined frames
   - Cons: limited to application internals - no kernel
D. Last branch record

Fixing Java Stack Traces

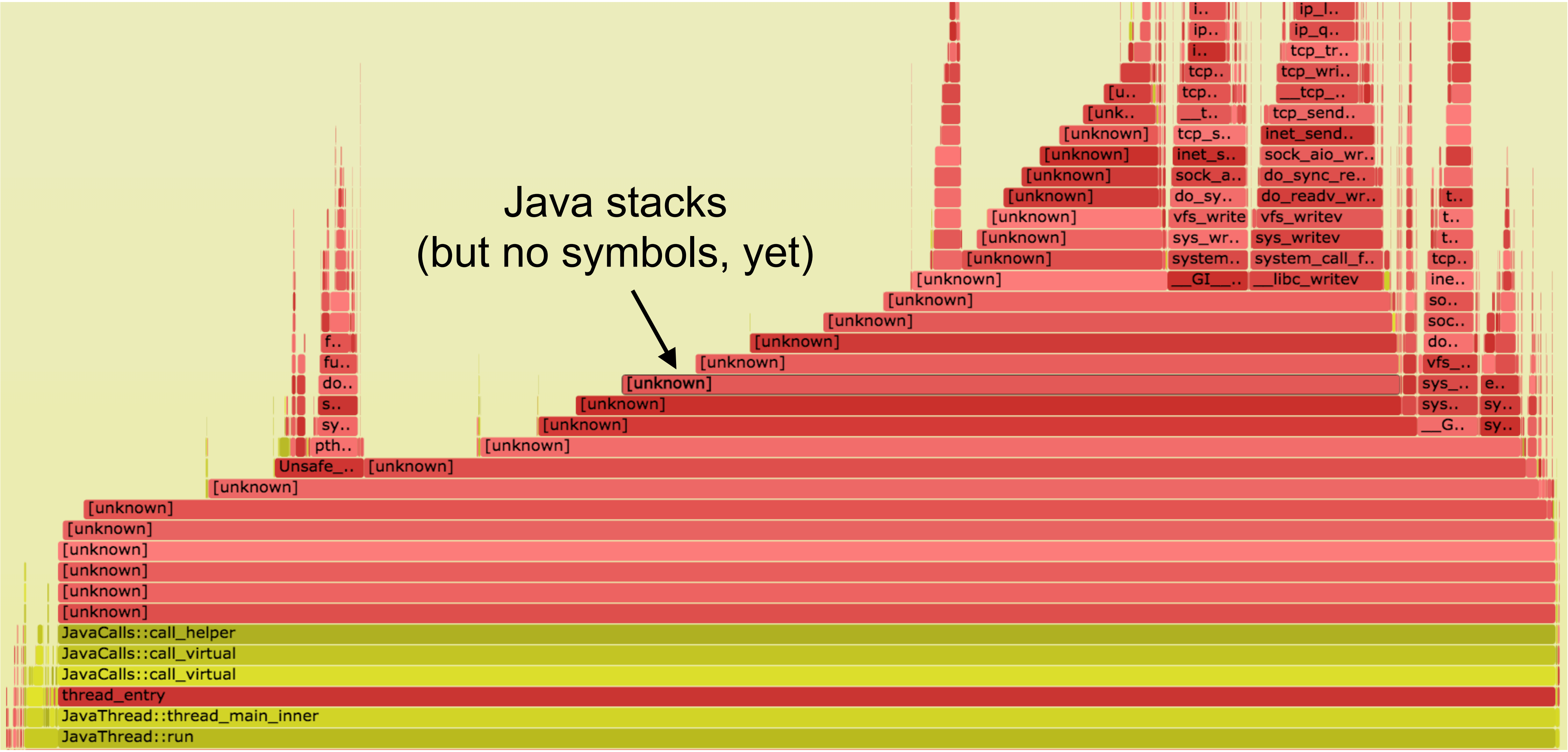
```
# perf script
[…]
java  4579 cpu-clock:
      7f417908c10b [unknown] (/tmp/…

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

```
# 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…
```

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

Fixed Stacks Flame Graph

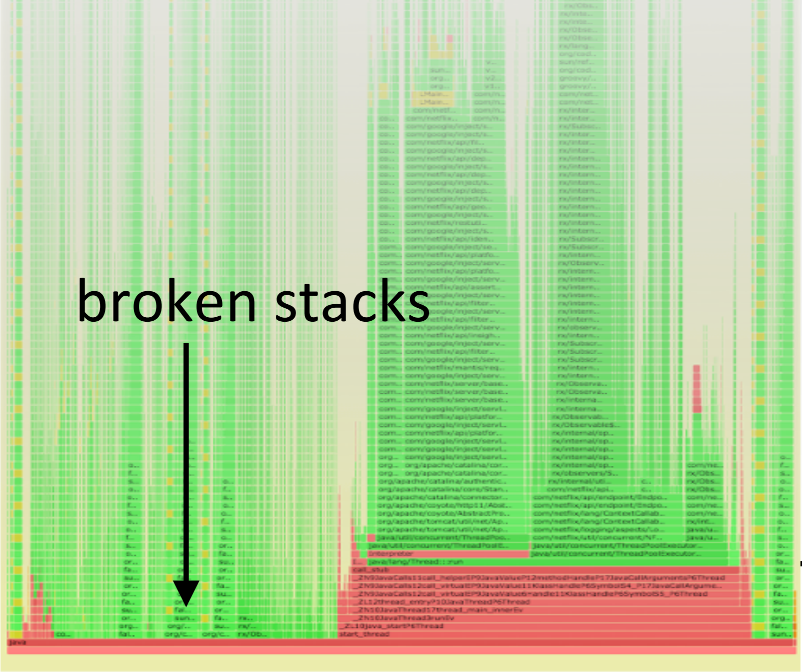


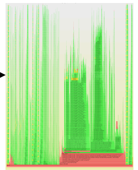
Inlining

- 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)

Stack Depth

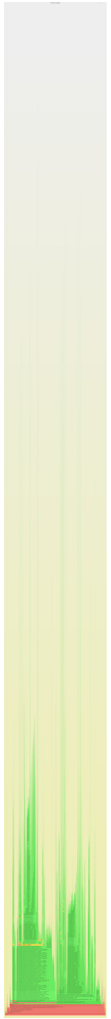
- perf had a 127 frame limit
- Now tunable in Linux 4.8
  - sysctl -w kernel.perf_event_max_stack=512
  - Thanks Arnaldo Carvalho de Melo!





perf_event_max_stack=1024

A Java microservice with a stack depth of > 900



Fixing

SYMBOLS

Fixing Native Symbols

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

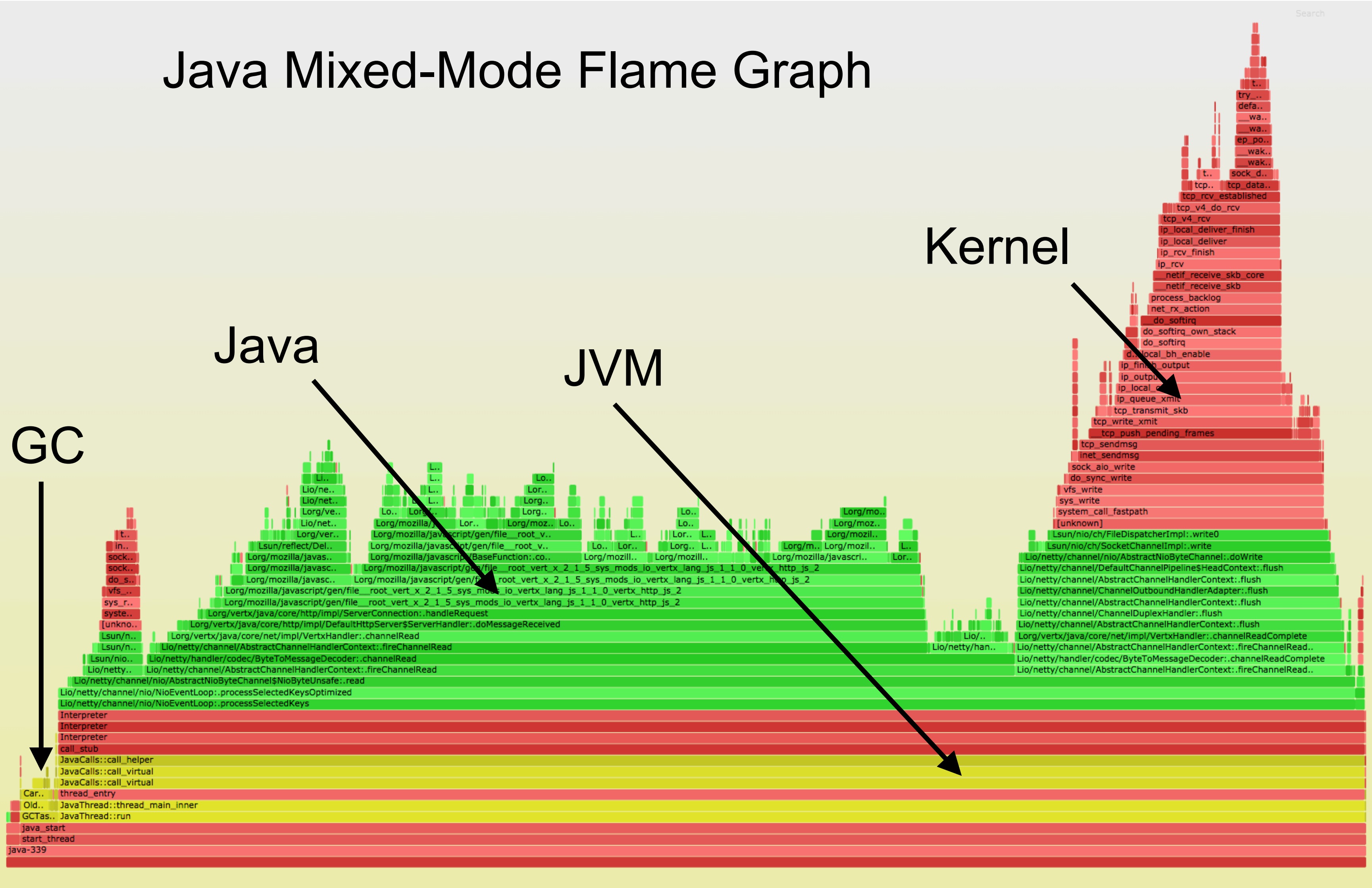
Fixing JIT Symbols (Java, Node.js, …)

- Just-in-time runtimes don't have a pre-compiled symbol table
- So Linux perf looks for an externally provided JIT symbol file: /tmp/perf-PID.map

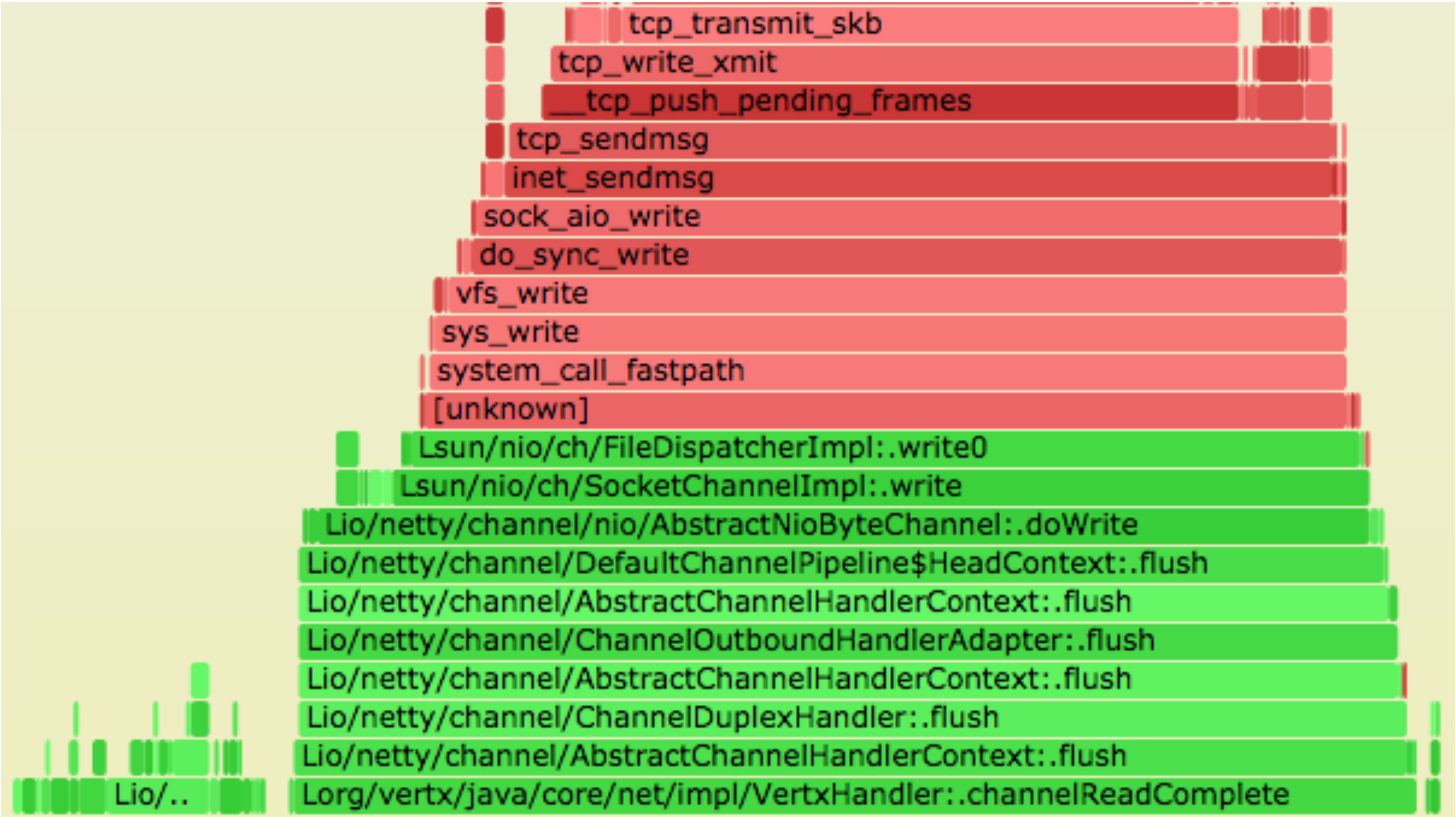
```
# perf script
Failed to open /tmp/perf-8131.map, continuing without symbols
[…]
java 8131 cpu-clock:
    7fff76f2dce1 [unknown] ([vdso])
    7fd3173f7a93 os::javaTimeMillis() (/usr/lib/jvm…
    7fd301861e46 [unknown] (/tmp/perf-8131.map)
[…]
```

- This can be created by runtimes; eg, Java's perf-map-agent

Fixed Stacks & Symbols



Stacks & Symbols (zoom)



Symbol Churn

- For JIT runtimes, symbols can change during a profile
- Symbols may be mistranslated by perf's map snapshot
- Solutions:
  A. Take a before & after snapshot, and compare
  B. perf's new support for timestamped symbol logs

Containers

- perf can't find any symbol sources
  - Unless you copy them into the host
- I'm testing Krister Johansen's fix, hopefully for Linux 4.13
  - lkml: "[PATCH tip/perf/core 0/7] namespace tracing improvements"

For Linux

INSTRUCTIONS

Linux CPU Flame Graphs

Linux 2.6+, via perf.data and perf script:

```
git clone --depth 1 https://github.com/brendangregg/FlameGraph
cd FlameGraph
perf record -F 99 -a –g -- sleep 30
perf script | ./stackcollapse-perf.pl |./flamegraph.pl > perf.svg
```

Linux 4.5+ can use folded output

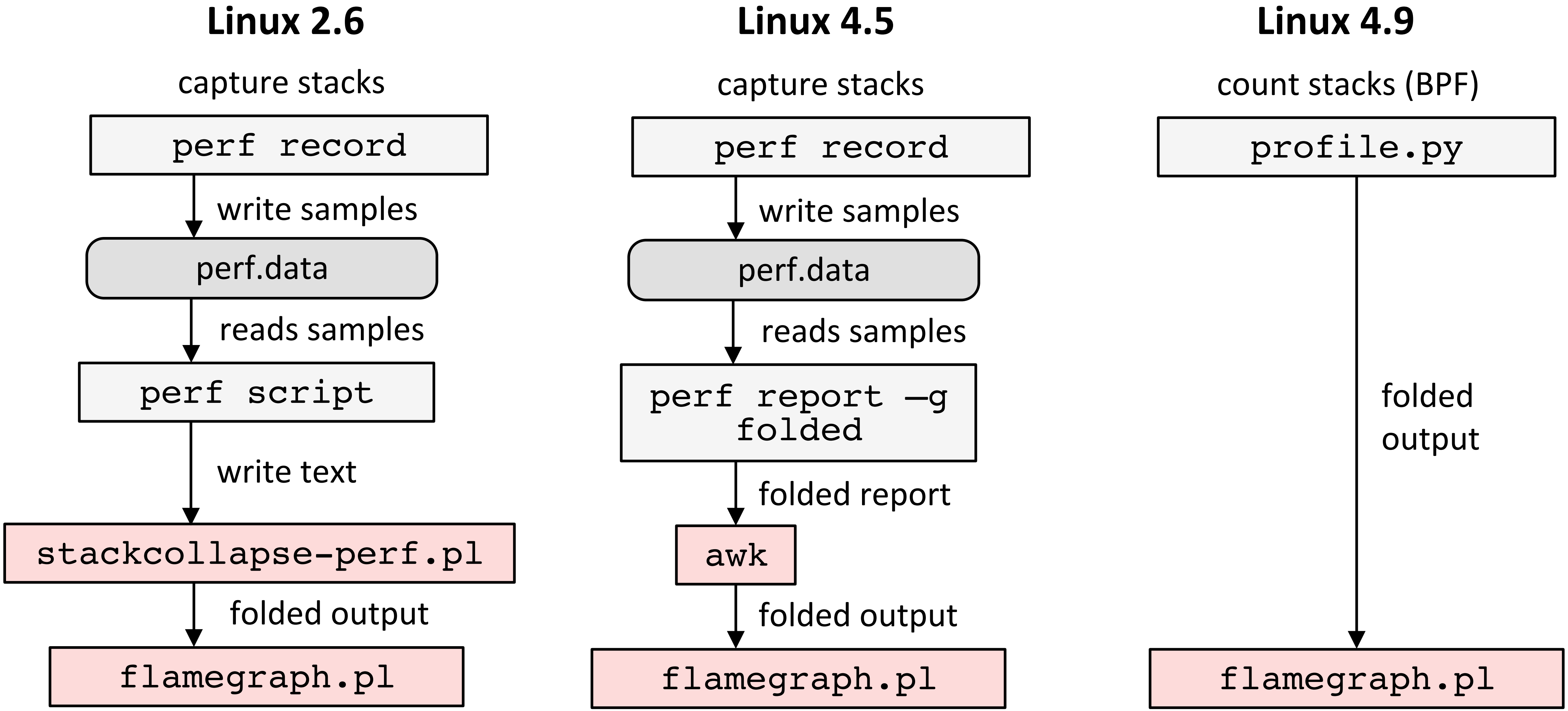
- Skips the CPU-costly stackcollapse-perf.pl step; see: http://www.brendangregg.com/blog/2016-04-30/linux-perf-folded.html

Linux 4.9+, via BPF:

```
git clone --depth 1 https://github.com/brendangregg/FlameGraph
git clone --depth 1 https://github.com/iovisor/bcc
./bcc/tools/profile.py -dF 99 30 | ./FlameGraph/flamegraph.pl > perf.svg
```

- Most efficient: no perf.data file, summarizes in-kernel

Linux Profiling Optimizations



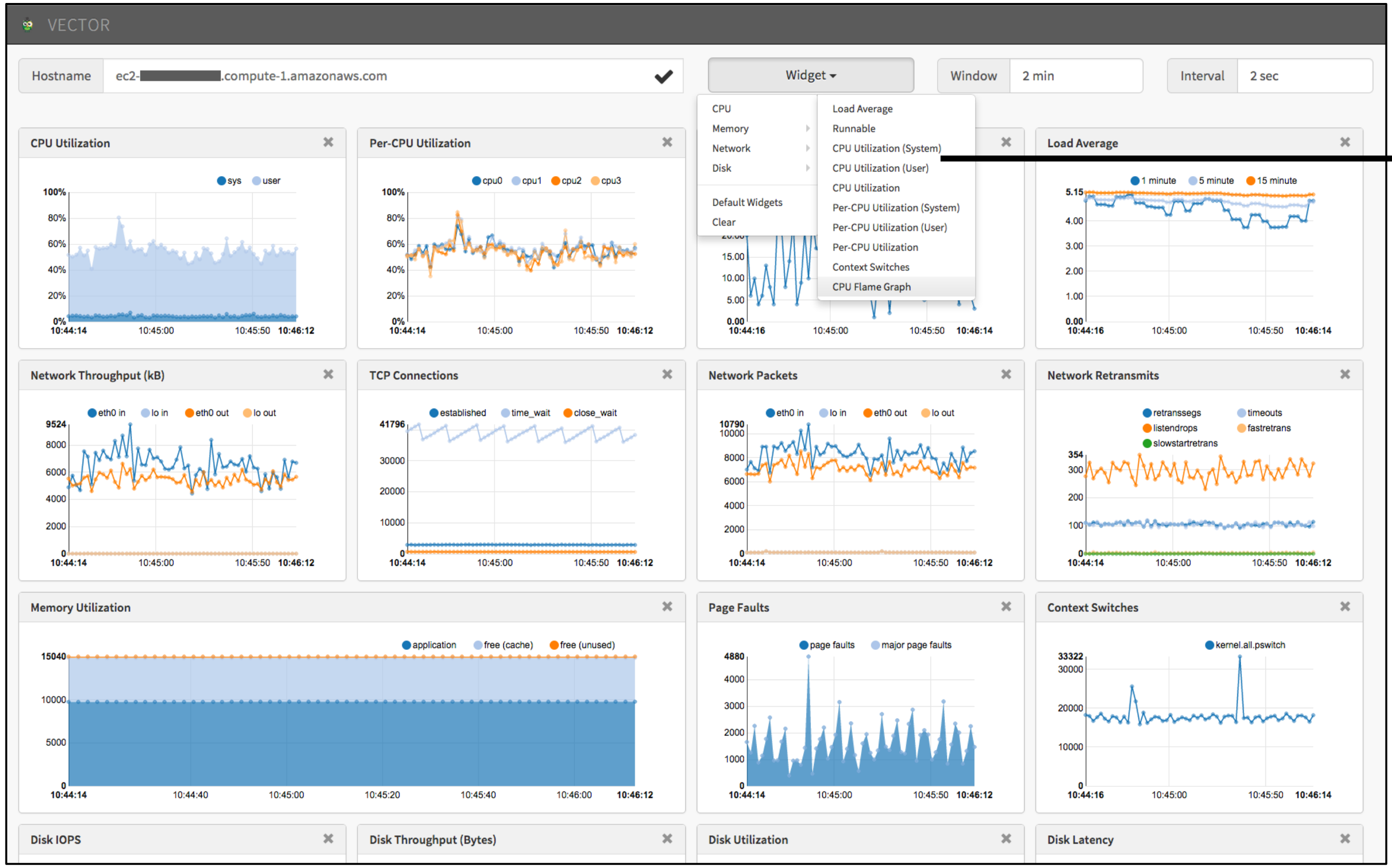
Language/Runtime Instructions

- Each may have special stack/symbol instructions
  - Java, Node.js, Python, Ruby, C++, Go, …
- I'm documenting some on:
  - http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
  - Also try an Internet search

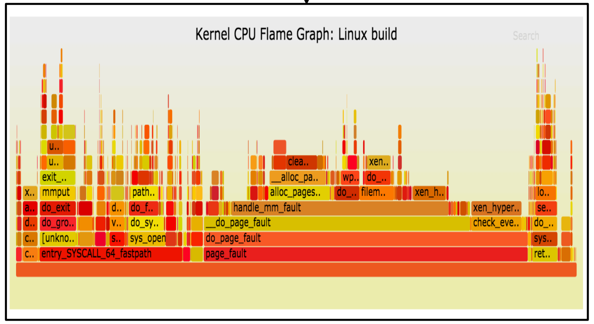
GUI Automation

Eg, Netflix Vector (self-service UI):

Vector



Flame Graphs

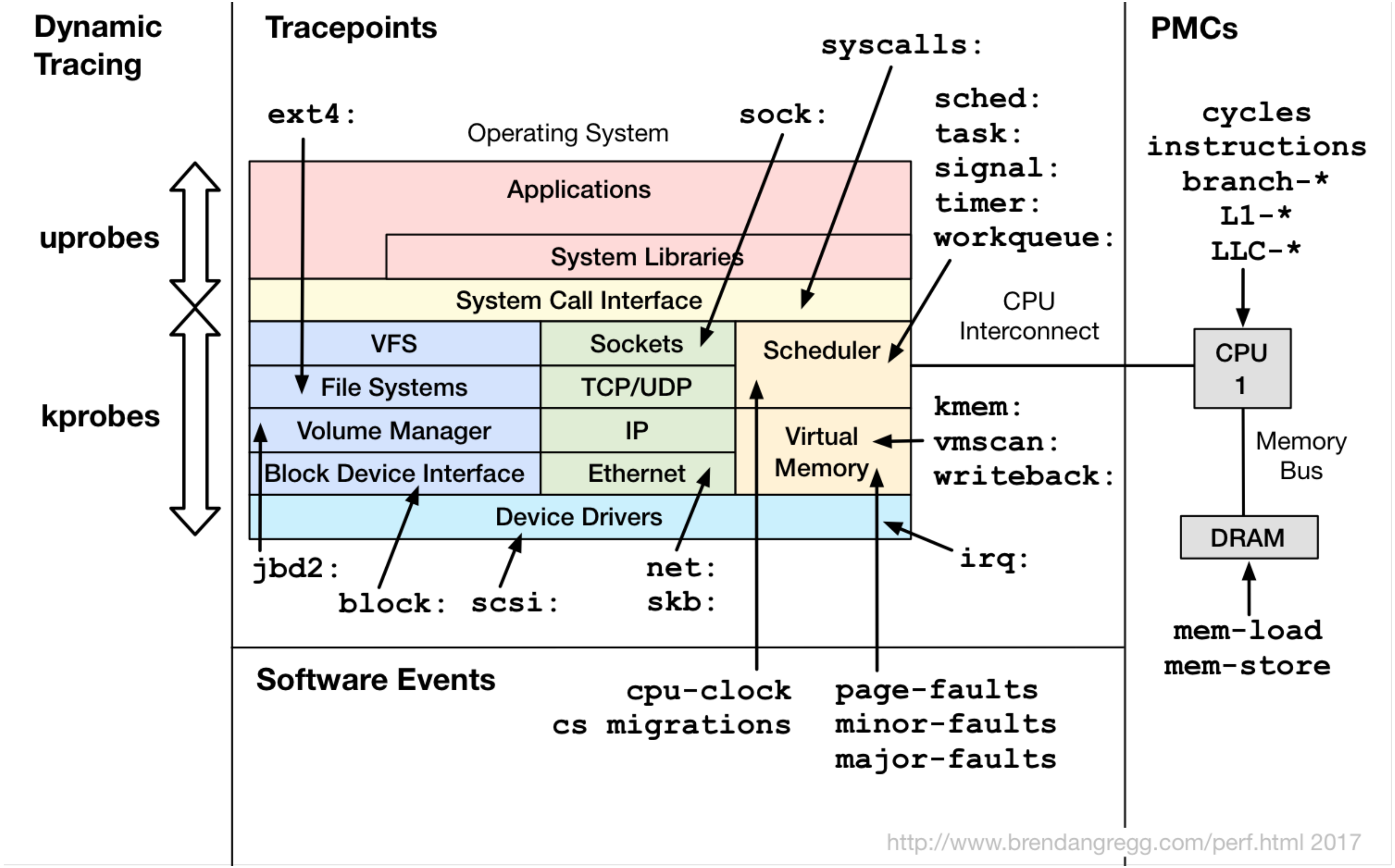


Should be open sourced; you may also build/buy your own

Future Work

ADVANCED FLAME GRAPHS

Flame graphs can be generated for stack traces from any Linux event source



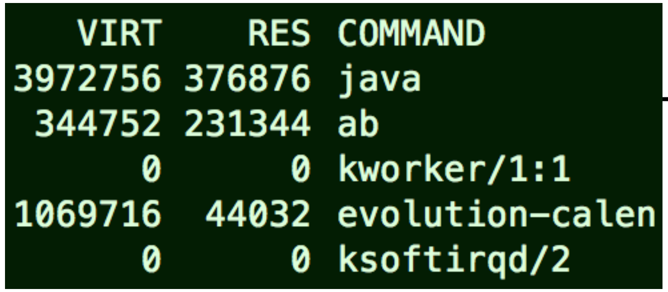
Page Faults

- Show what triggered main memory (resident) to grow:

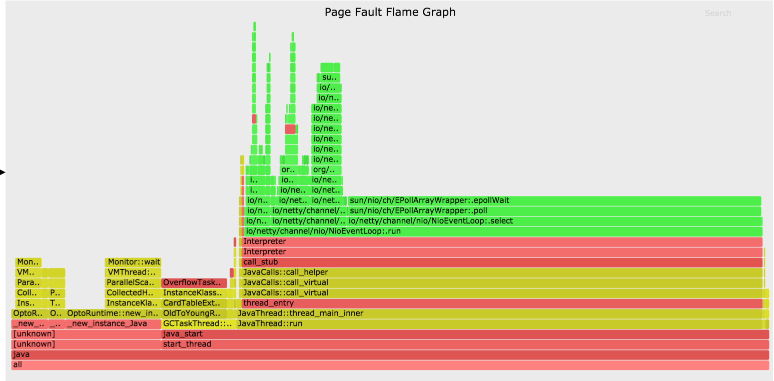
```
# perf record -e page-faults -p PID -g -- sleep 120
```

- "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

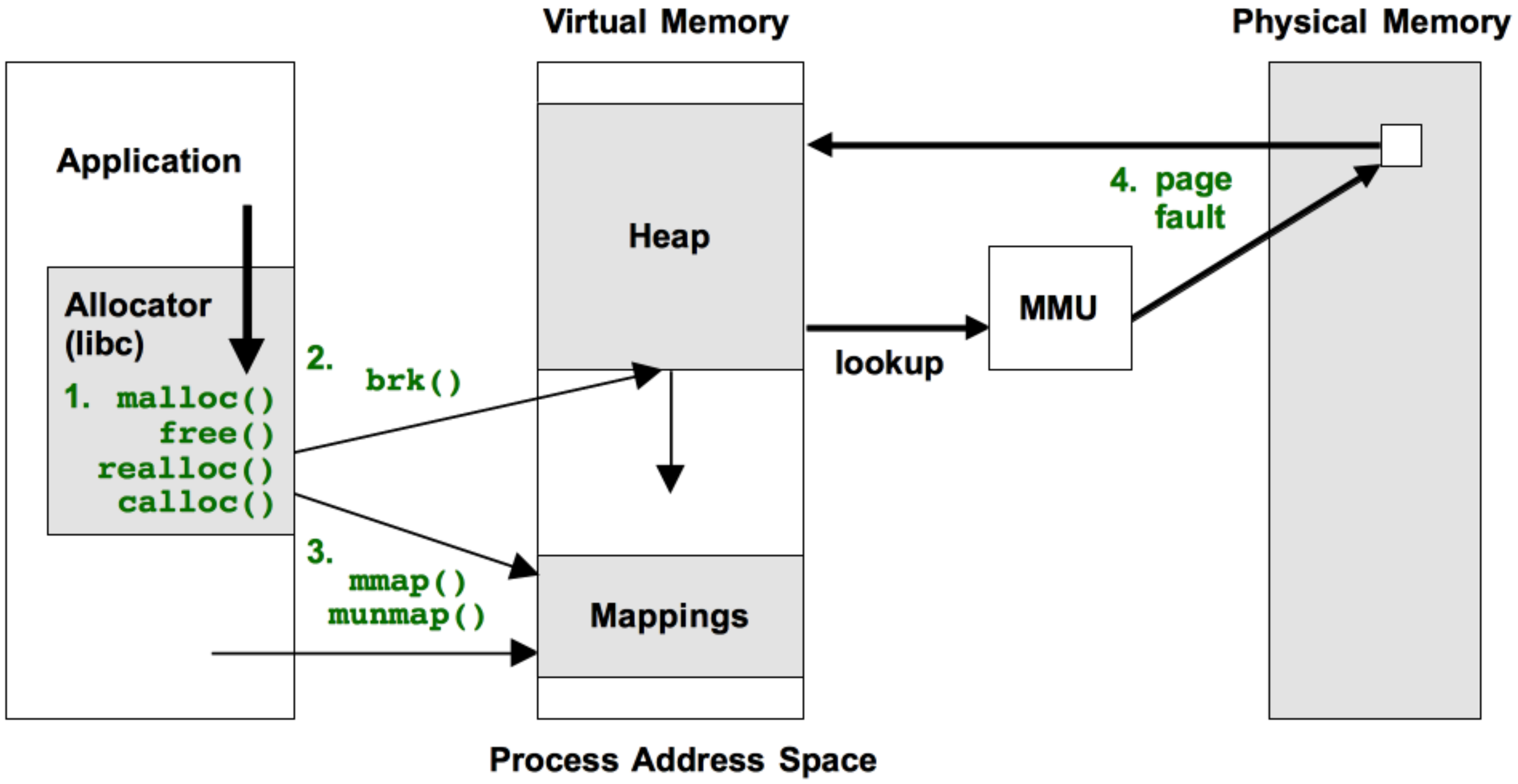
RES column in top(1)



grows because



Other Memory Sources



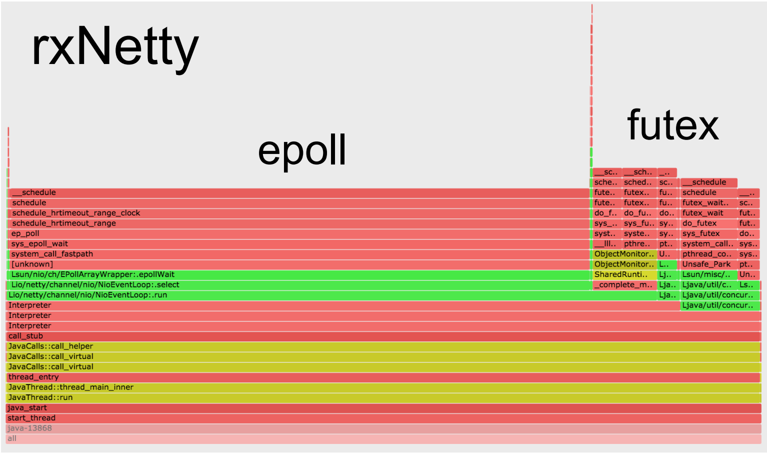
http://www.brendangregg.com/FlameGraphs/memoryflamegraphs.html

Context Switches

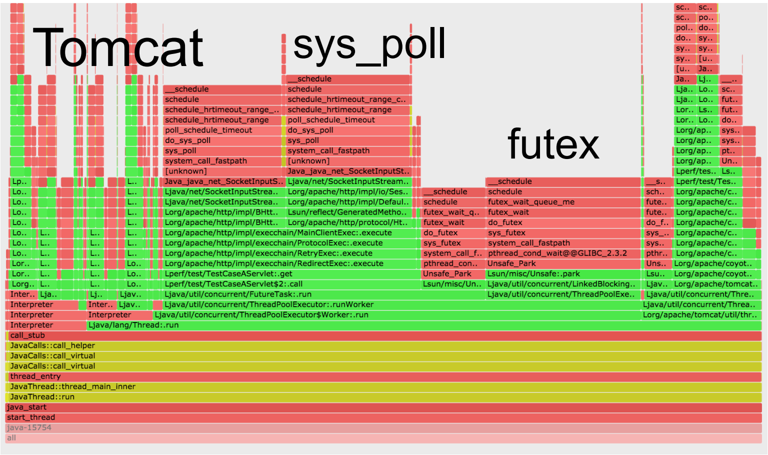
- Show why Java blocked and stopped running on-CPU:

```
# 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 could filter these, but you should have solved them beforehand (CPU load).
- e.g., was used to understand framework differences:



vs

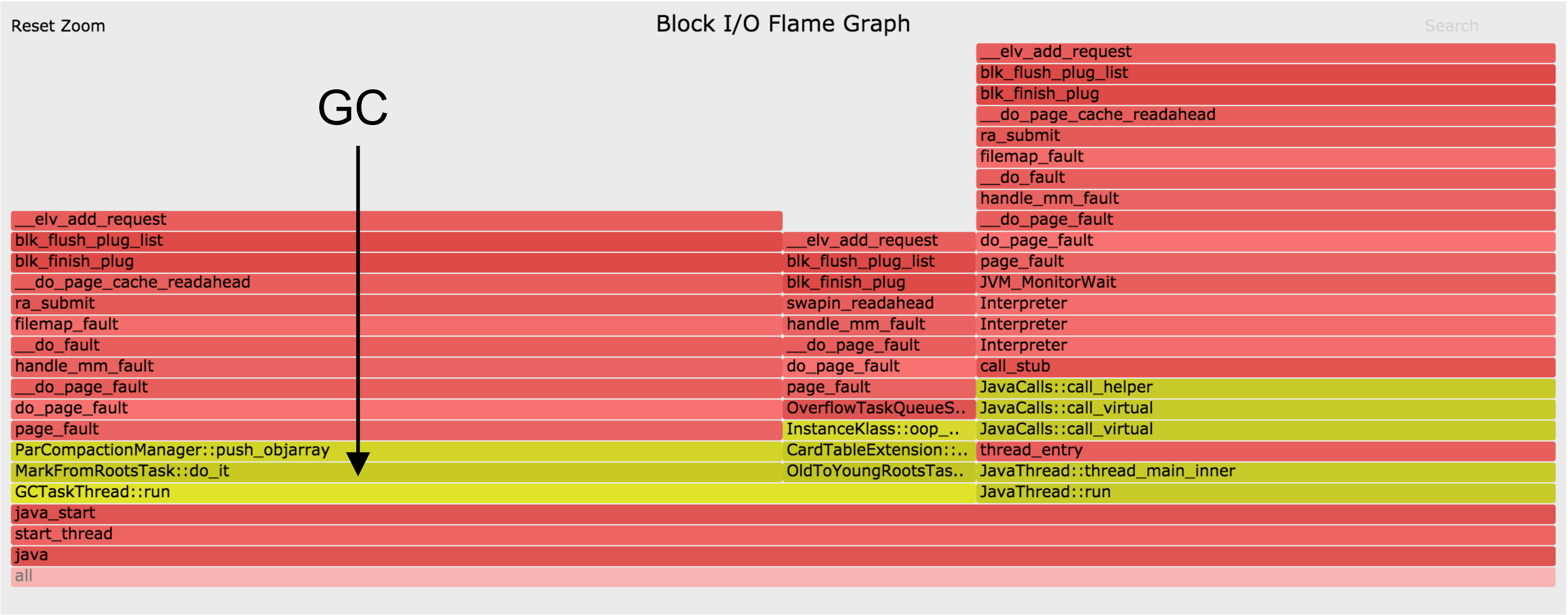


Disk I/O Requests

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

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

- e.g.: page faults in GC? This JVM has swapped out!:

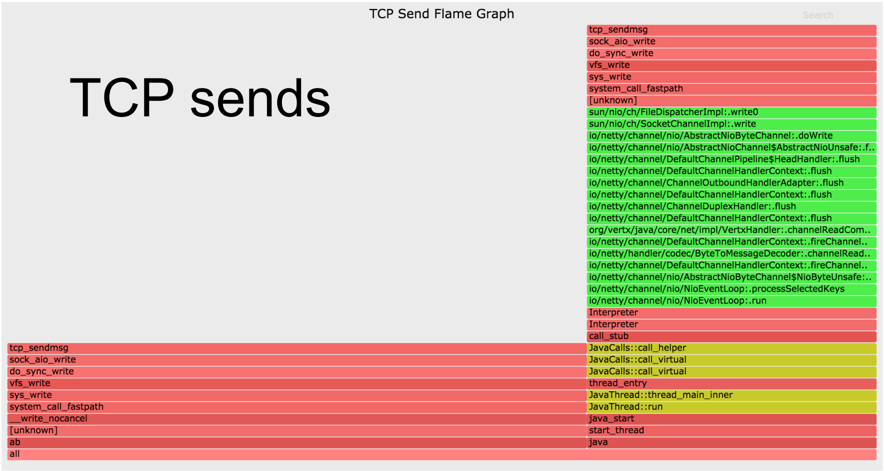


TCP Events

- TCP transmit, using dynamic tracing:

```
# perf probe tcp_sendmsg
# perf record -e probe:tcp_sendmsg -a -g -- sleep 1; jmaps
# perf script -f comm,pid,tid,cpu,time,event,ip,sym,dso,trace > out.stacks
# perf probe --del tcp_sendmsg
```

- Note: can be high overhead for high packet rates
  - For the current perf trace, dump, post-process cycle
- Can also trace TCP connect & accept
  - Lower frequency, therefore lower overhead
- TCP receive is async
  - Could trace via socket read

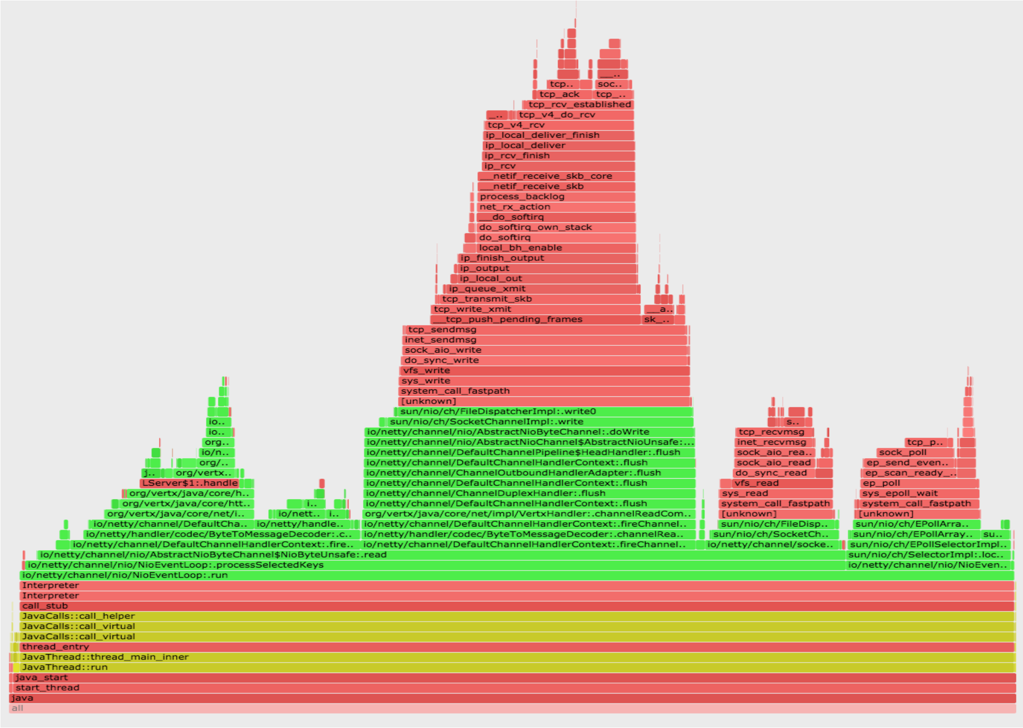


CPU Cache Misses

- 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
```

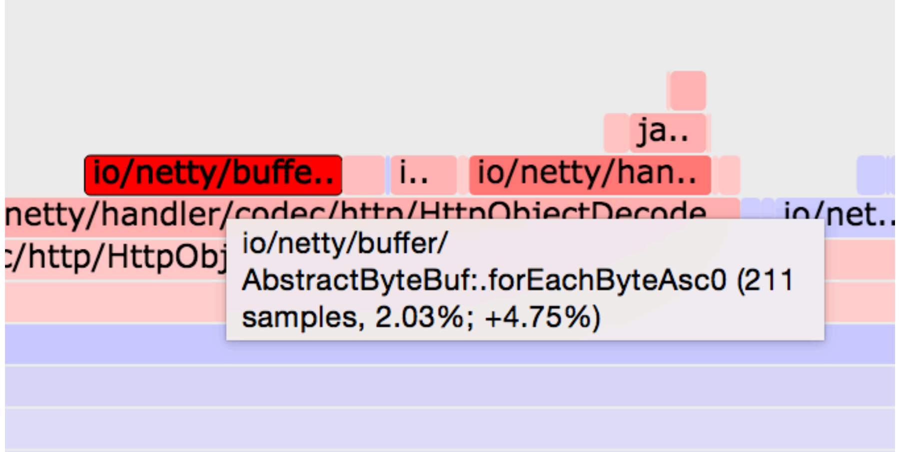
- -c is the count (samples once per count)
- Use other CPU counters to sample hits, misses, stalls

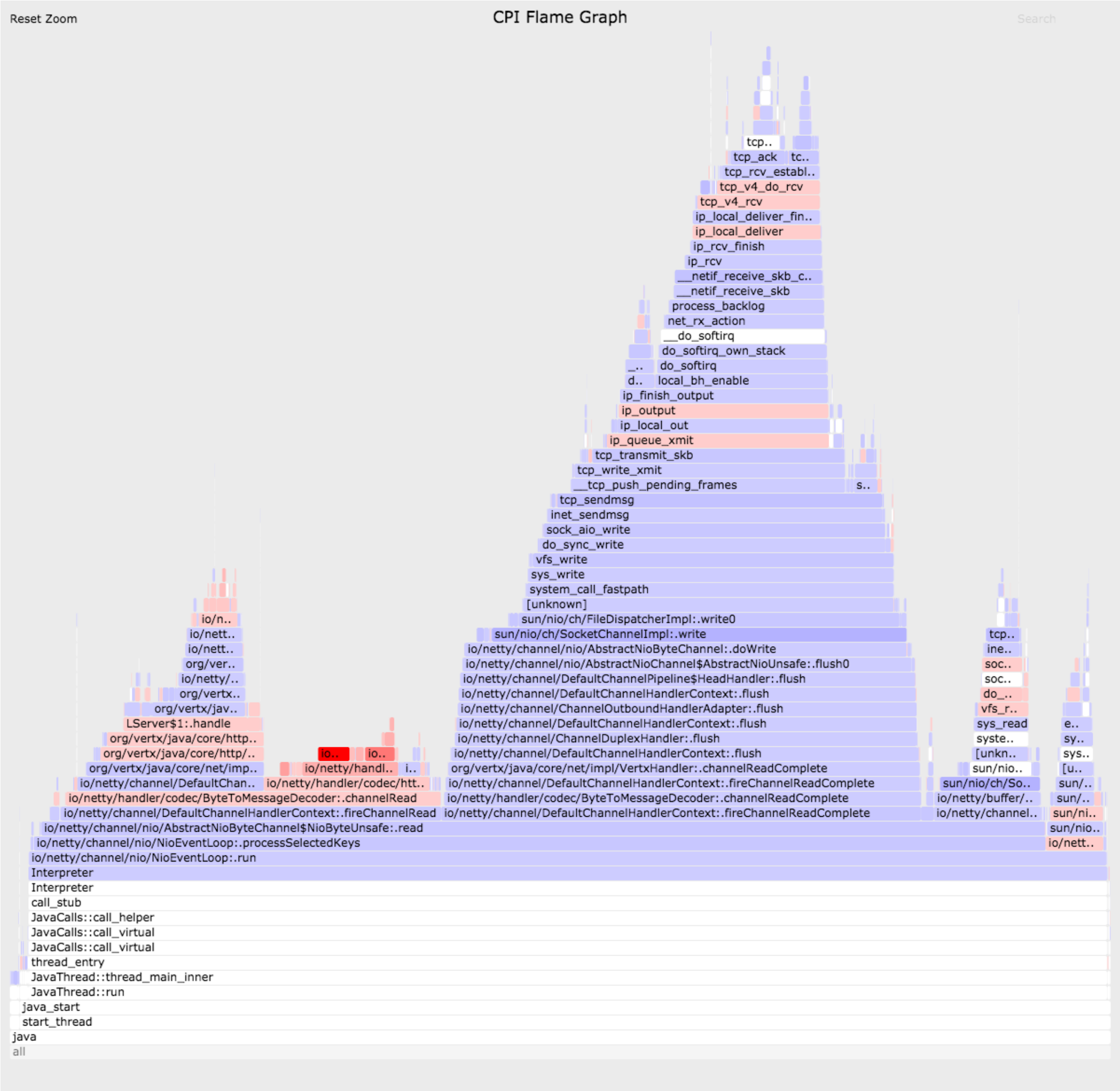


CPI Flame Graph

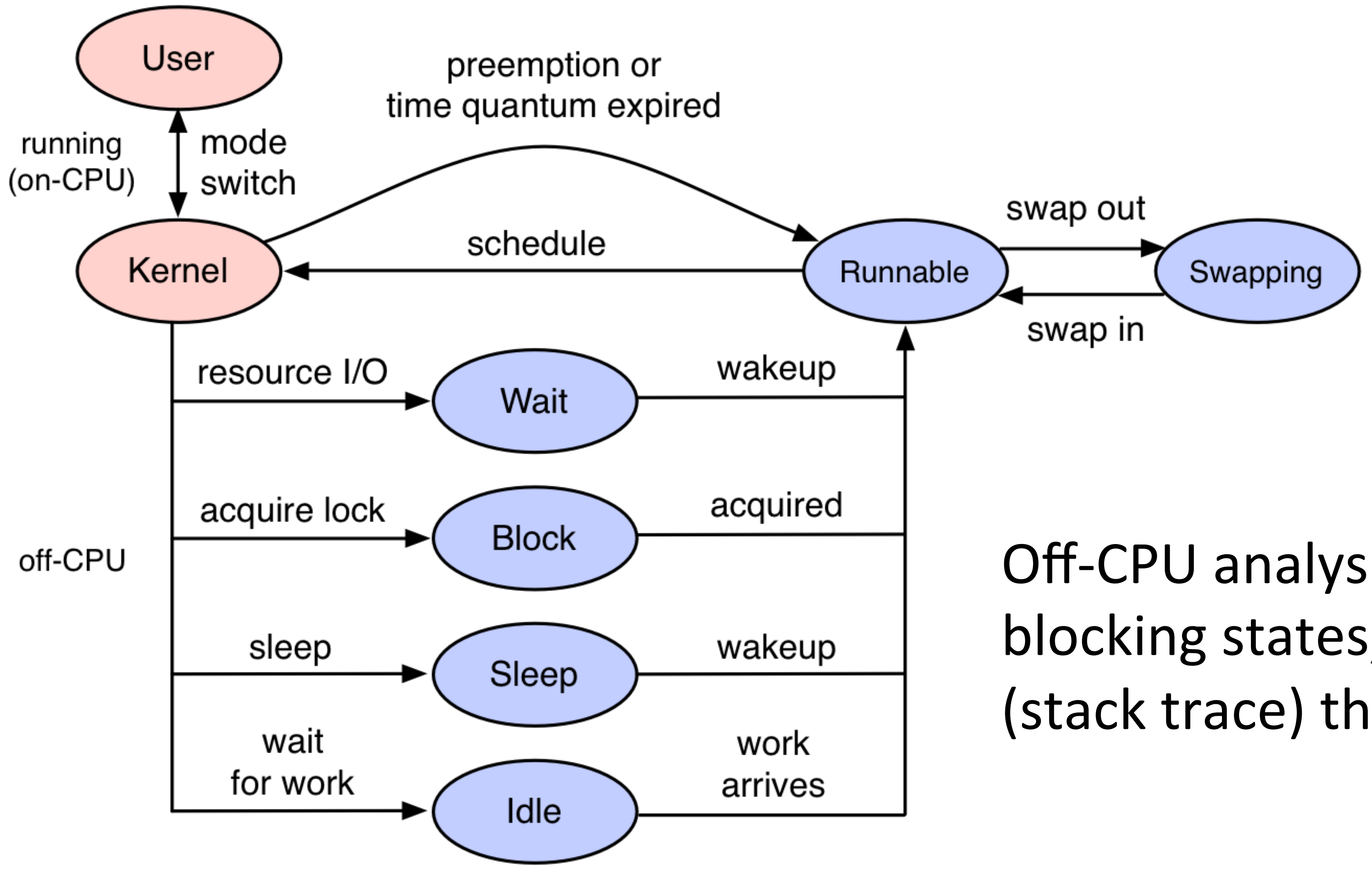
- Cycles Per Instruction
  - red == instruction heavy
  - blue == cycle heavy (likely memory stall cycles)

zoomed:



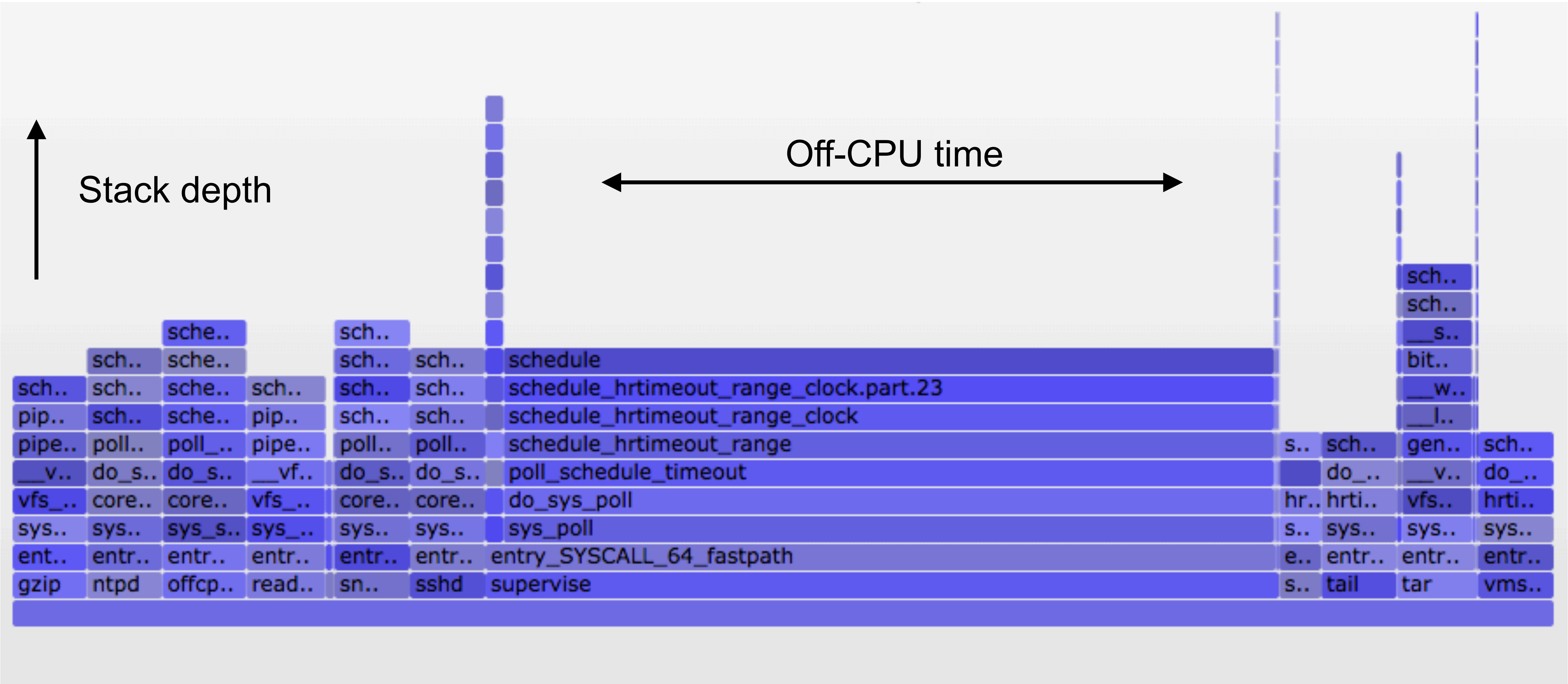


Off-CPU Analysis



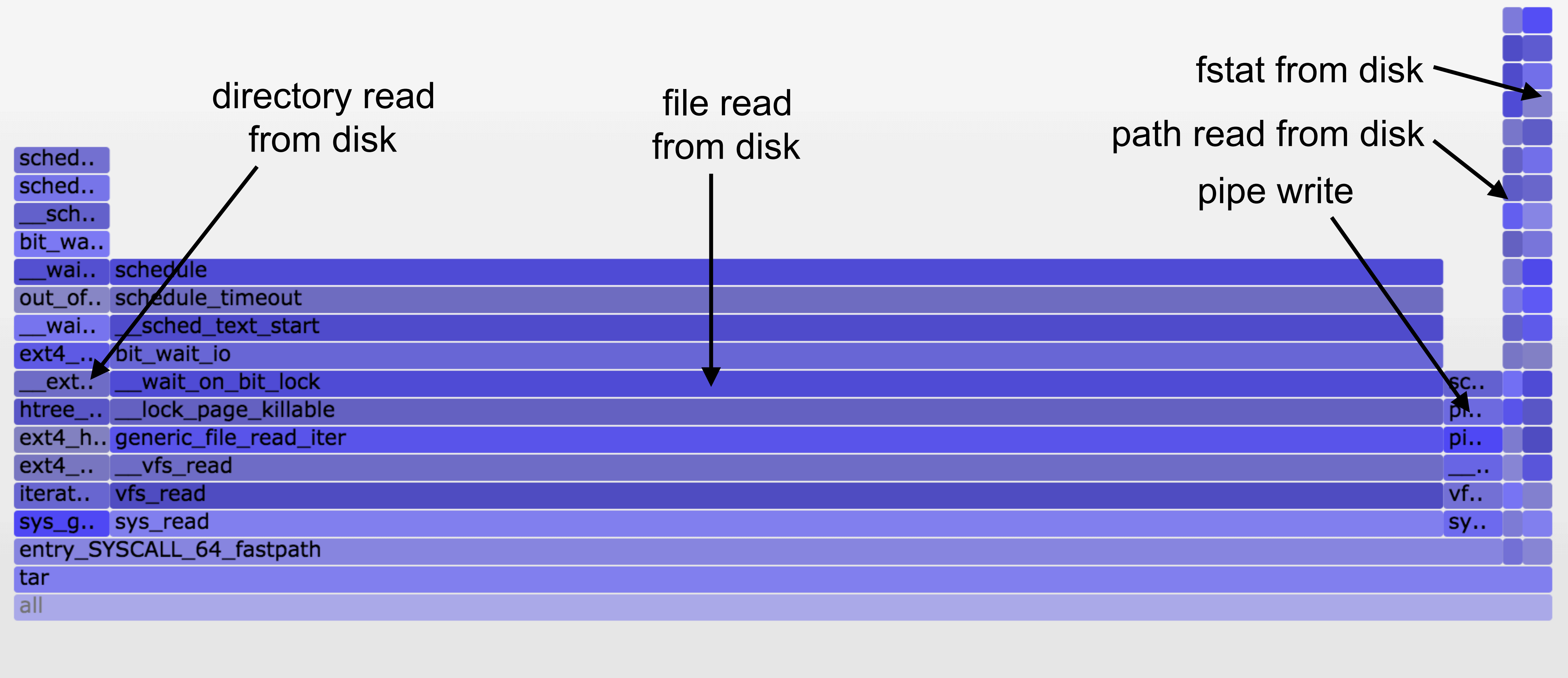
Off-CPU analysis is the study of blocking states, or the code-path (stack trace) that led to these states

Off-CPU Time Flame Graph



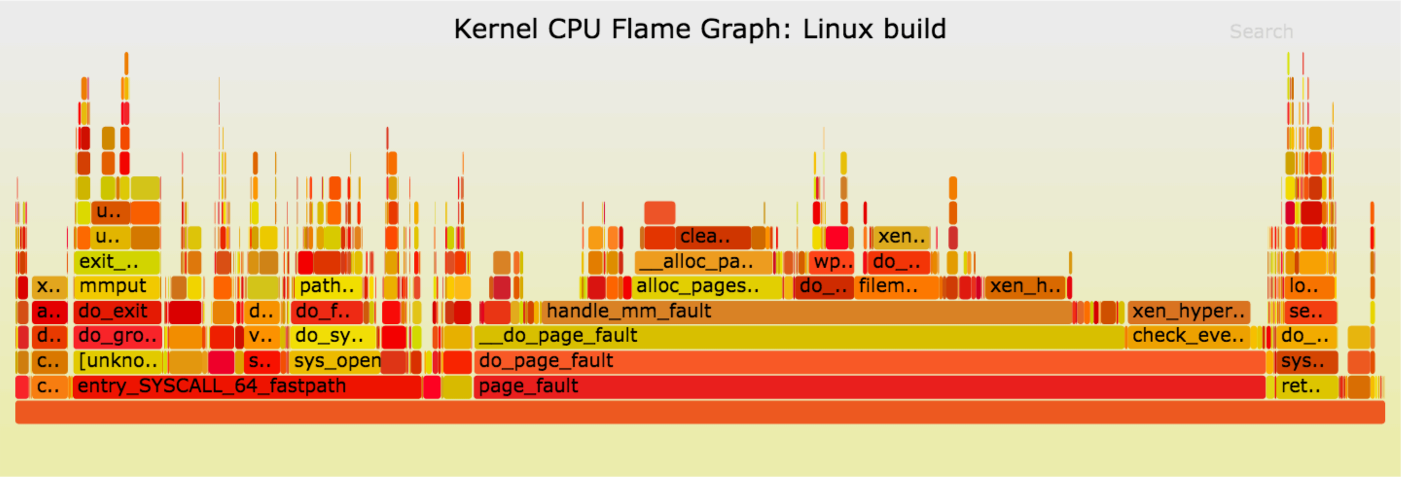
More info http://www.brendangregg.com/blog/2016-02-01/linux-wakeup-offwake-profiling.html

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

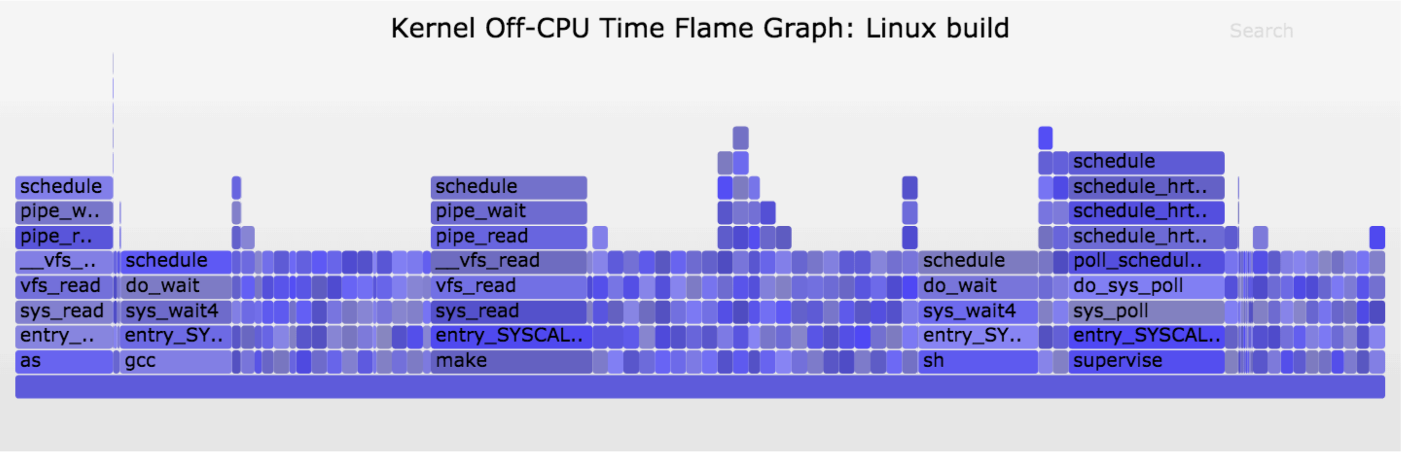


Only showing kernel stacks in this example

CPU + Off-CPU Flame Graphs: See Everything



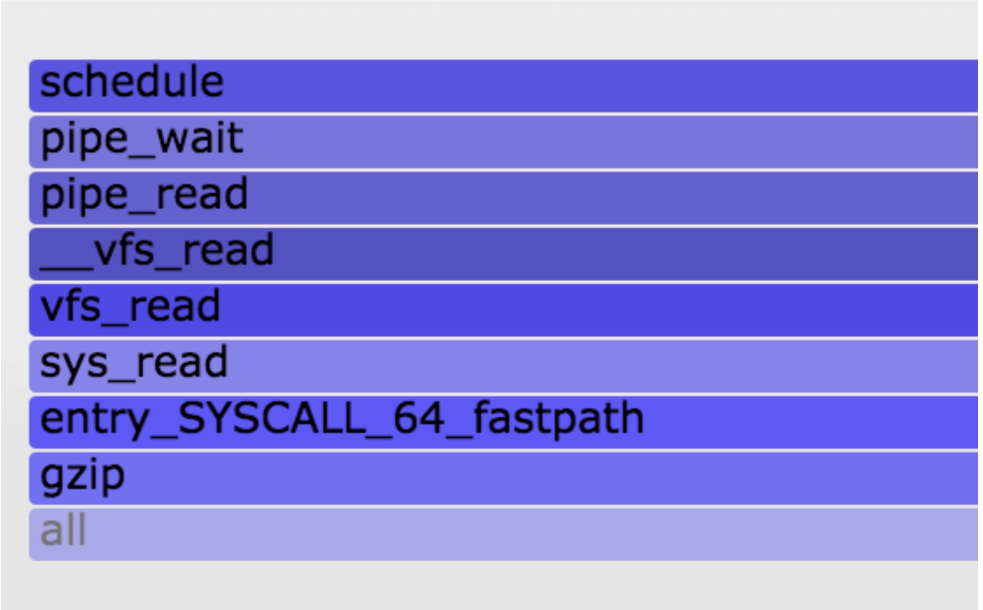
CPU



Off-CPU

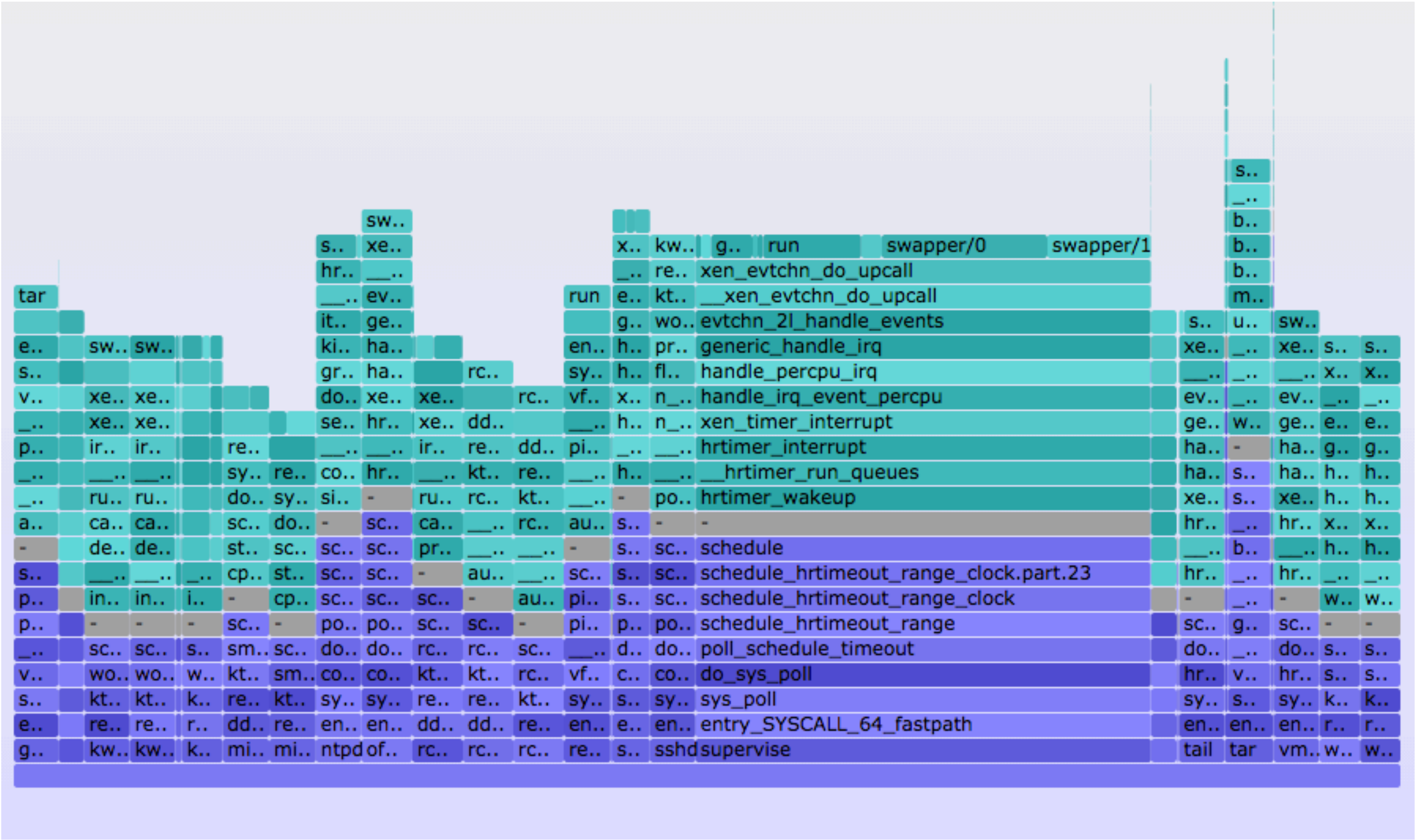
http://www.brendangregg.com/flamegraphs.html

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



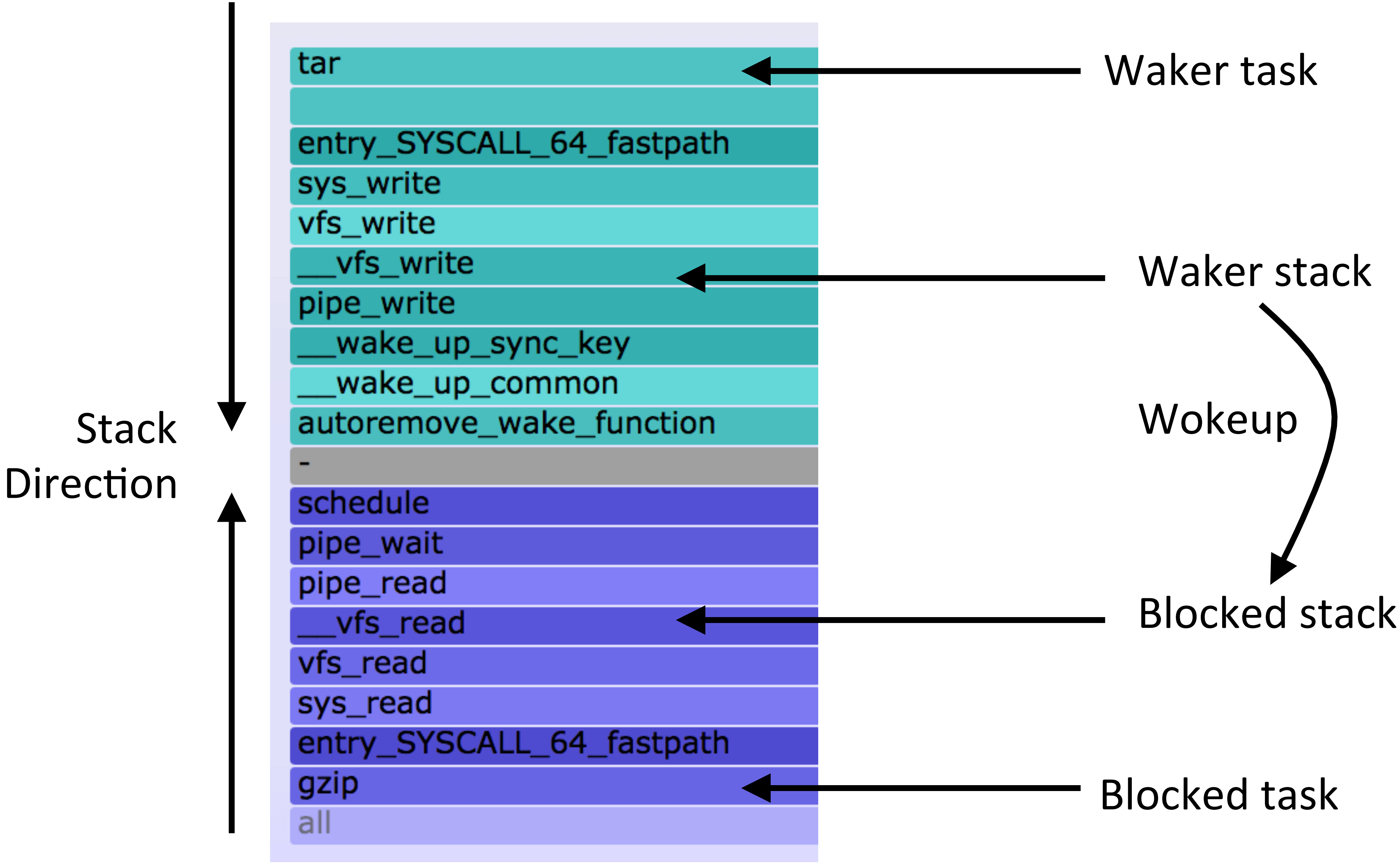
The off-CPU stack trace often doesn't show the root cause of latency.
What is gzip blocked on?

Off-Wake Time Flame Graph



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

Off-Wake Time Flame Graph (zoomed)



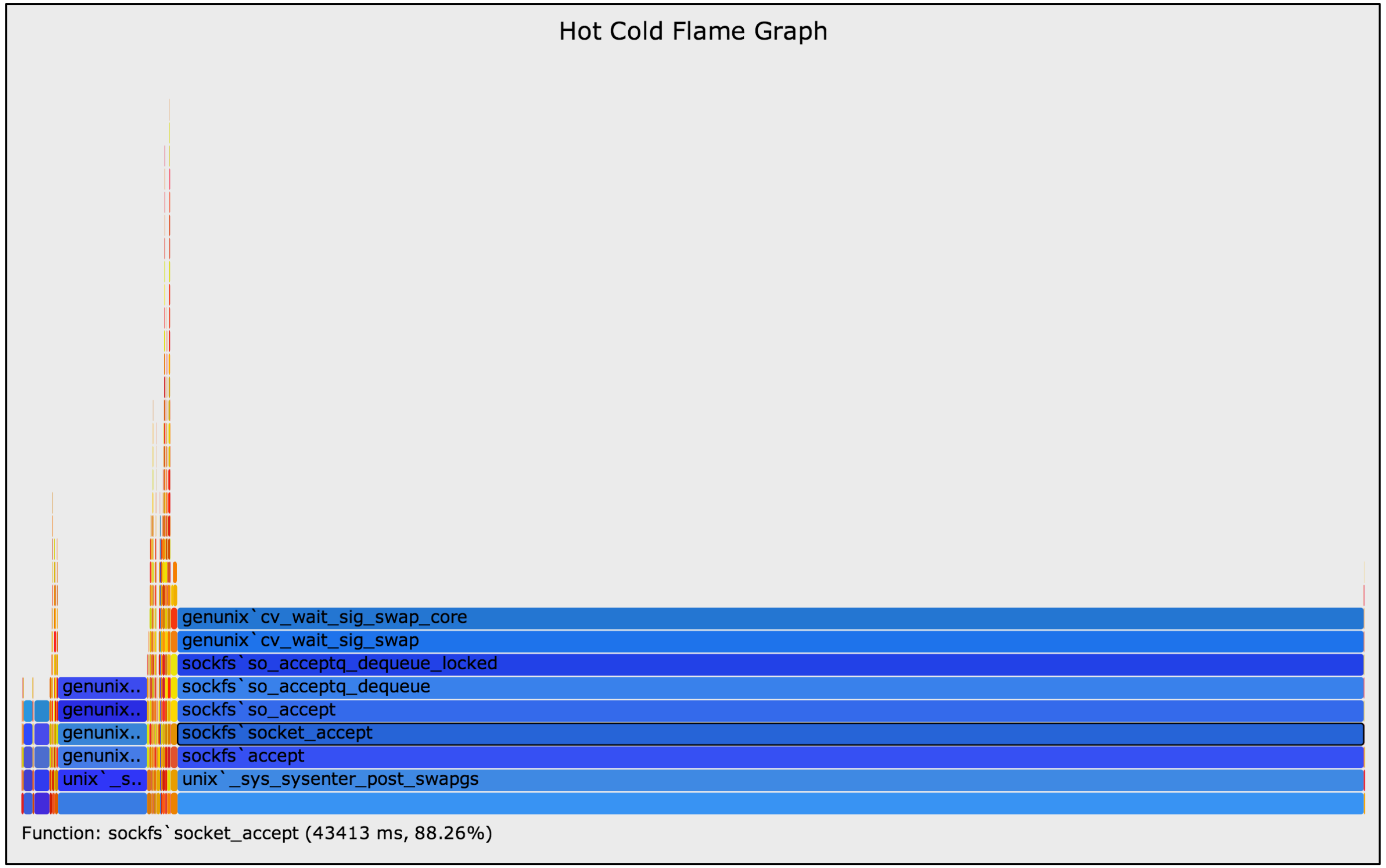
Chain Graphs

Walking the chain of wakeup stacks to reach root cause

Hot Cold Flame Graphs

Includes both CPU & Off-CPU (or chain) stacks in one flame graph

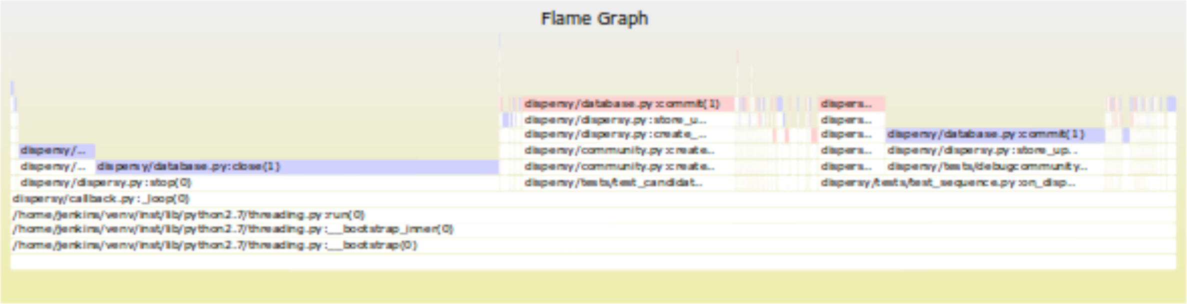
- However, Off-CPU time often dominates: threads waiting or polling



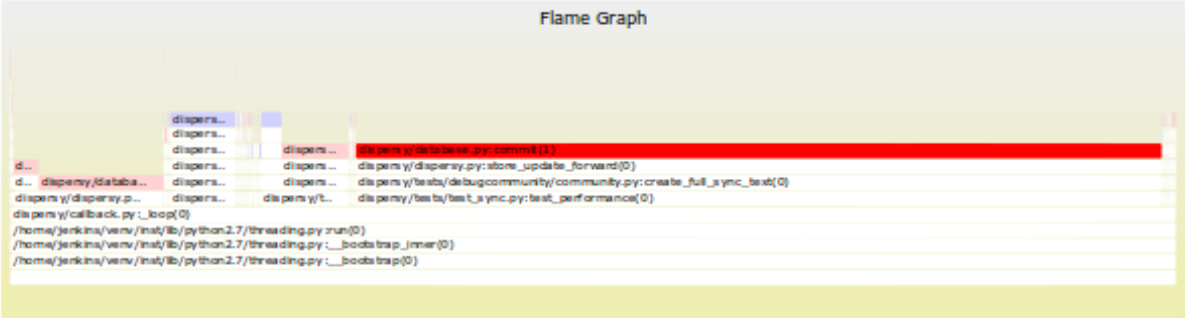
http://www.brendangregg.com/FlameGraphs/hotcoldflamegraphs.html

Flame Graph Diff

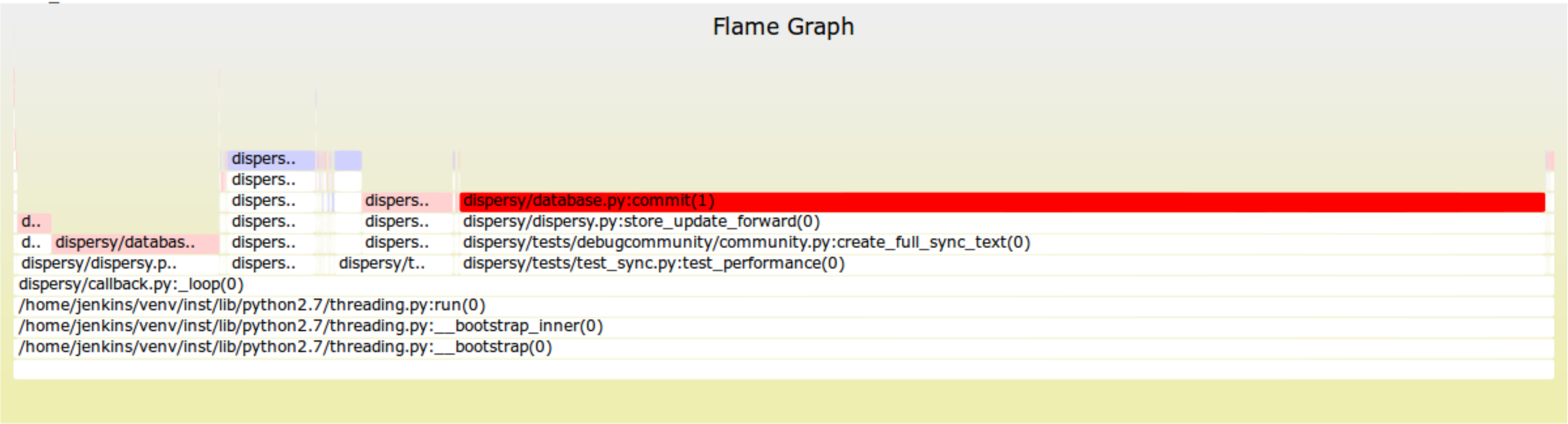
DFG1:



DFG2:



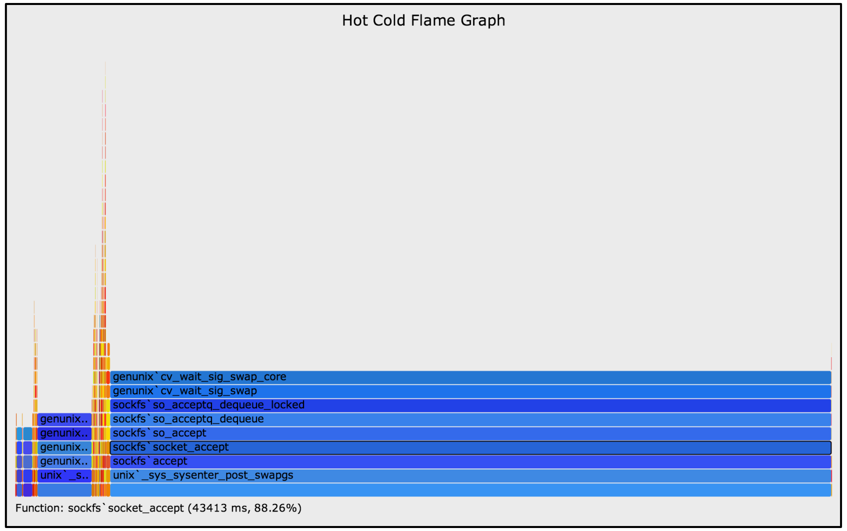
DFG_diff:

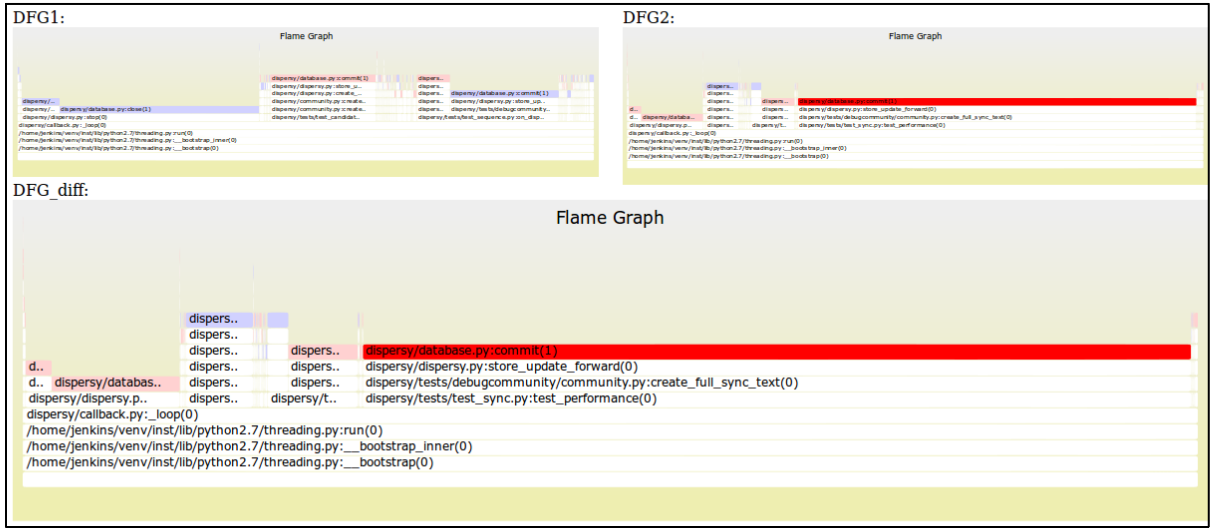


https://github.com/corpaul/flamegraphdiff

Take aways

1. Interpret CPU flame graphs
2. Understand pitfalls with stack traces and symbols
3. Discover opportunities for future development





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://techblog.netflix.com/2014/11/nodejs-in-flames.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
  - http://corpaul.github.io/flamegraphdiff/
- Linux perf_events
  - https://perf.wiki.kernel.org/index.php/Main_Page
  - http://www.brendangregg.com/perf.html
- Netflix Vector
  - https://github.com/netflix/vector
  - http://techblog.netflix.com/2015/04/introducing-vector-netflixs-on-host.html

Thank You

- Questions?
- http://www.brendangregg.com
- http://slideshare.net/brendangregg
- bgregg@netflix.com
- @brendangregg

Next topic: Performance Superpowers with Enhanced BPF

NETFLIX