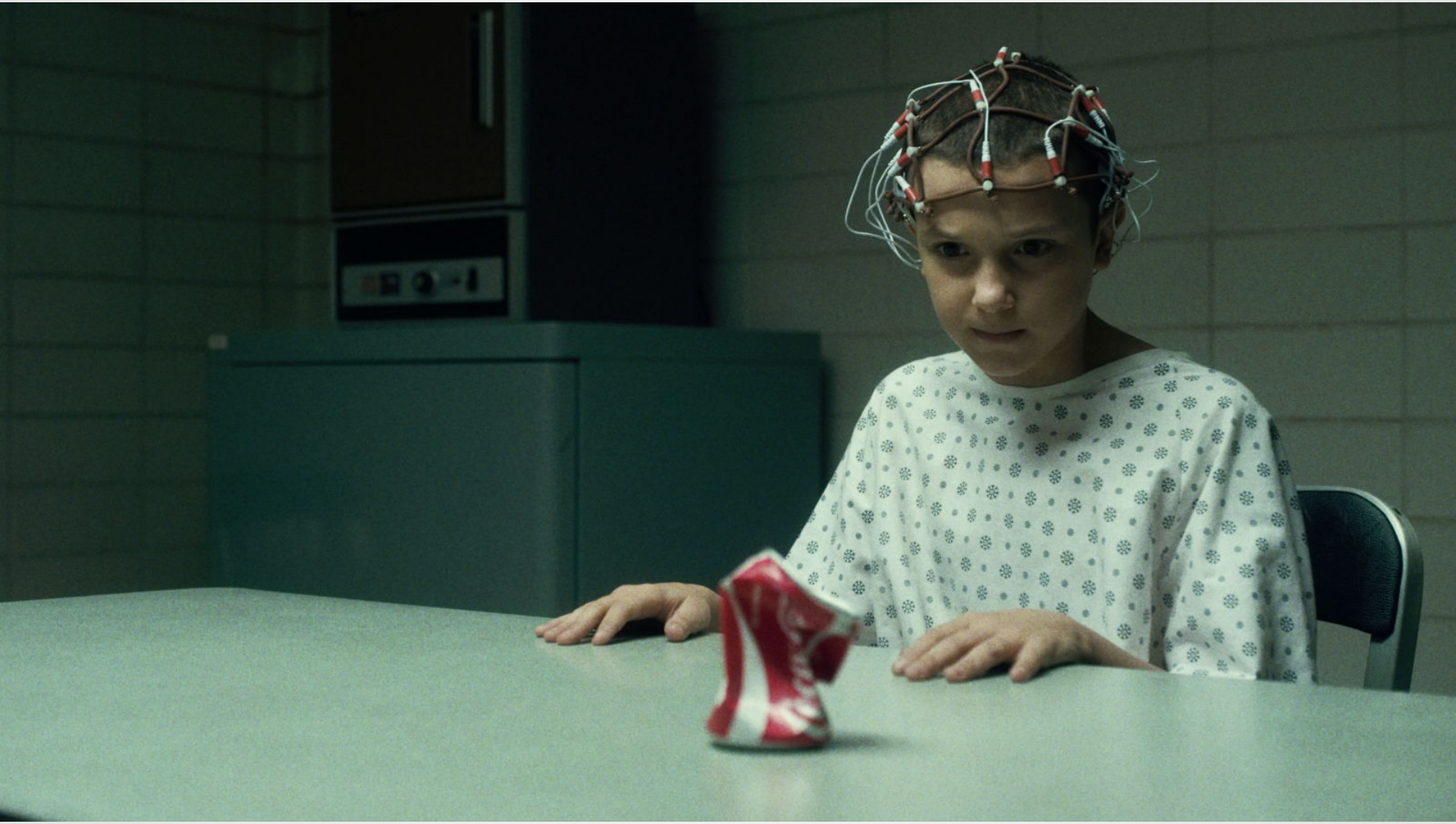
sysdig | Camp-Con-World-Fest-Summit

Designing Tracing Tools

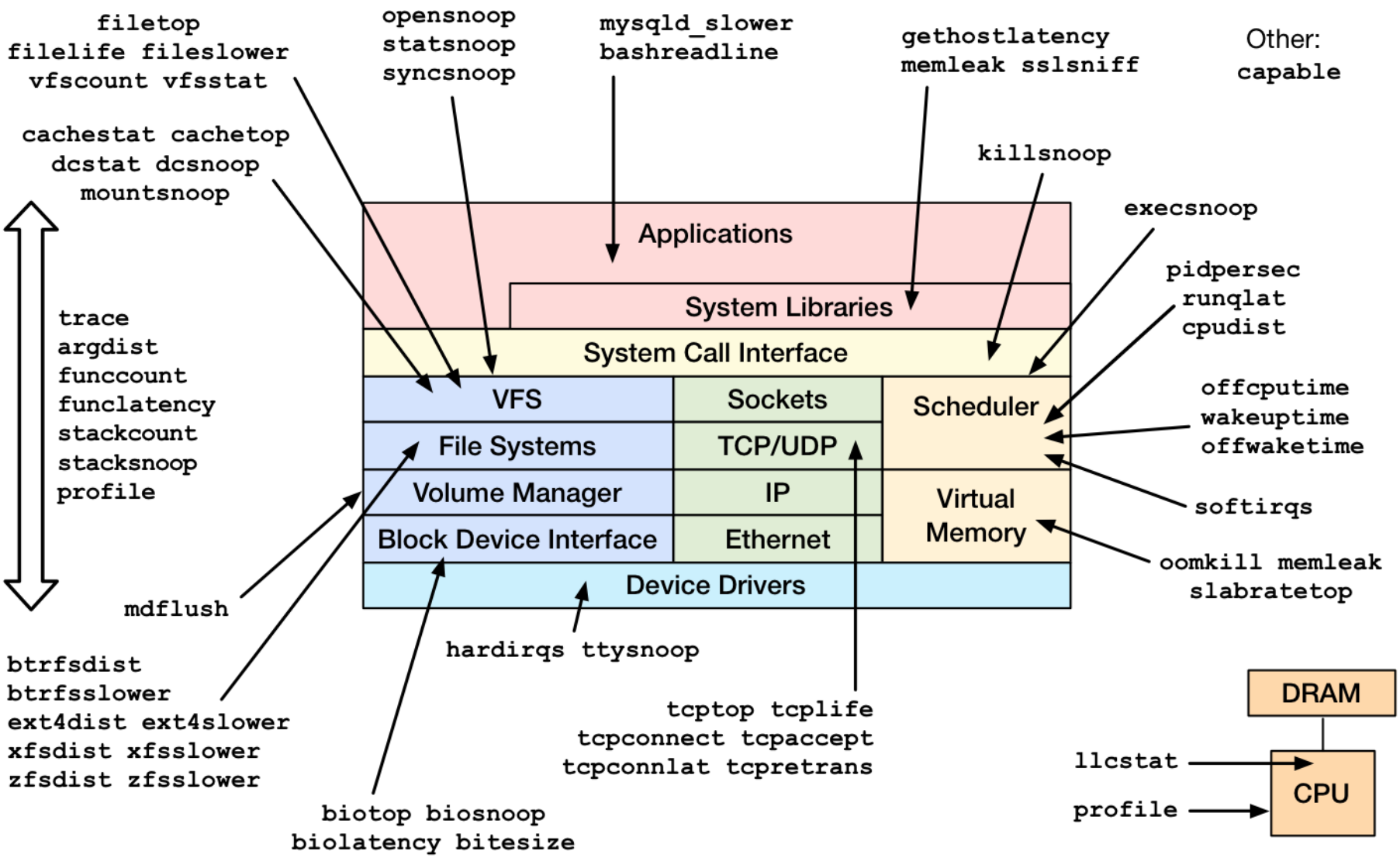
NETFLIX

Brendan Gregg, *Senior Performance Architect*

Wielding Superpowers



I'm currently developing more tracing tools (bcc/BPF)



https://github.com/iovisor/bcc#tools 2016

Tool Design

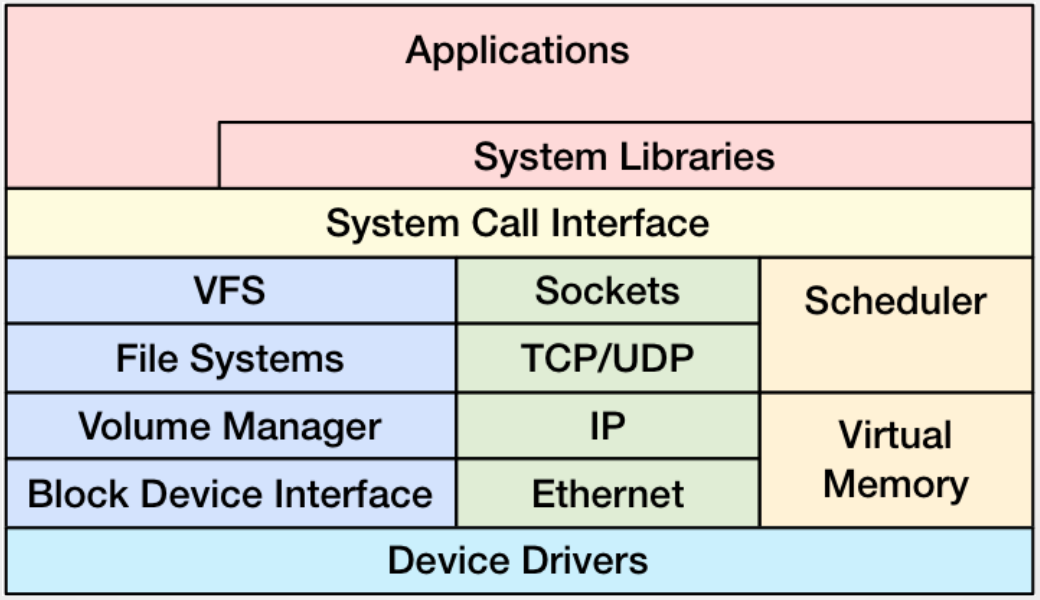
- For tool developers
- For everyone else: what you can ask for
  - Tool templates
  - GUI visualizations
- The following is applicable to all tracers
  - sysdig, bcc/BPF, DTrace, SystemTap, LTTng, …

Methodologies

Methodology-driven Design

- Methodologies provide ideas for purposeful tools
- Find/draw a functional diagram, apply methods

Operating Systems



See: http://www.brendangregg.com/methodology.html

Methodology Examples

Eg, at the syscall layer (well known & documented):

- Workload Characterization
  - exec() or open() per-event trace (execsnoop, opensnoop)
  - connect()/accept() per-event trace (tcpconnect, tcpaccept)
  - read()/write() size histogram (one-liners)
- Latency Analysis
  - read()/write() latency histogram (funclatency, …)
- USE Method
  - network utilization by thread (not done yet)
  - syscall errors (fserrors, soerrors)

CLI Tracing Tools

CLI Templates

1. Per event output
   - `*snoop, *slower 0, …`
2. Filtered event output
   - `*slower`
3. Interval summary
   - `*stat, *top`
4. Count summary
   - `*count`
5. Histogram summary
   - `*dist, *latency`
6. Heatmap summary
   - `spectrogram.lua, subsecoffset.lua, …`

Template 1: Per Event Output

Examples: `*snoop, *slower 0, …`

```
# opensnoop
PID    COMM               FD ERR PATH
10085  sshd                3   0 /lib/x86_64-linux-gnu/libkeyutils.so.1
10085  sshd                3   0 /lib/x86_64-linux-gnu/libresolv.so.2
10085  sshd                3   0 /lib/x86_64-linux-gnu/libgpg-error.so.0
10085  sshd                3   0 /dev/urandom
10085  sshd               -1   2 /lib/x86_64-linux-gnu/.libcrypto.so.1.0.0.hmac
10085  sshd               -1   2 /proc/sys/crypto/fips_enabled
10085  sshd                3   0 /proc/filesystems
10085  sshd                3   0 /dev/null
10085  sshd                3   0 /proc/10085/fd
10085  sshd                3   0 /usr/lib/ssl/openssl.cnf
10085  sshd                3   0 /etc/gai.conf
10085  sshd                3   0 /etc/nsswitch.conf
10085  sshd                3   0 /etc/ld.so.cache
10085  sshd                3   0 /lib/x86_64-linux-gnu/libnss_compat.so.2
10085  sshd                3   0 /etc/ld.so.cache
10085  sshd                3   0 /lib/x86_64-linux-gnu/libnss_nis.so.2
[…]
```

Template 2: Filtered Event Output

Examples: `*slower`

```
# sysdig -c fileslower 1
TIME                     PROCESS      TYPE     LAT(ms) FILE
2014-04-13 20:40:43.973 cksum        read           2 /mnt/partial.0.0
2014-04-13 20:40:44.187 cksum        read           1 /mnt/partial.0.0
2014-04-13 20:40:44.689 cksum        read           2 /mnt/partial.0.0
2014-04-13 20:40:45.005 cksum        read           2 /mnt/partial.0.0
2014-04-13 20:40:45.193 cksum        read           1 /mnt/partial.0.0
[…]
```

Tools like this can also do all event output:

```
# sysdig -c fileslower 0
TIME                     PROCESS      TYPE     LAT(ms) FILE
2014-04-13 20:59:04.414 ls           read           0 /lib/x86_64-linux-gnu/librt.so.1
2014-04-13 20:59:04.414 ls           read           0 /lib/x86_64-linux-gnu/libacl.so.1
2014-04-13 20:59:04.414 ls           read           0 /lib/x86_64-linux-gnu/libc.so.6
2014-04-13 20:59:04.414 ls           read           0 /lib/x86_64-linux-gnu/libdl.so.2
2014-04-13 20:59:04.414 ls           read           0 /lib/x86_64-linux-gnu/libattr.so.1
2014-04-13 20:59:04.415 ls           read           0 /proc/filesystems
2014-04-13 20:59:04.415 ls           read           0 /proc/filesystems
[...]
```

Template 3: Interval Summary

Examples: `*stat, *top`

```
# dcstat
TIME         REFS/s   SLOW/s   MISS/s    HIT%
08:11:47:      2059      141       97   95.29
08:11:48:     79974      151      106   99.87
08:11:49:    192874      146      102   99.95
08:11:50:      2051      144      100   95.12
08:11:51:     73373    17239    17194   76.57
08:11:52:     54685    25431    25387   53.58
08:11:53:     18127     8182     8137   55.12
08:11:54:     22517    10345    10301   54.25
08:11:55:      7524     2881     2836   62.31
08:11:56:      2067      141       97   95.31
08:11:57:      2115      145      101   95.22
[…]
```

Template 4: Count Summary

Examples: `*count`

```
# funccount 'vfs_*'
Tracing... Ctrl-C to end.
^C
ADDR             FUNC                          COUNT
ffffffff811efe81 vfs_create                        1
ffffffff811f24a1 vfs_rename                        1
ffffffff81215191 vfs_fsync_range                   2
ffffffff81231df1 vfs_lock_file                    30
ffffffff811e8dd1 vfs_fstatat                     152
ffffffff811e8d71 vfs_fstat                       154
ffffffff811e4381 vfs_write                       166
ffffffff811e8c71 vfs_getattr_nosec               262
ffffffff811e8d41 vfs_getattr                     262
ffffffff811e3221 vfs_open                        264
ffffffff811e4251 vfs_read                        470
Detaching...
```

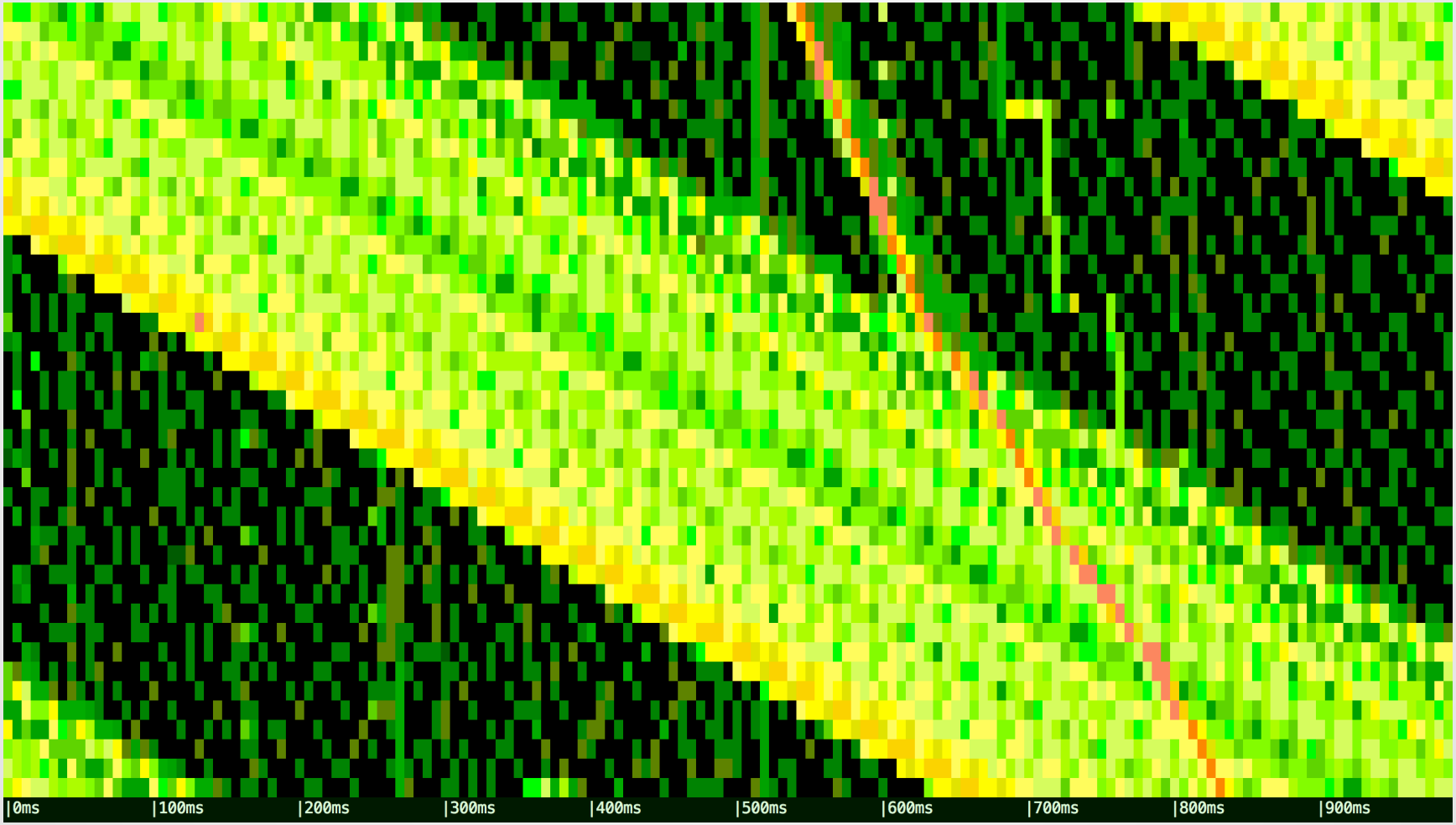
Template 5: Histogram Summary

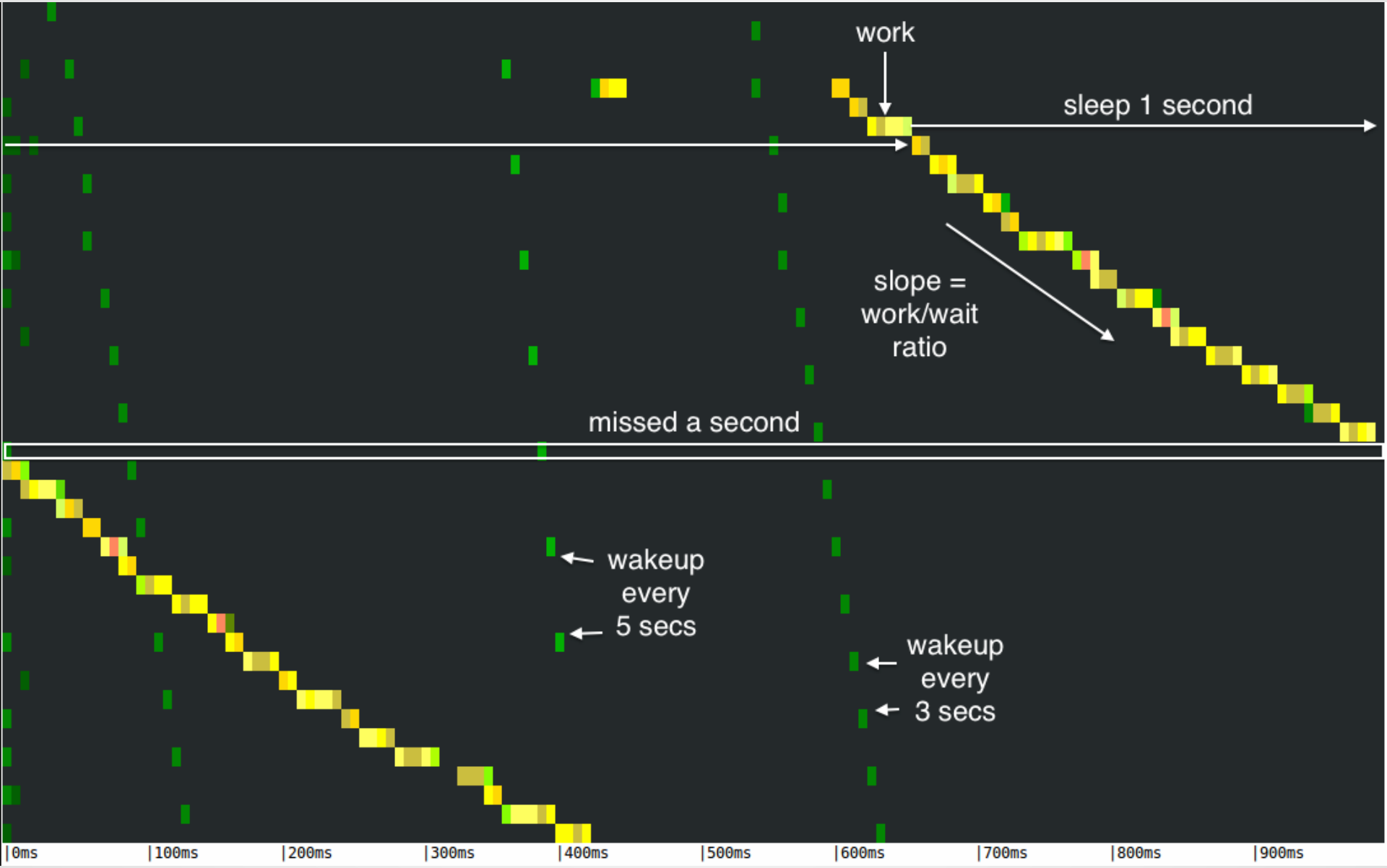
Examples: `*dist, *latency`

```
# biolatency
Tracing block device I/O... Hit Ctrl-C to end.
^C
     usecs               : count     distribution
       4 -> 7            : 0        |                                        |
       8 -> 15           : 0        |                                        |
      16 -> 31           : 0        |                                        |
      32 -> 63           : 0        |                                        |
      64 -> 127          : 1        |                                        |
     128 -> 255          : 12       |********                                |
     256 -> 511          : 15       |**********                              |
     512 -> 1023         : 43       |******************************          |
    1024 -> 2047         : 52       |************************************    |
    2048 -> 4095         : 47       |********************************        |
    4096 -> 8191         : 52       |************************************    |
    8192 -> 16383        : 36       |*************************               |
   16384 -> 32767        : 15       |**********                              |
   32768 -> 65535        : 2        |*                                       |
   65536 -> 131071       : 2        |*                                       |
```

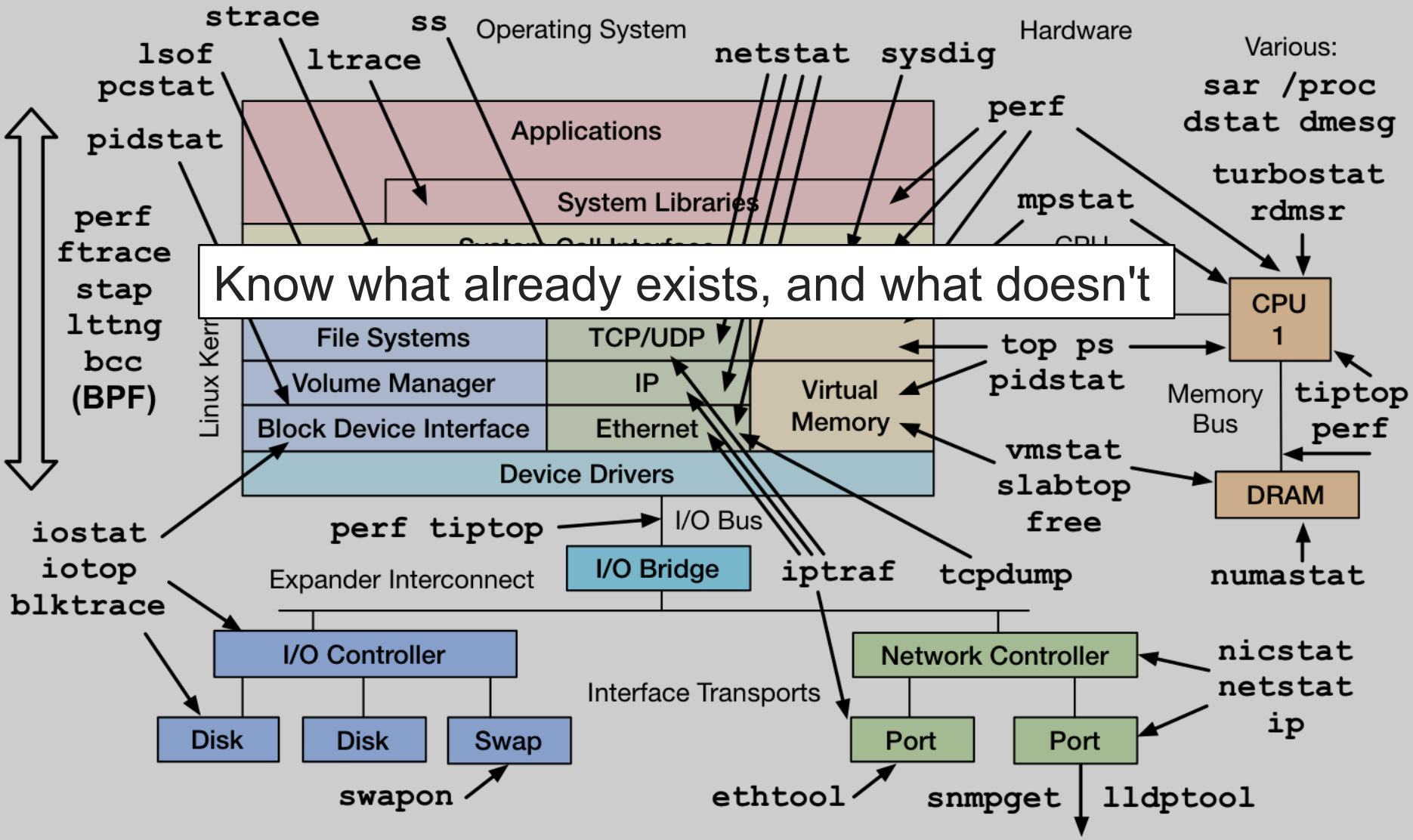
Template 6: Heatmap Summary

Example: `subsecoffset.lua` (aka "spectrogram")

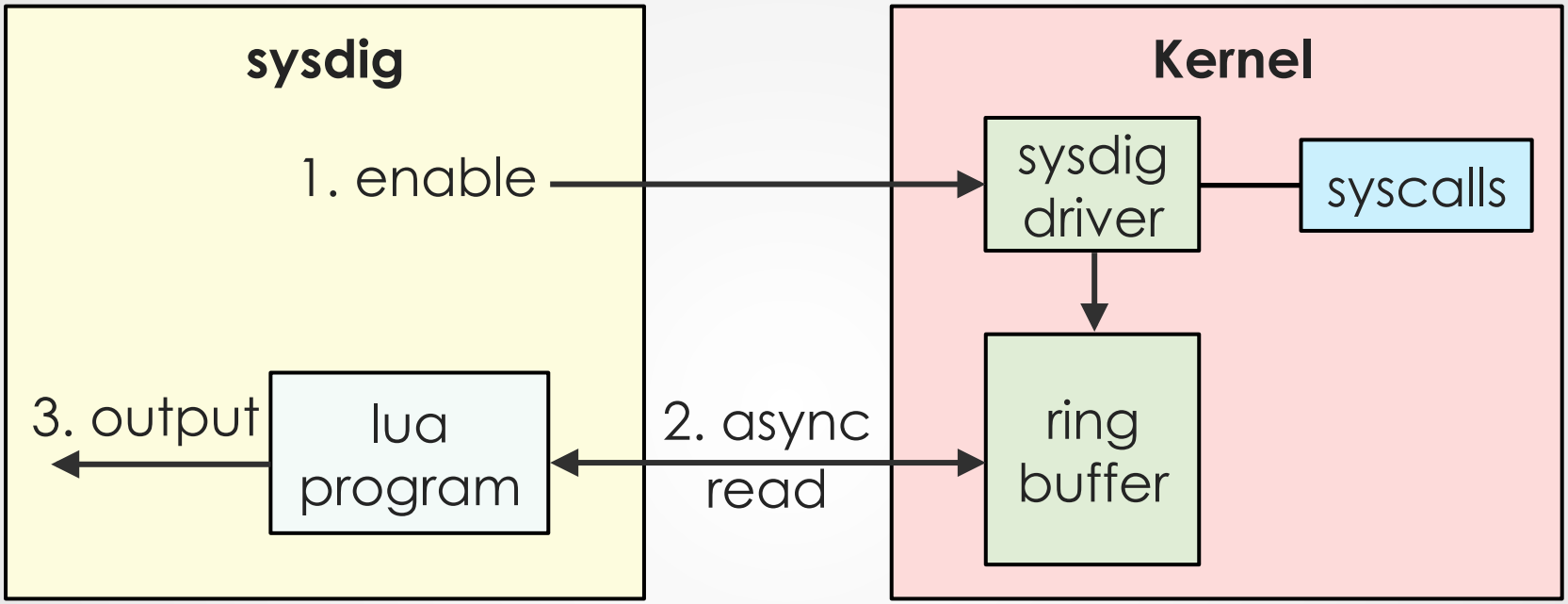



work
sleep 1 second
slope =
work/wait
ratio
missed a second
wakeup
every
5 secs
wakeup
every
3 secs
|0ms
|100ms
|200ms
|300ms
|400ms
|500ms
|600ms
|700ms
|800ms
|900ms

Valuable


strace
ss
Operating System
Hardware
lsof
pcstat
ltrace
netstat
sysdig
Various:
sar /proc
dstat dmesg
pidstat
perf
Applications
System Libraries
turbostat
rdmsr
mpstat
perf
ftrace
stap
lttng
bcc
(BPF)
Know what already exists, and what doesn't
CPU
1
File Systems
TCP/UDP
top ps
pidstat
Volume Manager
IP
Virtual
Memory
Memory
Bus
tiptop
perf
Block Device Interface
Ethernet
vmstat
slabtop
free
Device Drivers
DRAM
iostat
iotop
blktrace
perf tiptop
I/O Bus
I/O Bridge
iptraf
tcpdump
numastat
Expander Interconnect
I/O Controller
Network Controller
nicstat
netstat
ip
Interface Transports
Disk
Disk
Swap
Port
Port
swapon
ethtool
snmpget
lldptool

Low Overhead (or documented)



- Understand tracing internals
  - For example, sysdig's design has ~20x lower overhead than strace (it still has overhead: test and measure to see if it's acceptable)
  - Tracing overhead is usually relative to event rate
- Design for low overhead, and document expectations

Documentation

- Good tools have 3 docs:
  1. Code comments
  2. Man page
  3. Examples file
- Man page
  - troff, docbook, …
- Examples file:

.TH	**Title heading**
.SH	Section heading
.IP	Indented paragraph
.TP	Indented paragraph with label
.B	Bold
\-	-

common man macros (see groff_man(7))

```
Demonstrations of biosnoop, the Linux eBPF/bcc version.

biosnoop traces block device I/O (disk I/O), and prints a line of output
per I/O. Example:

# ./biosnoop
TIME(s)         COMM           PID    DISK    T  SECTOR    BYTES   LAT(ms)
0.000004001     supervise      1950   xvda1   W  13092560  4096       0.74
[...]
```

Concise, intuitive, self-explanatory

```
# ./iolatency
Tracing block I/O. Output every 1 seconds. Ctrl-C to end.

  >=(ms) .. <(ms)    : I/O       |Distribution                              |
         0 -> 1      : 4381      |########################################|
         1 -> 2      : 9         |#                                       |
         2 -> 4      : 5         |#                                       |
         4 -> 8      : 0         |                                        |
         8 -> 16     : 1         |#                                       |
[…]
```

- Useful startup message
  - What I'm tracing, when there's output, when I'll end
- Vigorous tooling is concise
  - No wasted text; leave less useful output for non-default options
- Unix philosophy: do one thing and do it well

POSIX-style Arguments

```
# ./biolatency -h
usage: biolatency [-h] [-T] [-Q] [-m] [-D] [interval] [count]

Summarize block device I/O latency as a histogram

positional arguments:
  interval            output interval, in seconds
  count               number of outputs

optional arguments:
  -h, --help          show this help message and exit
  -T, --timestamp     include timestamp on output
  -Q, --queued        include OS queued time in I/O time
  -m, --milliseconds  millisecond histogram
  -D, --disks         print a histogram per disk device

examples:
    ./biolatency            # summarize block I/O latency as a histogram
    ./biolatency 1 10       # print 1 second summaries, 10 times
    ./biolatency -mT 1      # 1s summaries, milliseconds, and timestamps
    ./biolatency -Q         # include OS queued time in I/O time
    ./biolatency -D         # show each disk device separately
```

POSIX-style Arguments

Option	Alternate	Expectation
-a	--all	all events
-c CMD	--cmd …	run this command
-d SECONDS	--duration …	duration of tool execution
-h	--help	help
-i FILE	--input …	input file
-i SECONDS	--interval …	summary interval
-n name	--name …	this process name only
-o FILE	--output …	output file
-p PID	--pid …	this process ID only
-P	--by-process	per-process ID breakdown
-P PORT	--port …	this TCP port only
-t or -T	--[no]timestamp	include or exclude timestamps
-v	--verbose	verbose output
-x	--extended, --errors	extended output, or only failures
[interval [count]]	-	summary interval, and # of outputs

Testing, Testing, Testing

- If you can't write the workload, you can't write the tool
  - Be it 10 lines of C, some shell, or dd
  - dd if=/dev/urandom of=/dev/null bs=1k count=23
- Known workload testing: create 23 events
- Testing can be time consuming
  - I spend more time testing a tool than writing it
  - Automatic tool testing is a difficult problem

Example: gethostlatency

```
# gethostlatency
TIME      PID    COMM            LATms HOST
06:10:24  28011  wget            90.00 www.iovisor.org
06:10:28  28127  wget             0.00 www.iovisor.org
06:10:41  28404  wget             9.00 www.netflix.com
06:10:48  28544  curl            35.00 www.netflix.com.au
06:11:10  29054  curl            31.00 www.plumgrid.com
06:11:16  29195  curl             3.00 www.facebook.com
06:11:25  29404  curl            72.00 foo
06:11:28  29475  curl             1.00 foo
```

Example: ext4slower

```
# ext4slower 1
Tracing ext4 operations slower than 1 ms
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
06:49:17 bash           3616   R 128     0           7.75 cksum
06:49:17 cksum          3616   R 39552   0           1.34 [
06:49:17 cksum          3616   R 96      0           5.36 2to3-2.7
06:49:17 cksum          3616   R 96      0          14.94 2to3-3.4
06:49:17 cksum          3616   R 10320   0           6.82 411toppm
06:49:17 cksum          3616   R 65536   0           4.01 a2p
06:49:17 cksum          3616   R 55400   0           8.77 ab
06:49:17 cksum          3616   R 36792   0          16.34 aclocal-1.14
06:49:17 cksum          3616   R 15008   0          19.31 acpi_listen
06:49:17 cksum          3616   R 6123    0          17.23 add-apt-
repository
06:49:17 cksum          3616   R 6280    0          18.40 addpart
06:49:17 cksum          3616   R 27696   0           2.16 addr2line
06:49:17 cksum          3616   R 58080   0          10.11 ag
06:49:17 cksum          3616   R 906     0           6.30 ec2-meta-data
[…]
```

Example: biolatency

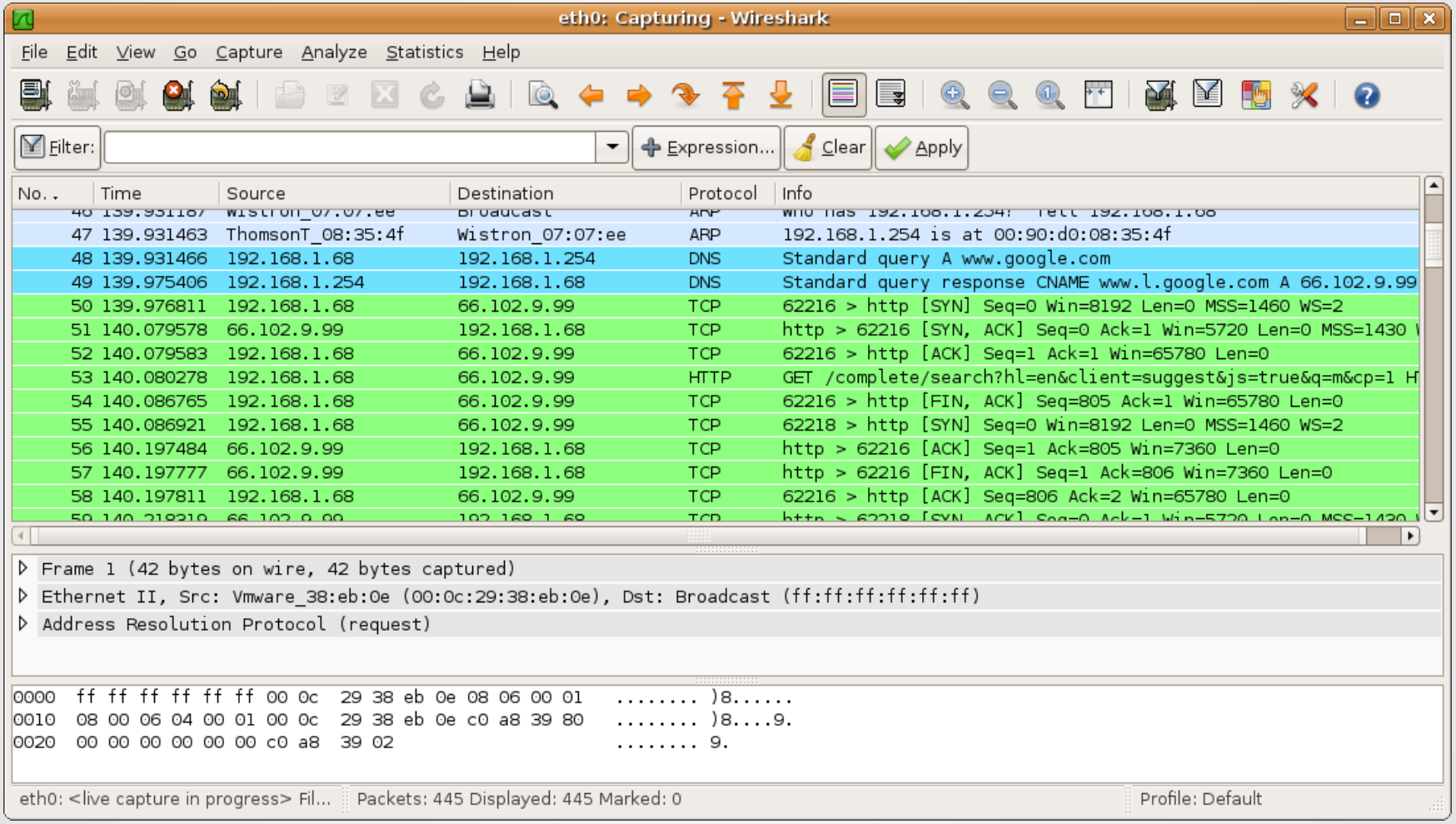
```
# biolatency -m 1 5
Tracing block device I/O... Hit Ctrl-C to end.

     msecs               : count     distribution
       0 -> 1            : 36       |****************************************|
       2 -> 3            : 1        |*                                       |
       4 -> 7            : 3        |***                                     |
       8 -> 15           : 17       |*****************                       |
      16 -> 31           : 33       |********************************        |
      32 -> 63           : 7        |*******                                 |
      64 -> 127          : 6        |******                                  |
[…]
```

GUI Tracing Tools

GUI Visualizations

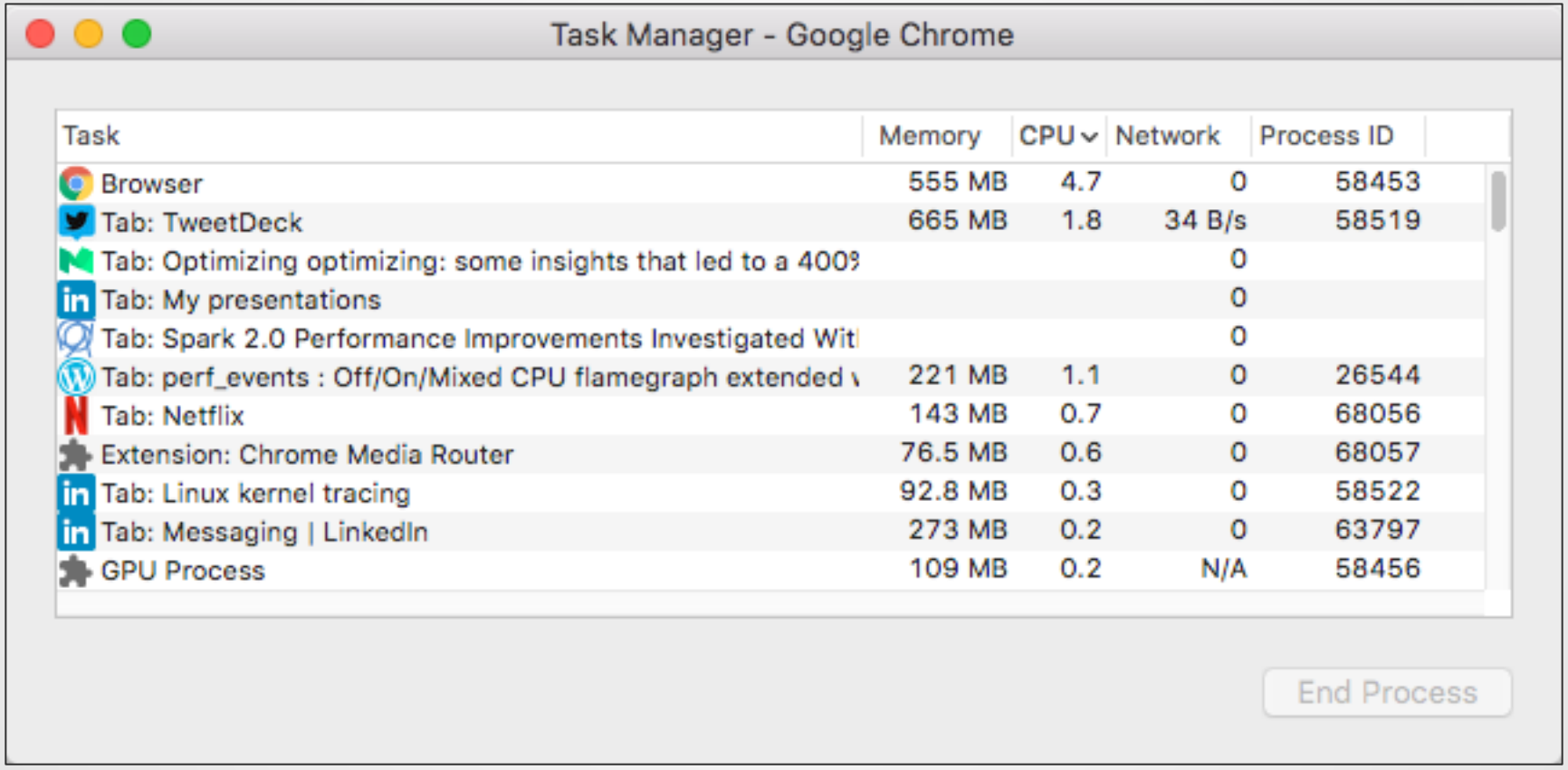
1. Event logs
2. Tables
3. Line graphs
4. Histograms
5. Heatmaps (spectrographs)
6. Waterfall charts
7. Directed graphs
8. Flame graphs
9. Flame charts

Visualization 1: Event Logs

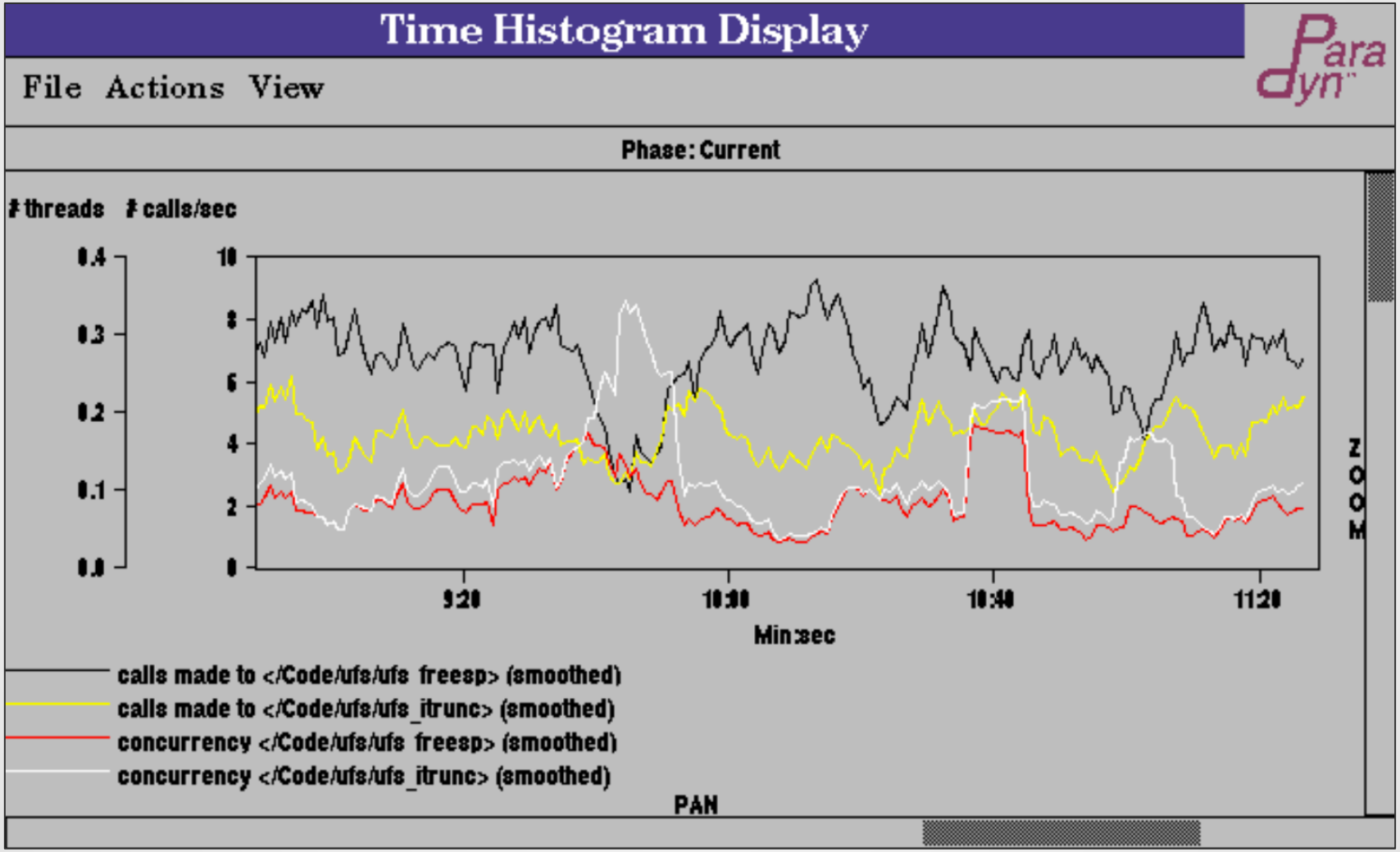


https://commons.wikimedia.org/wiki/File:Wireshark_screenshot.png

Visualization 2: Tables

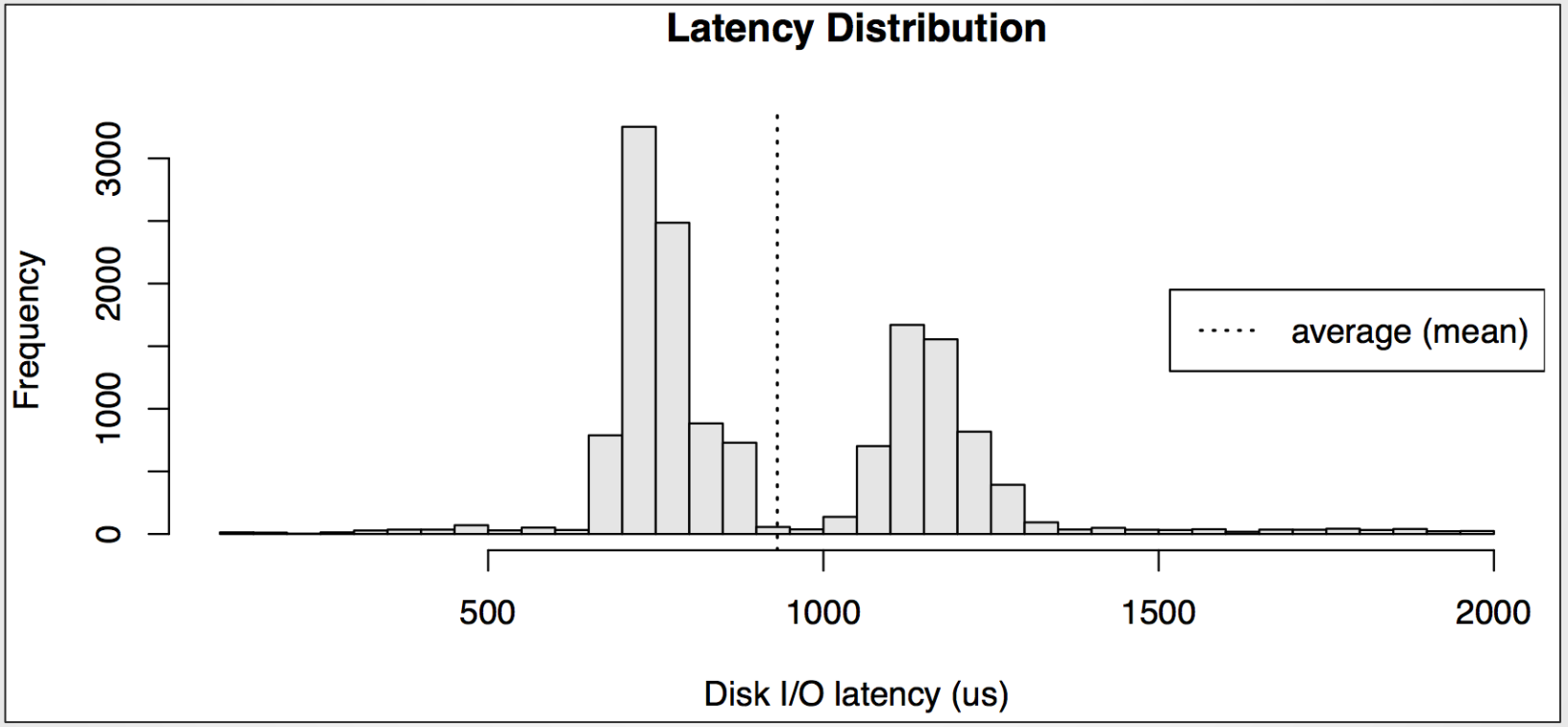


Visualization 3: Line Graphs



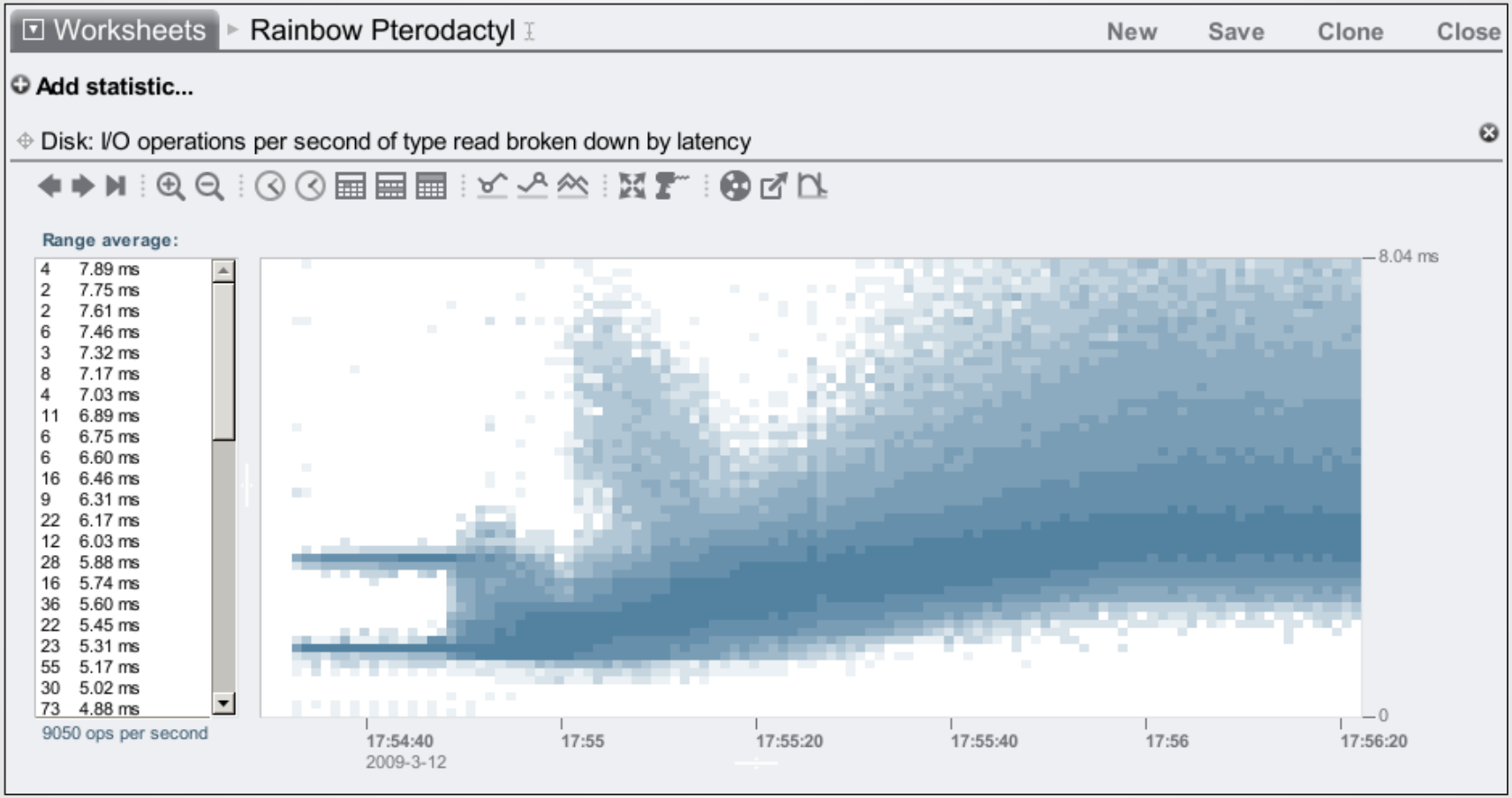
http://www.paradyn.org/html/screen-shots.html

Visualization 4: Histograms



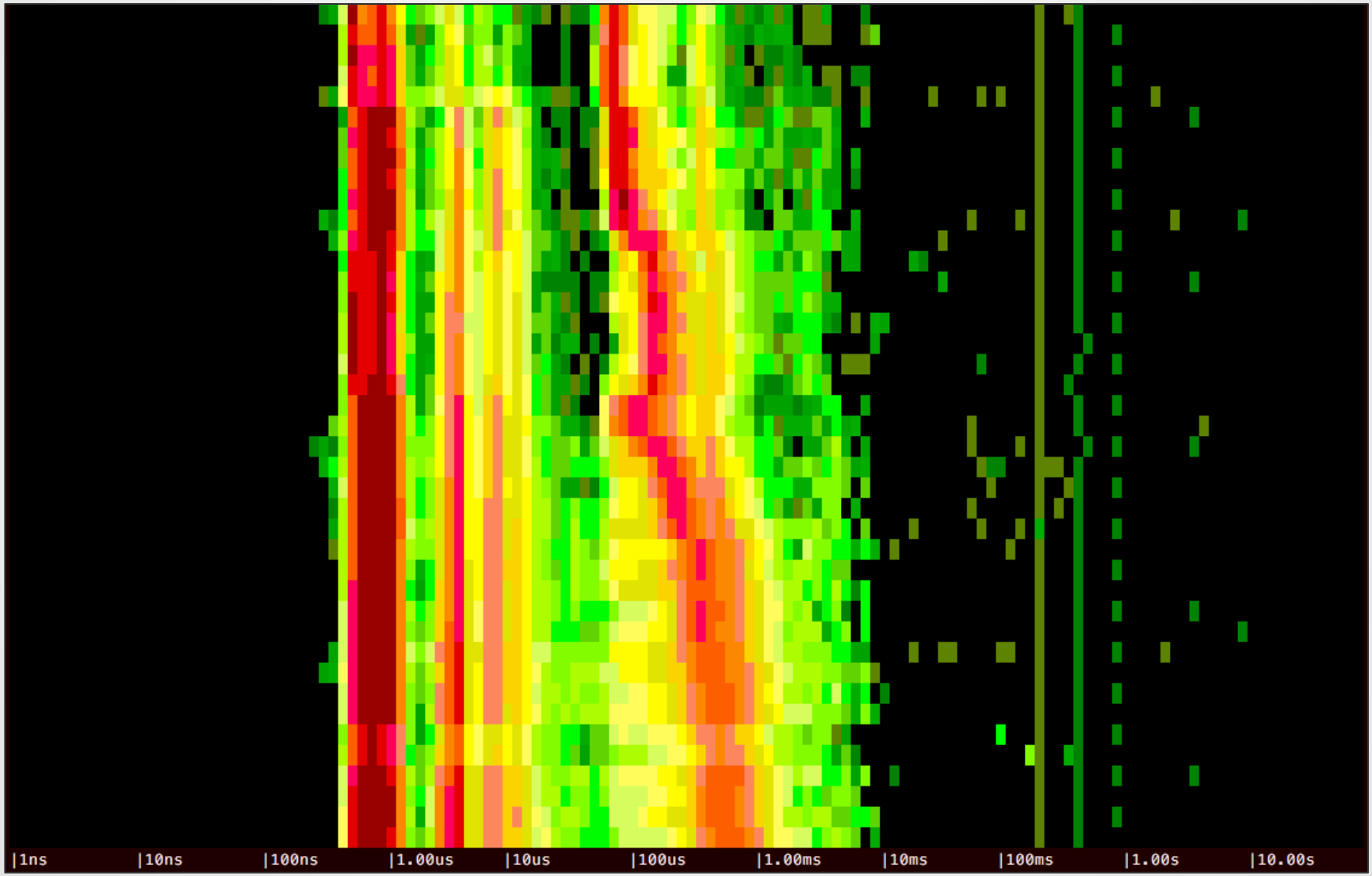
Or a density plot
Or as a frequency trail (can cascade)

Visualization 5: Heat Maps

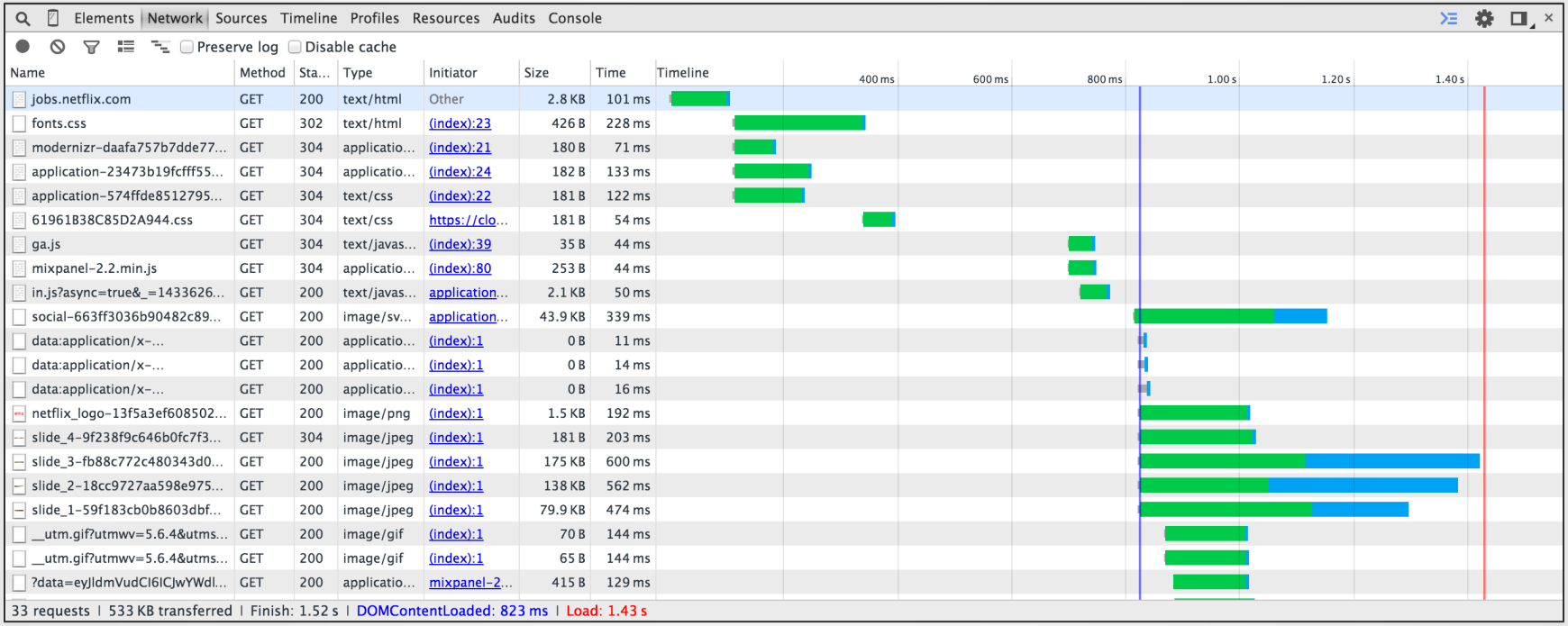


eg, Oracle ZFS Storage Appliance Analytics (DTrace-based)

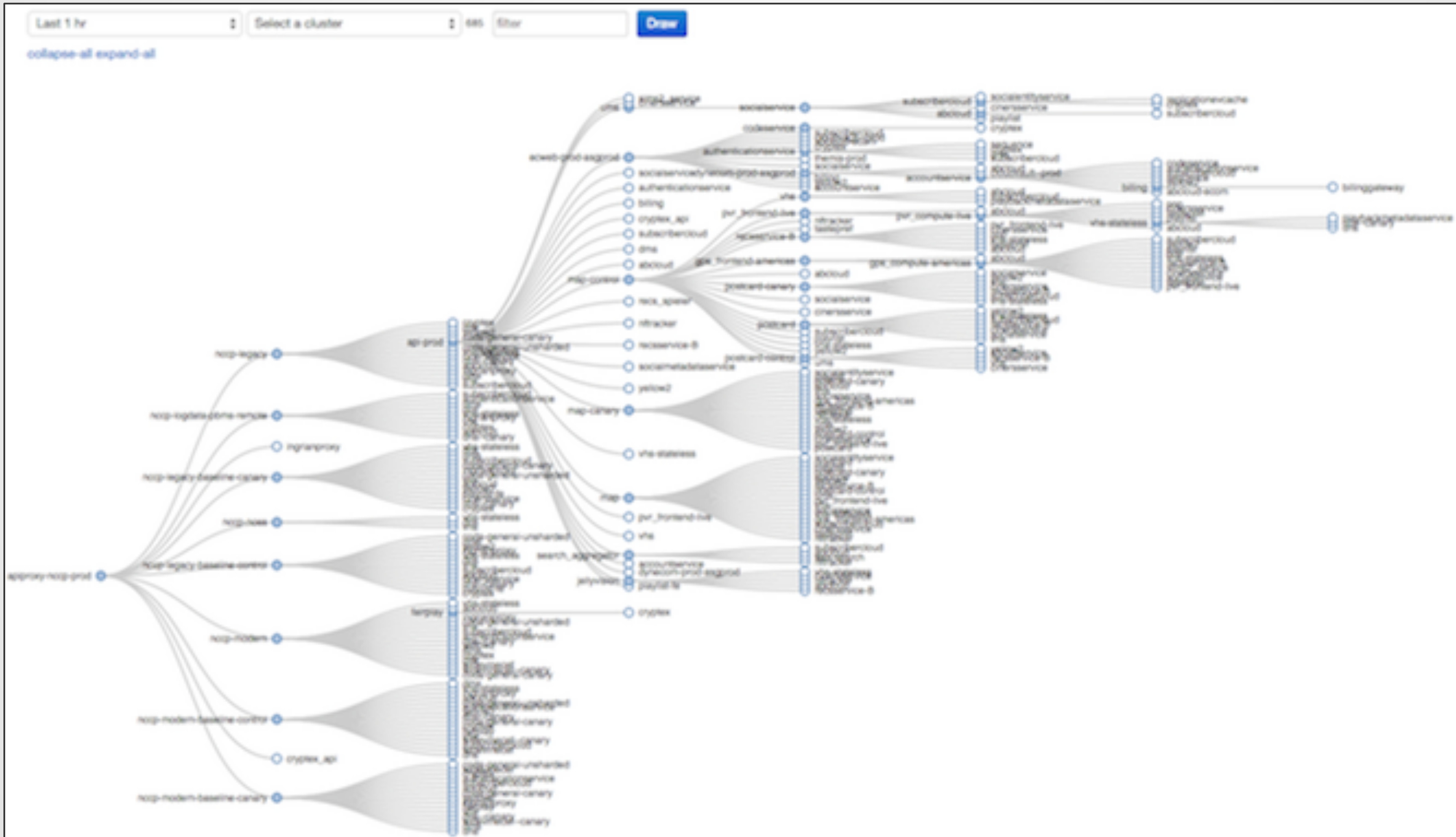
Visualization 5: Spectrograms



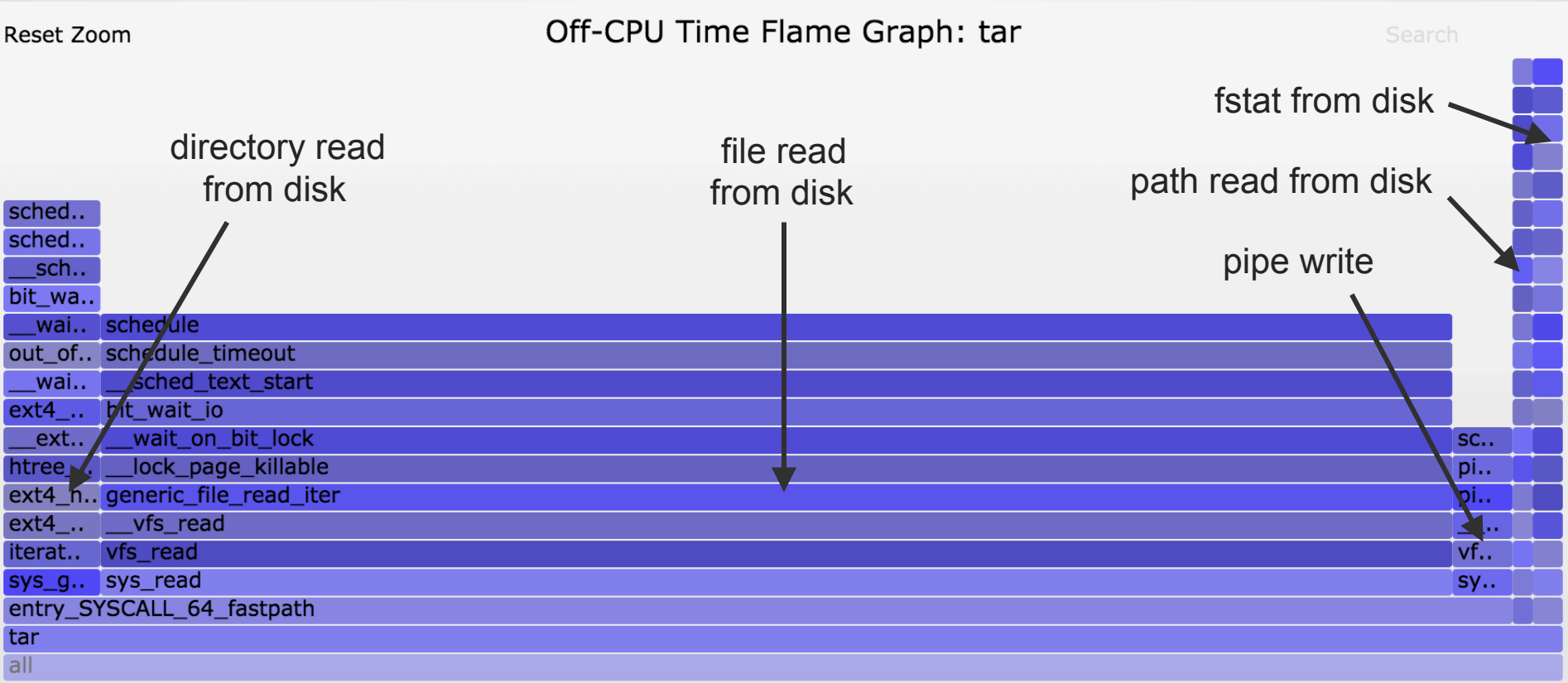
Visualization 6: Waterfall Charts



Visualization 7: Directed Graphs

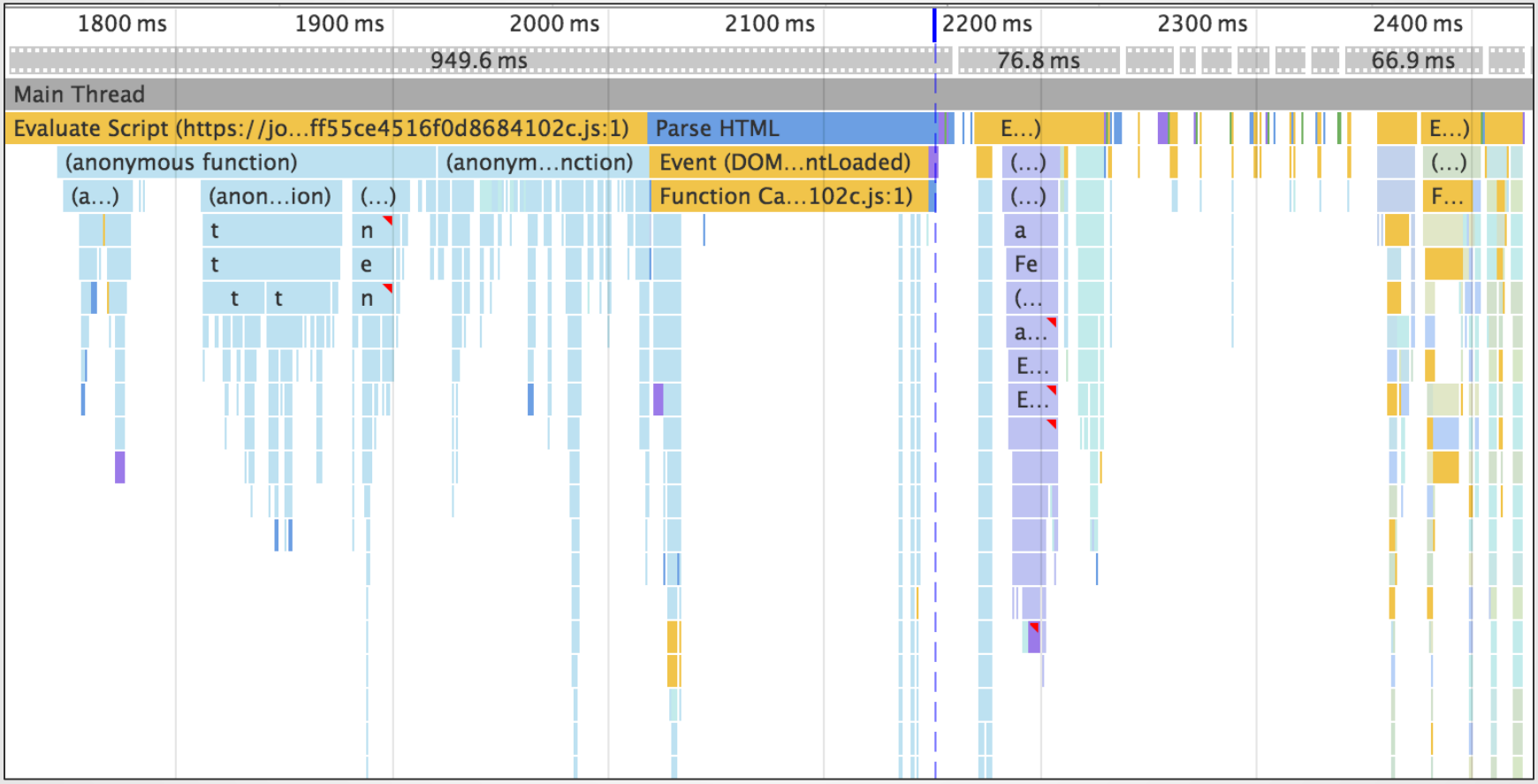


Visualization 8: Flame Graphs



Commonly used with CPU profilers. Also useful for tracers: off-CPU time, ...

Visualization 9: Flame Charts



Desirable Attributes

- Valuable
  - Methodologies provide ideas for purposeful metrics
- Documented
  - Tool tips, wikis
- Tested
- Real Time
- Dashboards
  - To support methodologies

Thank You!

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

References & Links:

- http://www.brendangregg.com/heatmaps.html
- http://www.brendangregg.com/flamegraphs.html
- http://www.slideshare.net/brendangregg/monitorama-2015-netflix-instance-analysis