LinuxCon North America 2016

Investigating System Performance for DevOps Using Kernel Tracing

EfficiOS

jeremie.galarneau@efficios.com
@LeGalarneau

Presenter


Jérémie Galarneau


EfficiOS Inc.

- Head of Support
- http://www.efficios.com


Maintainer of

- Babeltrace
- LTTng-Tools

Projects

- LTTng-tools
  - Session configuration daemon
  - Network streaming daemon
  - Command-line Interface
- Babeltrace
  - CLI trace reader
  - CTF-Writer library
  - CTF reader/writer Python bindings

Previous Talks

- Have mostly focused on LTTng **tracer** internals
  - Scalability,
  - Reducing per-event cost

- Today
  - How traces can be collected in the real world,
  - Tools we have developed to make them useful

LTTng: Two Tracers, Two Groups of Users

- User space (LTTng-UST)
  - Some open source projects are already instrumented (node.js, CoreCLR, QEMU, ...)
  - Manually instrument applications (requires knowledge of internals)
  - Often used as a flexible, high-speed logger
  - Understand interactions between user space and kernel space

- Kernel space (LTTng-modules)
  - Out-of-tree kernel modules (no kernel patching required, your distro has packages)
  - Leverages *mainline* kernel instrumentation
  - Only useful to kernel experts... or is it?

Do we need tools?

- Traces are **huge**
  - Recording very fine-grained events
    - `block, sched, irq, net, power, syscalls, ...`
  - Manual inspection works, but requires knowledge of application/kernel internals
  - Not always clear what you are looking for

Yes, we do!

- User space developers are building custom tools!
  - Know exactly what they are looking for
  - Modelling their application
  - Tracking internal resources (worker threads, memory pools, connections, users, etc.)

- Originally text-based tools
  - Piping hundreds of GBs of text traces through `grep, sed, perl, awk...`
  - Lots of one-off scripts being passed around
  - Unmanageable, hard to maintain, etc.
  - Break when Babeltrace's text output changes (new event fields)

Babeltrace Python Bindings

- Introduced Python bindings to read traces (2013)
  - Provide users with an easy way to “hack something together”
    - Debugging
    - Testing
  - Reasonably efficient under most scenarios
  - Scripts are maintained as internal tools

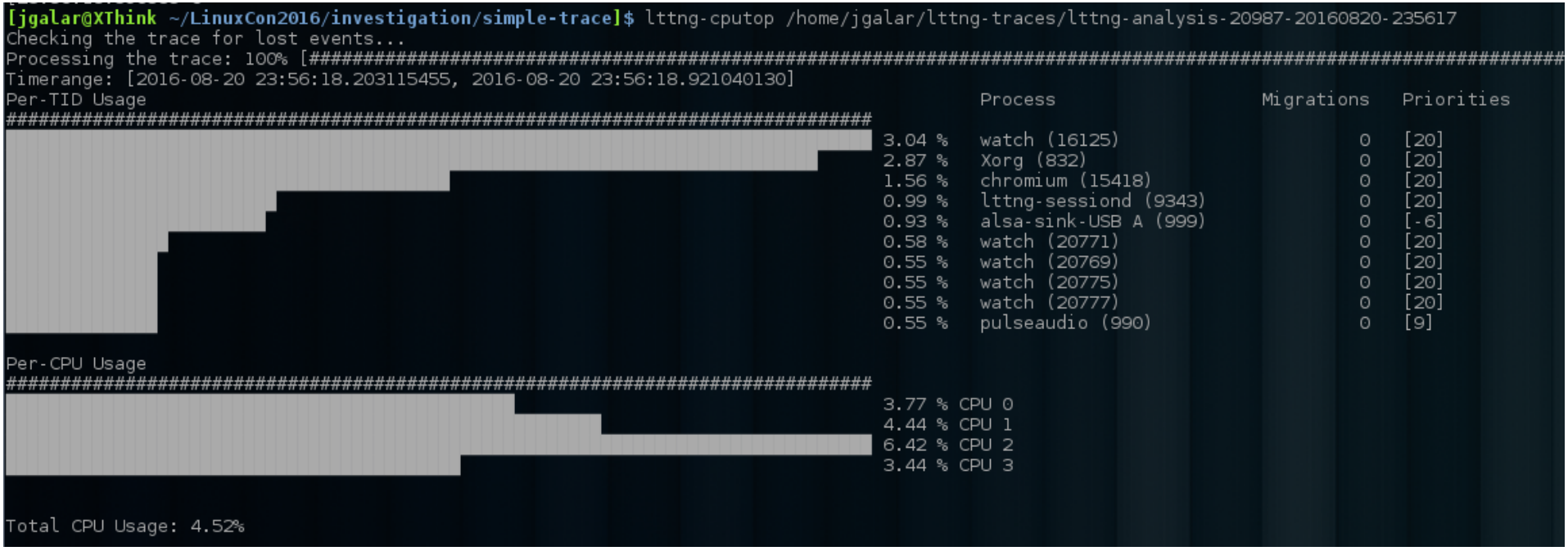
- Could we do the same for kernel space?

LTTng analyses

- Development started in early 2014
- Collection of utils
- Models some kernel subsystems to track their current state
  - Latency statistics and distributions (IO, Scheduling, IRQ)
  - System call statistics
  - IRQ handler duration
  - Top resource users

https://github.com/lttng/lttng-analyses

LTTng analyses



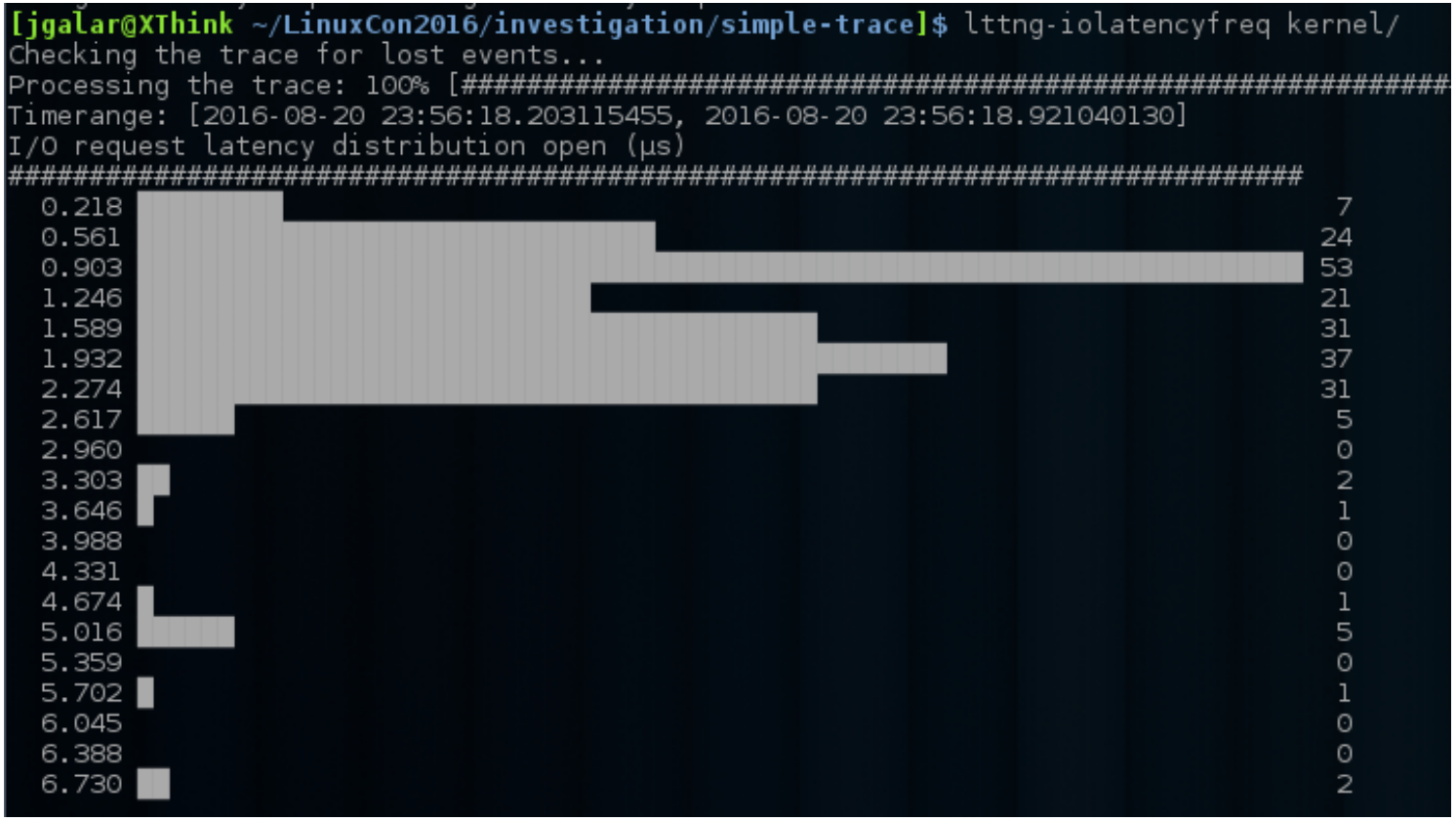
LTTng analyses

```
[jgalar@XThink ~/LinuxCon2016/investigation/simple-trace]$ lttng-iousagetop /home/jgalar/lttng-traces/lttng-analysis-20987-20160820-235617
Checking the trace for lost events...
Processing the trace: 100% [########################################################################################################################]
Timerange: [2016-08-20 23:56:18.203115455, 2016-08-20 23:56:18.921040130]
Per-process I/O Read                                                                 Process                 Disk       Net       Unknown
################################################################################
                                                                    113.62 KiB sh (20765)          113.62 KiB      0   B      0   B
                                                                    113.62 KiB sh (20767)          113.62 KiB      0   B      0   B
                                                                    113.62 KiB sh (20769)          113.62 KiB      0   B      0   B
                                                                    113.62 KiB sh (20771)          113.62 KiB      0   B      0   B
                                                                    113.62 KiB sh (20773)          113.62 KiB      0   B      0   B
                                                                    113.62 KiB sh (20777)          113.62 KiB      0   B      0   B
                                                                    113.62 KiB sh (20775)          113.62 KiB      0   B      0   B
                                                                     25.70 KiB lttng-sessiond (9296)     0   B      0   B  25.70 KiB
                                                                     23.77 KiB lttng (20778)        15.21 KiB   8.57 KiB      0   B
                                                                      5.25 KiB watch (16125)            0   B      0   B   5.25 KiB

Per-process I/O Write                                                                Process                 Disk       Net       Unknown
################################################################################
                                                                    25.70 KiB lttng (20778)            0   B  25.70 KiB      0   B
                                                                     8.59 KiB lttng-sessiond (9296)    0   B      0   B   8.59 KiB
                                                                     2.41 KiB Chrome_ChildIOT (15407)  0   B      0   B   2.41 KiB
                                                                     1.47 KiB xcompmgr (939)           0   B      0   B   1.47 KiB
                                                                      777   B terminator (15005)       0   B      0   B    777   B
                                                                      768   B sh (20765)               0   B      0   B    768   B
                                                                      768   B sh (20767)               0   B      0   B    768   B
                                                                      768   B sh (20769)               0   B      0   B    768   B
                                                                      768   B sh (20771)               0   B      0   B    768   B
                                                                      768   B sh (20773)               0   B      0   B    768   B

Per-file I/O Read                                                                    Path
################################################################################
```

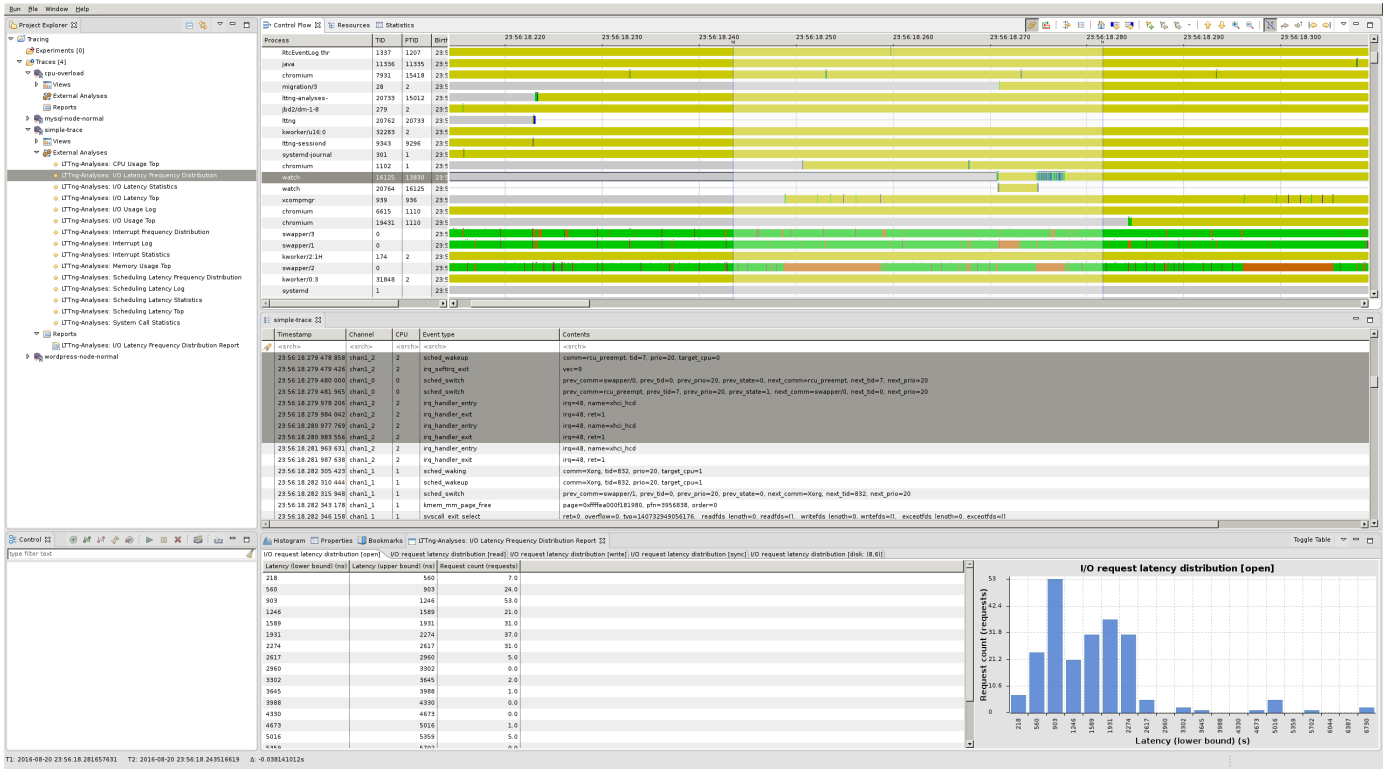
LTTng analyses

```
[jgalar@XThink ~/LinuxCon2016/investigation/simple-trace]$ lttng-syscallstats kernel/
Checking the trace for lost events...
Processing the trace: 100% [##################################################################################################################################]
Timerange: [2016-08-20 23:56:18.203115455, 2016-08-20 23:56:18.921040130]
Per-TID syscalls statistics (usec)
watch (16125, TID: 16125)                  Count          Min        Average          Max      Stdev  Return values
 - read                                    44107        0.138          0.797     4251.704     50.316  {'success': 44108}
 - rt_sigaction                               14        0.247          0.489        0.924      0.247  {'success': 15}
 - close                                      14         0.39           0.61        1.058      0.193  {'success': 15}
 - poll                                       14        0.538          1.613        3.075      1.036  {'success': 15}
 - pipe                                        7        5.466          6.542        7.635      0.885  {'success': 8}
 - wait4                                       7        3.255           3.83        5.179      0.642  {'success': 8}
 - fcntl                                       7        0.519          1.045        1.255       0.25  {'success': 8}
 - clone                                       7        64.16         68.005       74.077       3.14  {'success': 8}
 - newfstat                                    7        0.772          0.878        1.195      0.151  {'success': 8}
 - newstat                                     7        8.248          9.101       10.215       0.67  {'success': 8}
 - newuname                                    7        1.044           1.23        1.517      0.174  {'success': 8}
 - nanosleep                                   6    100096.694    100102.787   100107.064      4.424  {'success': 7}
Total:                                     44204
-------------------------------------------------------------------------------------------------------------------------
lttng-consumerd (?, TID: 9423)             Count          Min        Average          Max      Stdev  Return values
 - ioctl                                     325        0.177          1.234       55.117      3.355  {'EAGAIN': 2, 'success': 325}
 - splice                                    162        1.382          3.313       12.323      2.013  {'success': 163}
 - sync_file_range                           162        6.949           30.1      512.894     42.733  {'success': 163}
 - fadvise64                                  81        1.889          2.676       24.711      2.783  {'success': 82}
Total:                                       730
-------------------------------------------------------------------------------------------------------------------------
threaded-ml (?, TID: 7926)                 Count          Min        Average          Max      Stdev  Return values
 - read                                      162         0.24          1.971       17.789      2.599  {'EAGAIN': 51, 'success': 113}
 - write                                      97        0.456          2.508        9.372      1.444  {'success': 98}
 - fcntl                                      93        0.178          0.589        1.799      0.433  {'success': 94}
 - poll                                       49      166.732       14489.45    20299.653   5428.461  {'success': 50}
 - select                                     31        3.987          4.974         9.53      0.972  {'success': 32}
 - ioctl                                      31        0.781            1.1        3.642       0.49  {'success': 32}
Total:                                       463
```

LTTng analyses



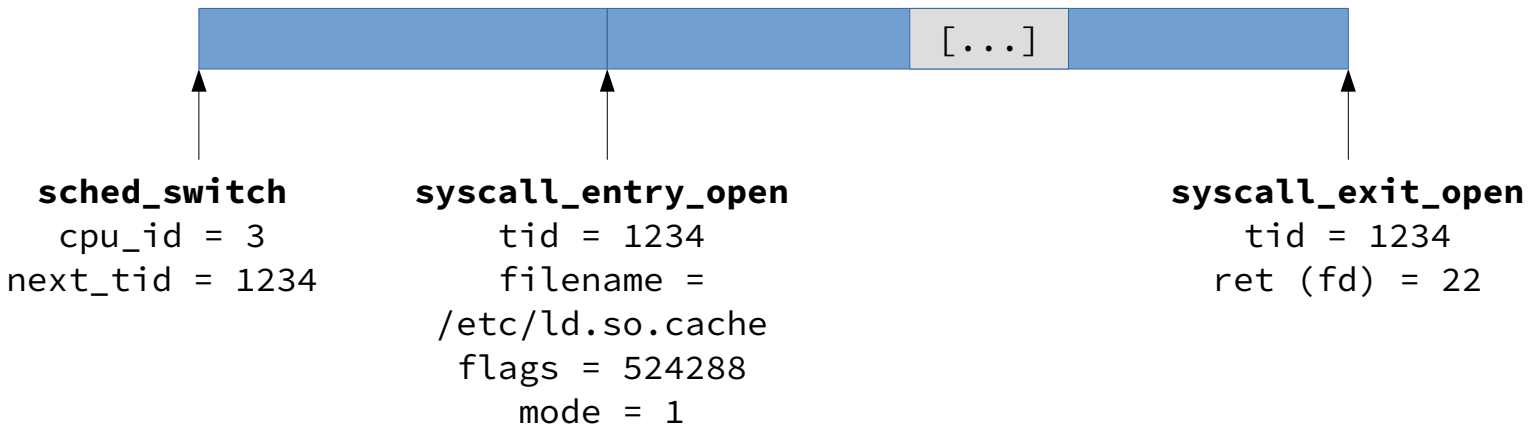
LTTng analyses - Trace Compass Integration



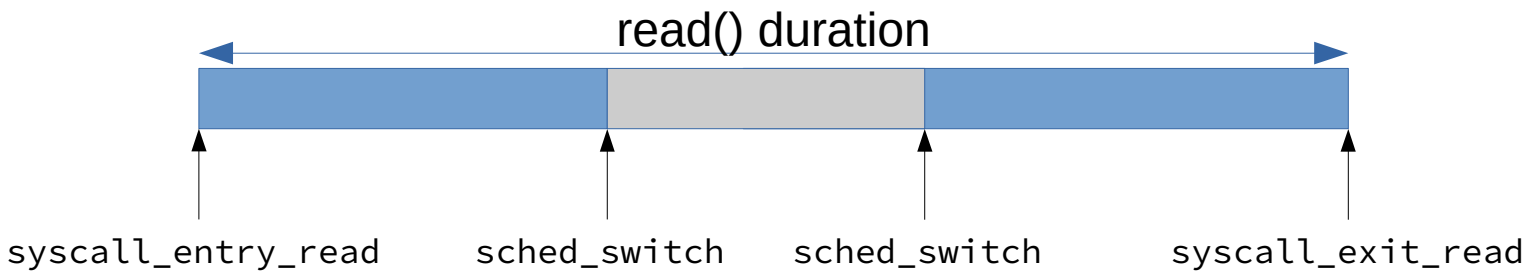
- Invoke custom analyses
- LAMI 1.0
  - Open Specification
  - JSON based

TRACE COMPASS

Principles – Tracking Resources



Principles – Tracking Latencies



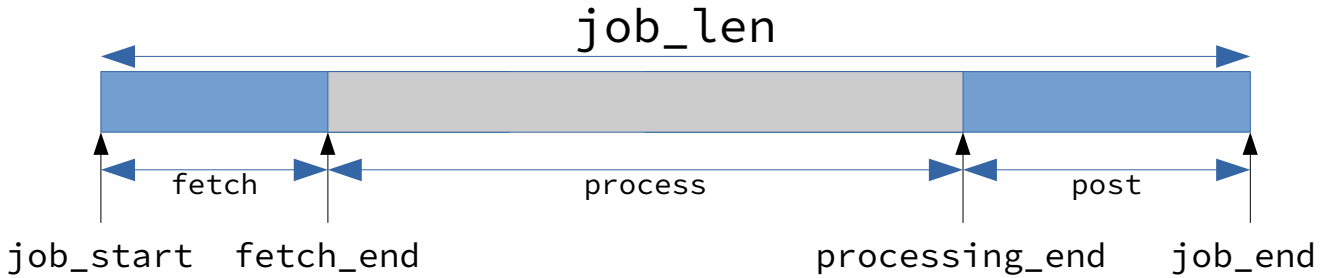
Custom Periods

```
$ lttng-periodlog
    --period [ NAME [ (PARENT) ] ] : BEGIN_EXPR [ : END_EXPR ]
    --period-aggregate PERIODS
    --period-aggregate-by PERIOD
    --period-group-by FIELD
    [...]
```

- Example for `read()` durations

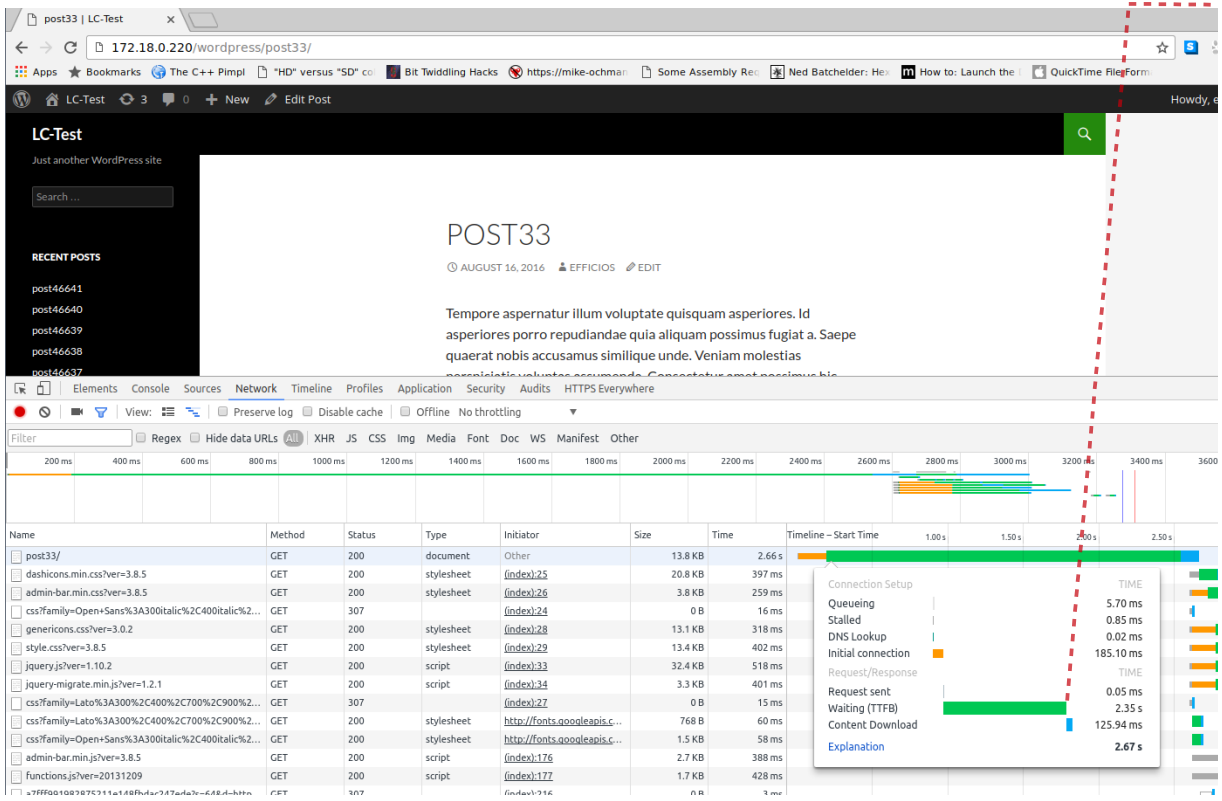
```
$ lttng-periodlog --top --freq --stats
    --period ‘read_duration : $evt.$name == “syscall_entry_read” :
                              $evt.$name == “syscall_exit_read”’
    --period-captures ‘read_duration : read_size = $evt.count’
```

Custom Periods




```
$ lttng-periodlog --period ‘job_duration :  $evt.$name == “job_start” :
                                            $evt.$name == “job_end”
                  --period ‘fetch(job_len): $evt.$name == “job_start” :
                                            $evt.$name == “fetch_end”’
                  [...]
```

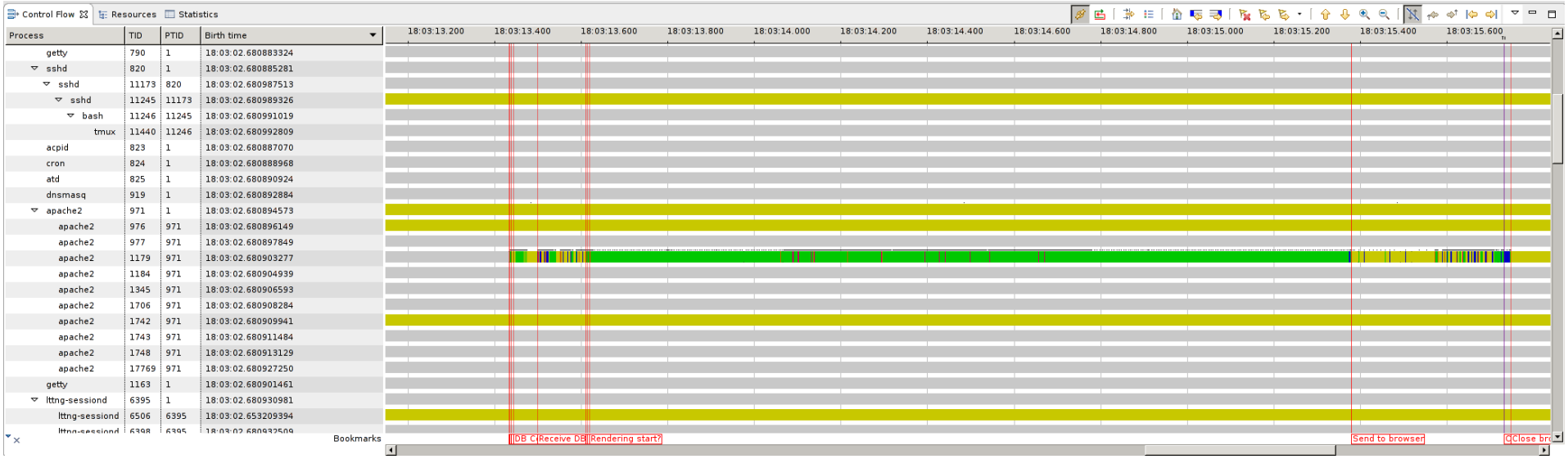
Tackling Some “Real” Problems



- Long Time to First Byte
- Demos on
  - WordPress
  - MariaDB/MySQL
  - Apache httpd
- The stack doesn’t matter... as long as it runs on Linux!

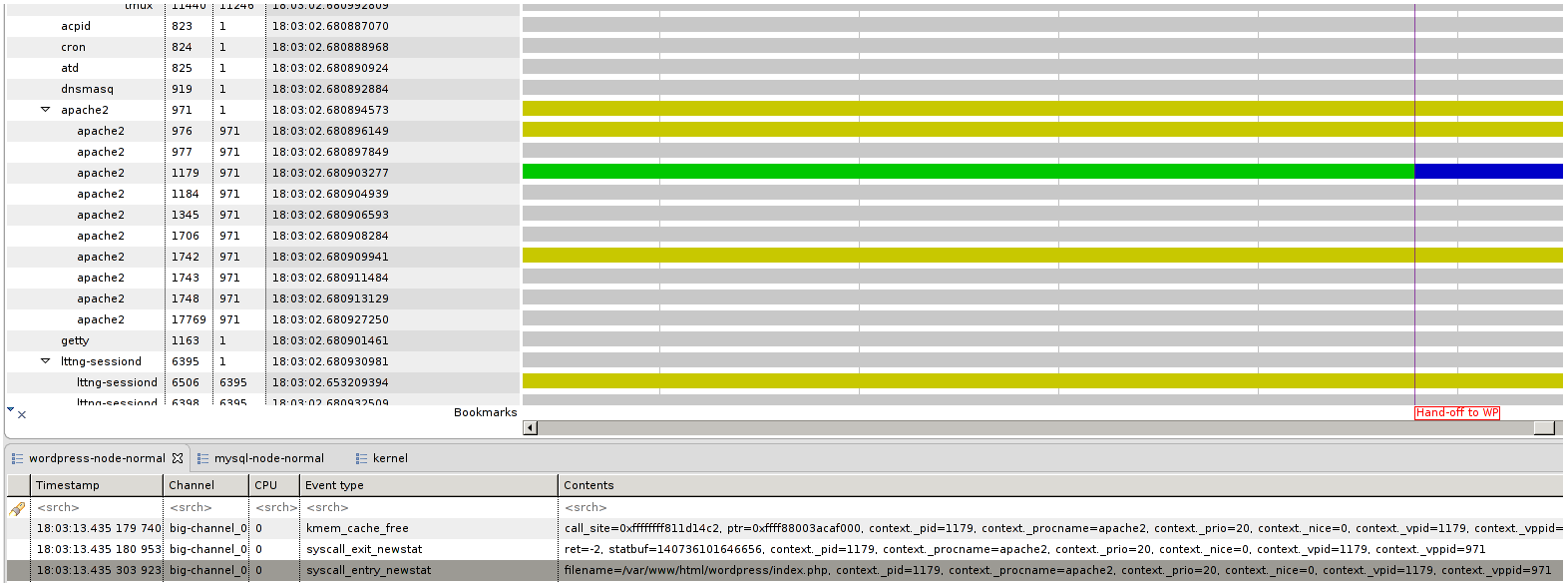
Anatomy of a Request

- request
  - `fd = accept4(...)`
  - `close(fd)`



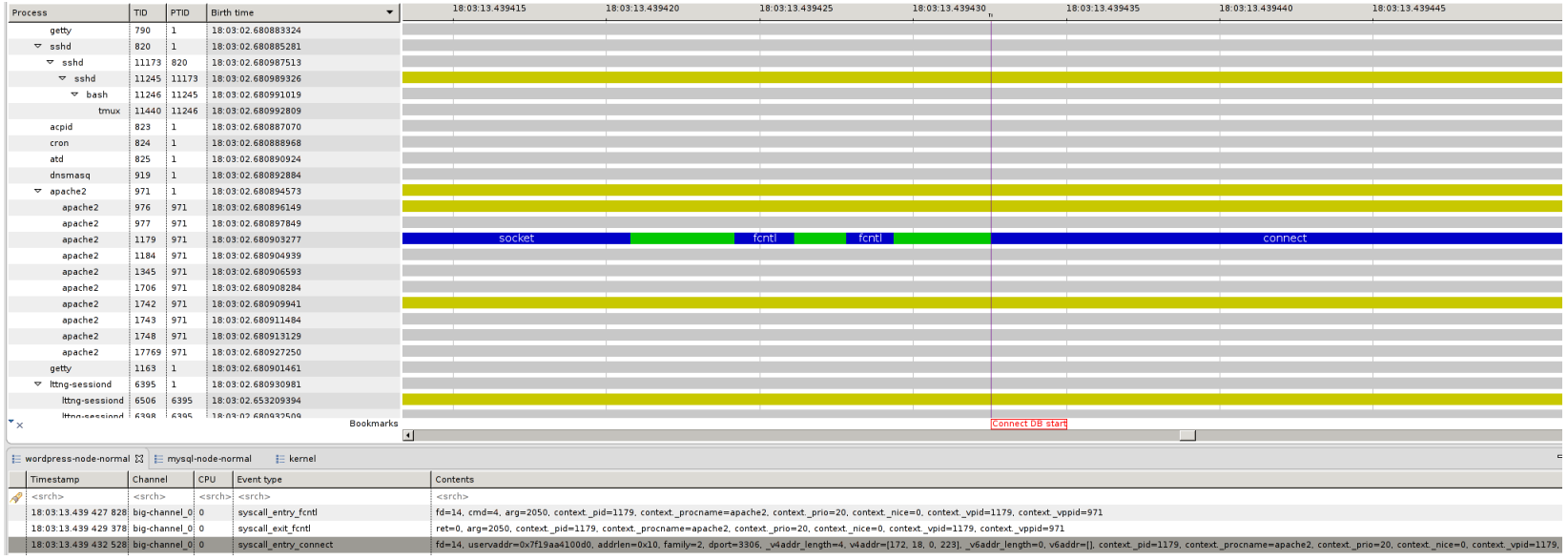
Anatomy of a Request

- url_rewrite
  - `fd = accept4(...)`
  - `newstat(...)`



Anatomy of a Request

- db_queries
  - `connect(...)`
  - `newstat(...)` -> PHP resumes execution

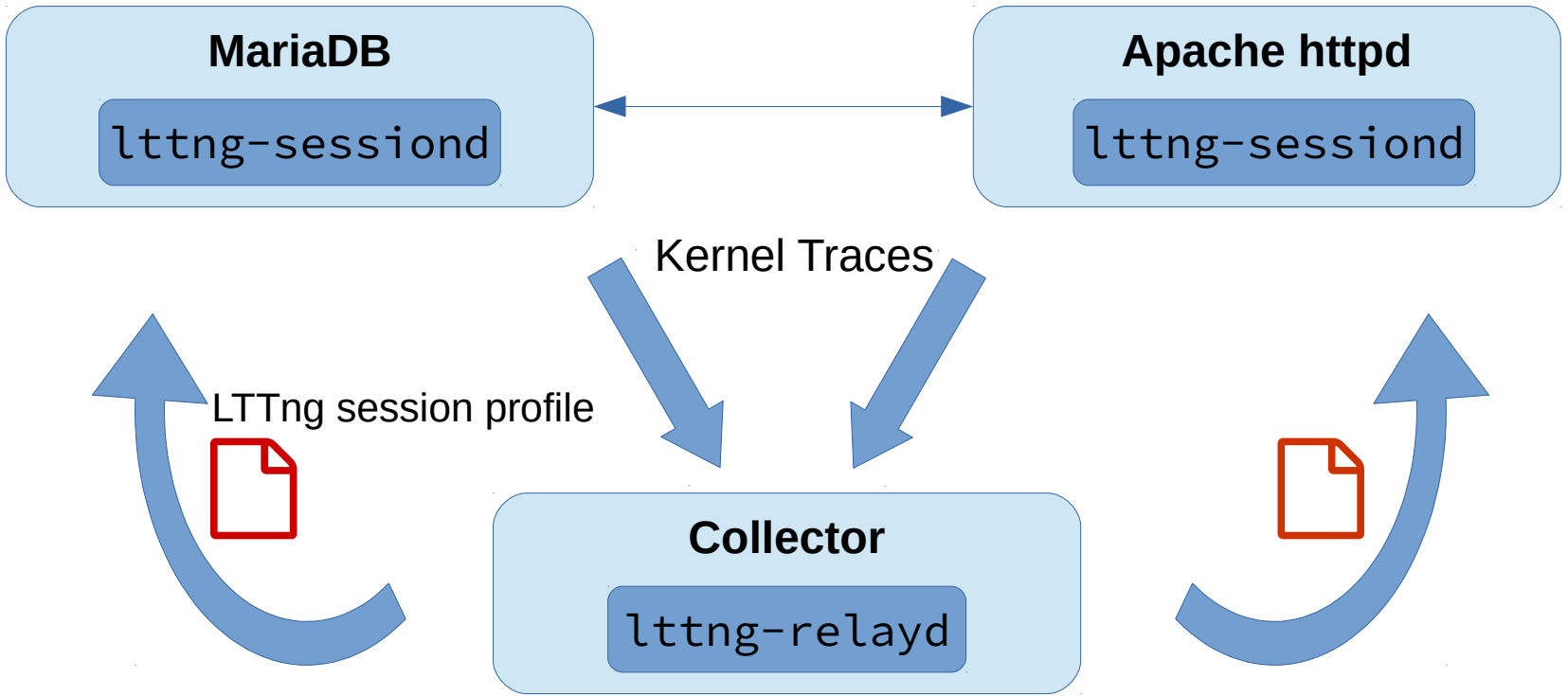


Anatomy of a Request

- Coarse breakdown of periods
  - The goal is not to benchmark
  - Want to know in which “phase” the problem happens

- `request (parent)`
  - `url_rewrite`
  - `db_queries`
  - `rendering`
  - `send_to_client`

Deploying LTTng



Case #1

- Almost all requests are unacceptably slow
- No connectivity problems between clients and servers

- Capture a trace for a couple of seconds on both nodes
  - Stream to my machine for storage and investigation
    - No need to plan for storage in production
    - Local buffering can be used
  - No push-back against the kernel if the network bandwidth is too low

Case #1

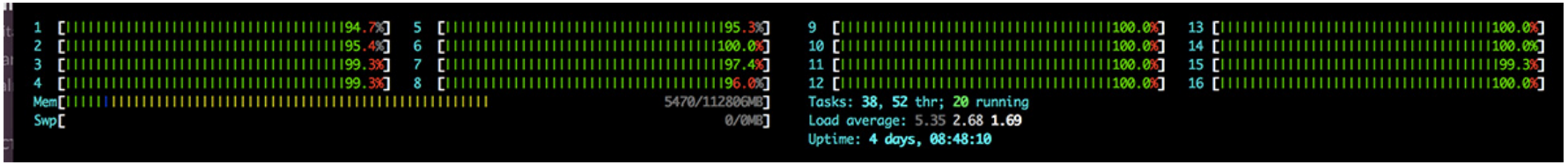
- Extract a log of all requests from the WordPress node's trace

```
$ lttng-periodlog --top --stats --freq
    --period 'request :
      $evt.$name == "syscall_exit_accept4" &&
      $evt.procname == "apache2" :
      $evt.pid == $begin.$evt.pid &&
      $evt.$name == "syscall_entry_close" &&
      $evt.fd == $begin.$evt.ret'
```

Not a tracers' forte

- Collecting long traces to compute CPU usage is inefficient

- Most monitoring tools would have found this right away



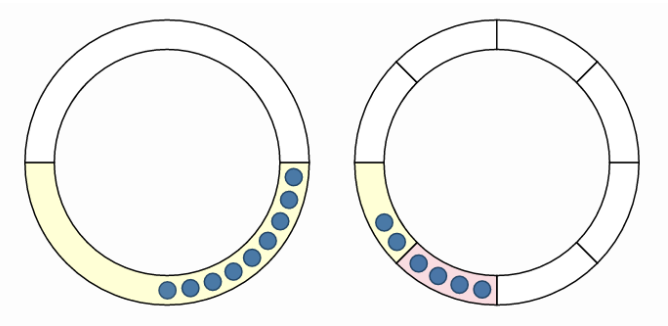
What Tracing is Good For

- Most monitoring tools focus on trends and averages
  - High-level view of resource usage
  - Error rates

- Tracers are great to understand
  - Usage problems
    - Lots of small `read()`, contention on a shared system resource, etc.
    - Race conditions
  - Spurious issues
    - One slow request out of many million

Snapshot Mode

- Also known as “flight-recorder” mode
  - Trace to in-memory buffers
  - Configurable buffer sizes
  - Low overhead



- Save only when an “`lttng snapshot record`” is issued
  - Save to disk
  - Stream through the network to the collector node

mod_trigger

- Proof-of-concept Apache httpd module inspired by mod_logslow
- Triggers a snapshot on all machines whenever a request takes longer than a set threshold

- Limitations
  - ssh into target machines can take too long and cause us to lose the information we were looking for (overwritten buffers)
  - No rate limiting, which can make things worse...

https://github.com/jgalar/mod_trigger

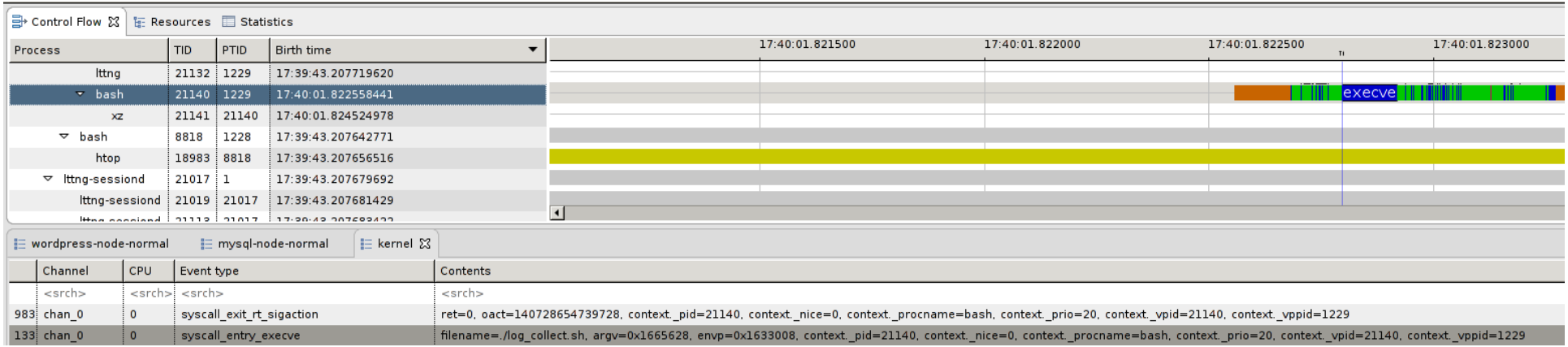
Case #2, #3 and #4

- Loaded the LTTng session profile on both machines
- Launch `lttng-relayd` on my machine
  - Runs on Linux, BSD and OS X

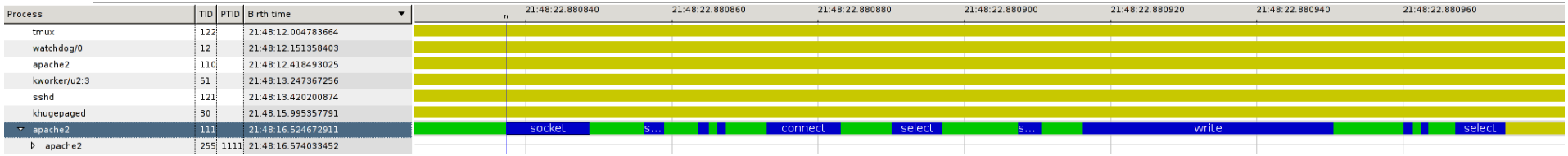
- Wait until a request fails to meet our QOS-levels
  - 1 second per request

Case #2

- xz is clearly taking-up a lot of CPU time
  - Trace Compass allows us to see which process launched it
  - Logs are being compressed and collected



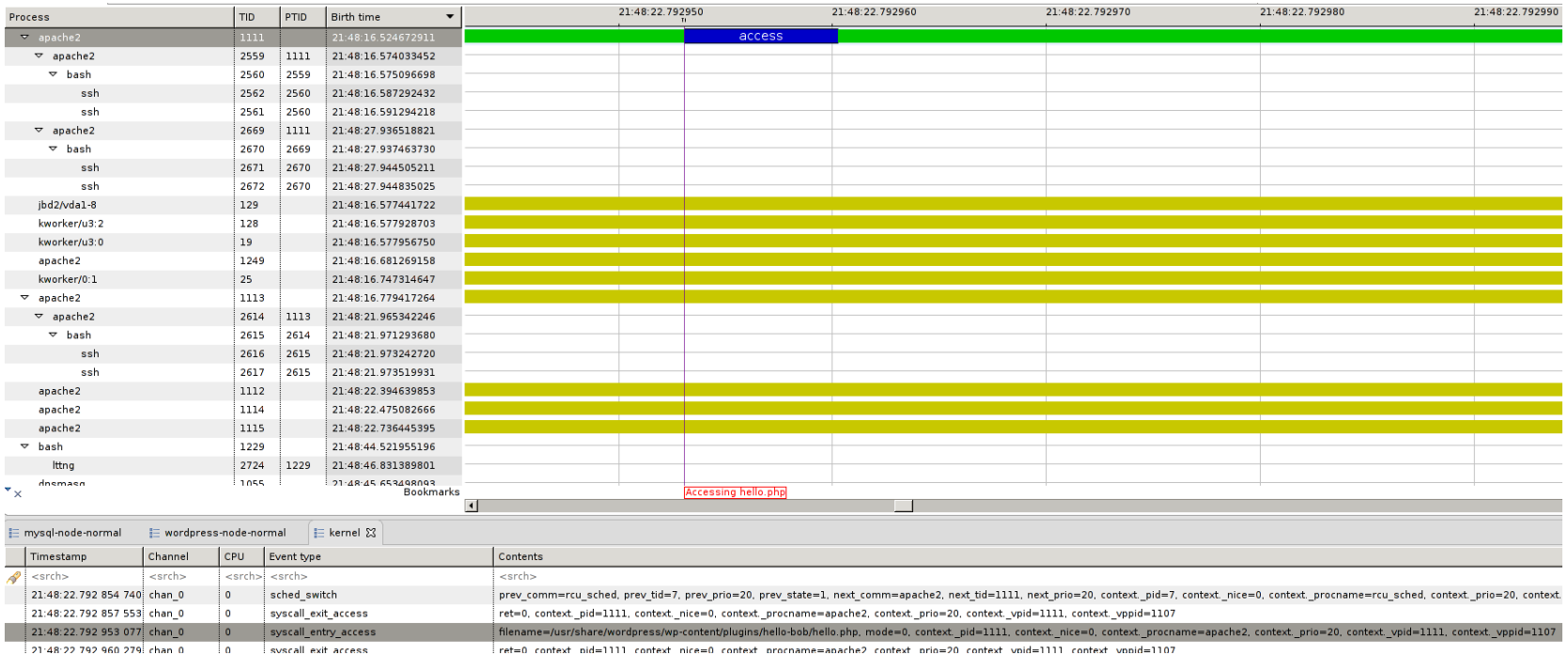
Case #3



- Long call to `select(...)`
- UDP socket created
- `connect()` to `8.8.6.6`

Case #3

- access(“wp-content/plugins/hello.php”)

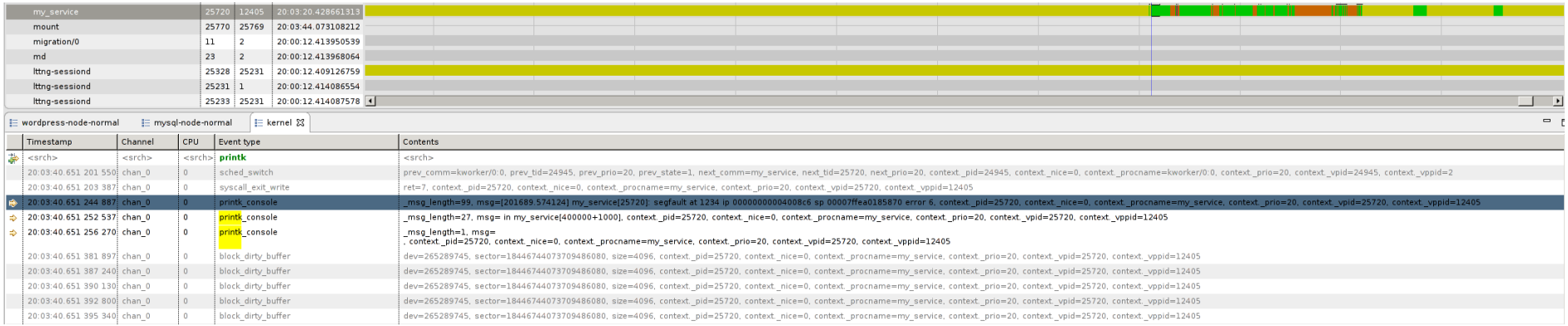


Case #3

- A WordPress plug-in is making DNS requests to 8.8.6.6 ... which doesn't exist!

```php
require_once 'Net/DNS2.php';

$nameservers = array(
    '8.8.4.4',
    '8.8.6.6',
    '8.8.8.8',
    '208.67.222.222',
    '208.67.220.220',
);

$resolver = new Net_DNS2_Resolver(array(
    'nameservers' => $nameservers,
    'ns_random' => true,
    'timeout' => 5,
));

try {
    $resp = $resolver->query("youtube.com.", 'A');
    $youtube_addr = $resp->answer[0]->address;
} catch(Net_DNS2_Exception $e) {
```

Case #4

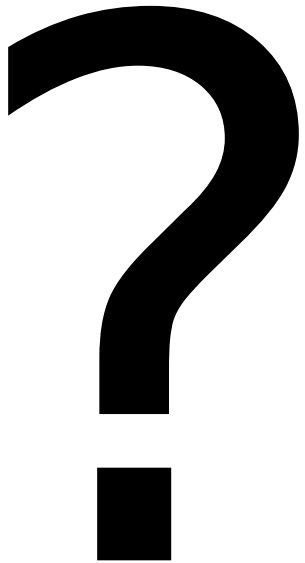


- Limitations
  - IO at the block layer is hard to track
  - We don't see the core dump file being created (no `open()`)
  - `block_dirty_buffer` happens in the context of crashing `my_service`
  - `block_rq_issue` and `block_rq_complete` happen in the context of `kworker/0:2`

Conclusion

- Tools make traces useful in the real world
  - Manual inspection works in a very narrow window
  - You don’t need to be a kernel developer to make sense of them
- *Almost* automated...
  - Use snapshots to collect on failure
  - lttng-analyses + your scripts can be run on the traces to start the investigation
- See https://github.com/jgalar/LinuxConNA2016

Questions ?



- lttng.org
- lttng-dev@lists.lttng.org
- @lttng_project

EfficiOS

- www.efficios.com
- @efficios