Large-scale performance monitoring framework for cloud monitoring

Live Trace Reading and Processing

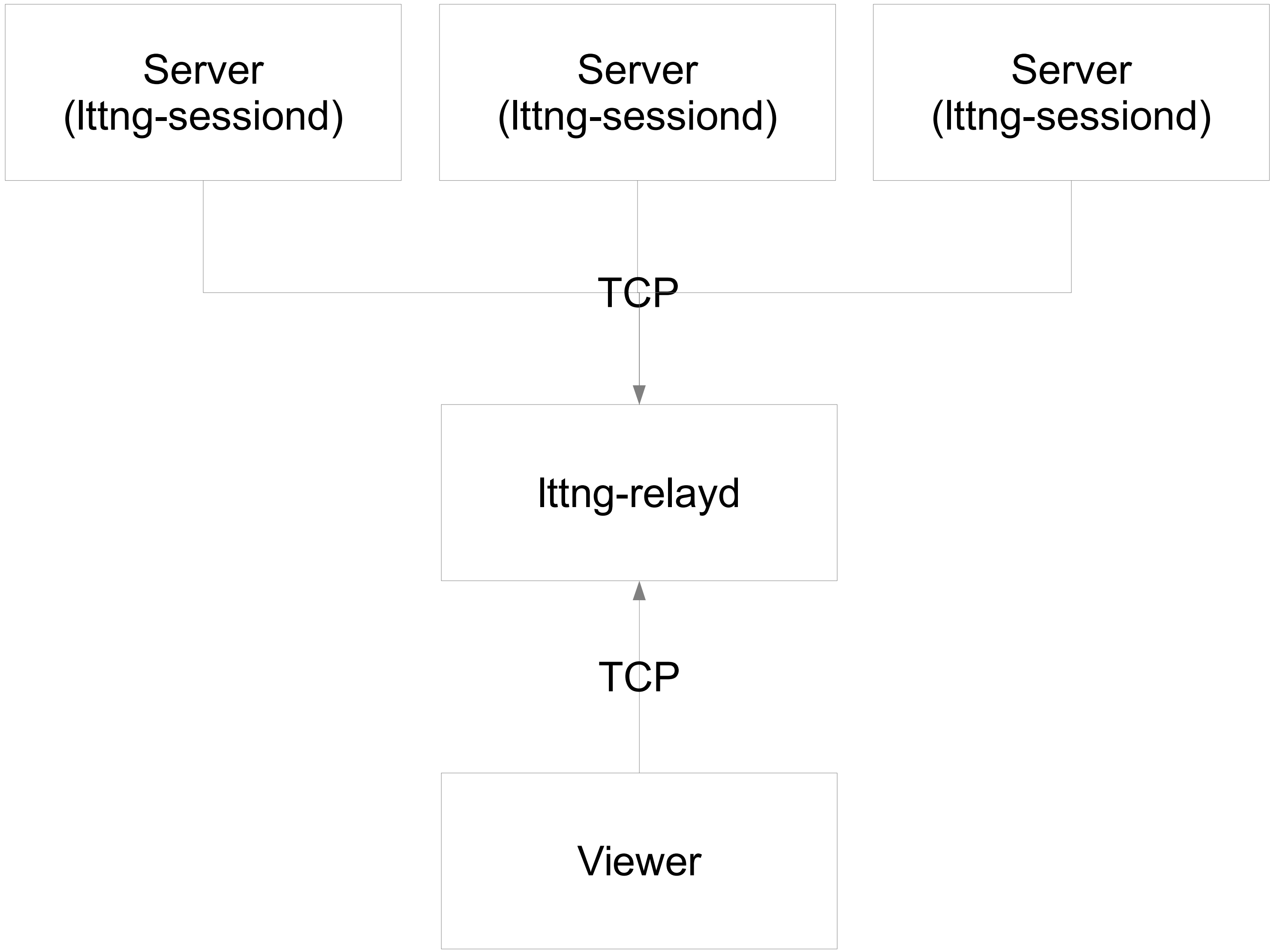
Julien Desfossez
Michel Dagenais

May 2014
École Polytechnique de Montreal

Live Trace Reading

- Read the trace while it is being recorded
- Local or remote session
- Configurable flush period (live-timer)
- Merged into LTTng 2.4.0
- Supported by Babeltrace 1.2 and LTTngTop
- Work in progress in TMF

Infrastructure integration



Live streaming session

On the server to trace :

```
$ lttng create --live 2000000 -U net://10.0.0.1
$ lttng enable-event -k sched_switch
$ lttng enable-event -k --syscall -a
$ lttng start
```

On the receiving server (10.0.0.1) :

```
$ lttng-relayd -d
```

On the viewer machine :

```
$ lttngtop -r 10.0.0.1
```

Or

```
$ babeltrace -i lttng-live net://10.0.0.1
```

What has been done since the last progress report meeting

- Bugfixing and release of LTTng 2.4.1
- Graphite integration tests
- Stress/performance testing
- Started Zipkin/Tomograph integration to trace OpenStack (Python)
- Working with an GSoC intern on Babeltrace to Zipkin
- Sysadmin-oriented analyses prototypes (Python)
- Writing the paper about live tracing

Graphite Integration

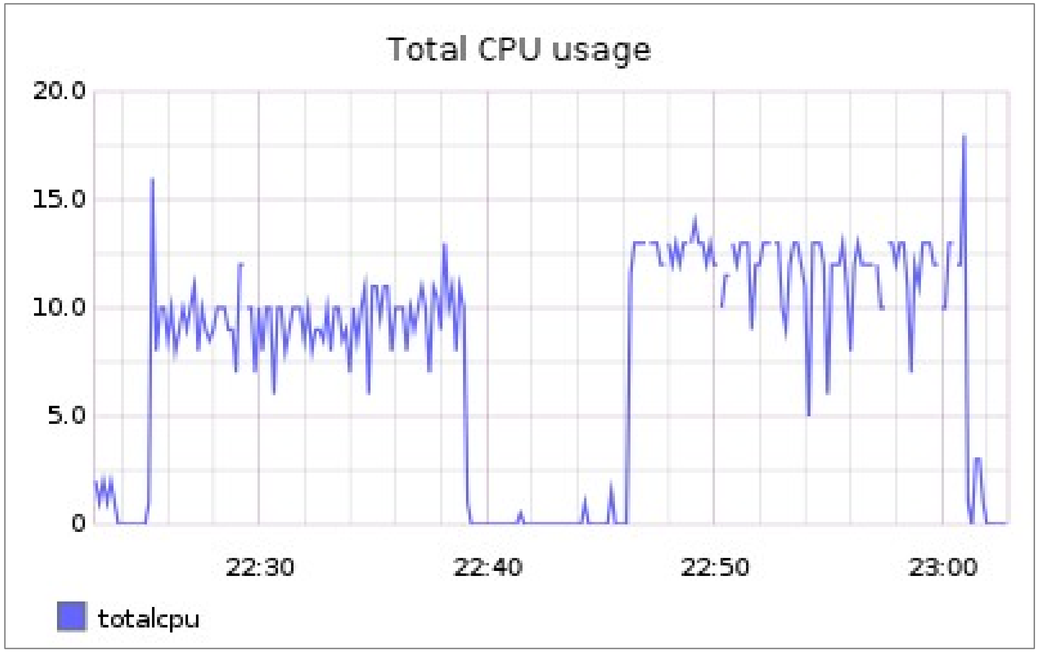
Stress-testing setup

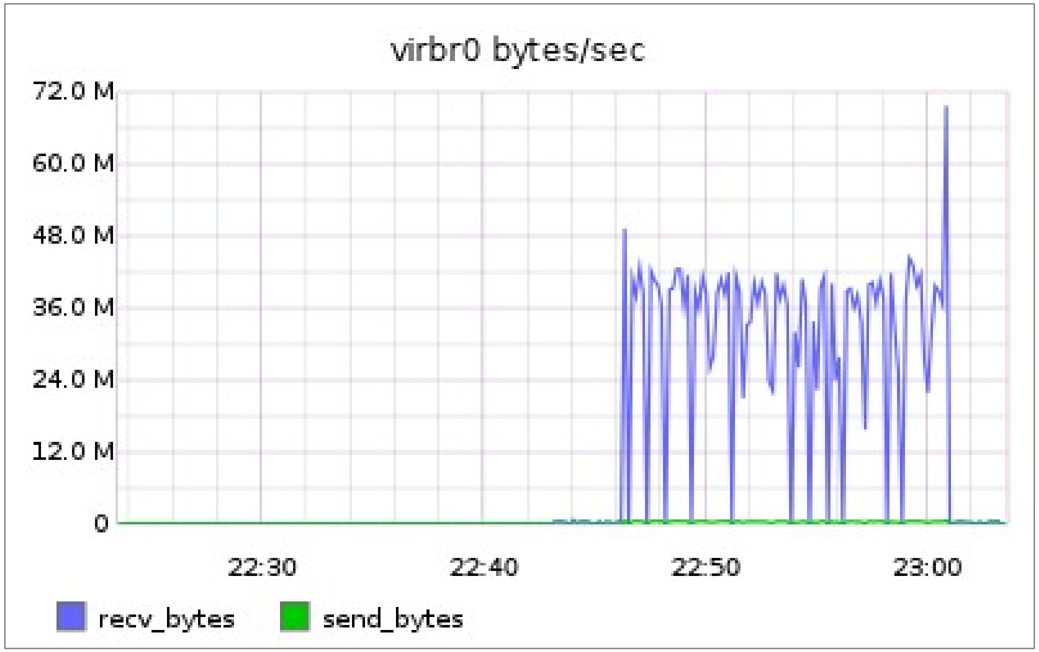
- 48 AMD Opteron(tm) Processor 6348
- 512GB RAM
- 4x1TB SSD (1 for the OS, 1 for the VMs, 1 for the traces)
- Ubuntu 14.04 LTS
- Linux Kernel 3.13.0-16
- LTTng Tools 2.4+ (git HEAD on March 10th)

Stress-testing

- 100 Ubuntu 12.04 VMs with 1GB RAM and 1 vCPU
- Streaming their traces to the host lttng-relayd with the live-timer of 5 seconds
- Tracing syscalls + sched_switch
- Running Sysbench OLTP (MySQL stress test)
- Measure overall impact on the system

100 Sysbench





Python analyses

demo

Next steps

- Finish writing the paper
- Work on the architecture to process traces and extract metrics from large group of machines
  - Studying the large-scale infrastructures monitoring systems
  - Studying HTTP analytics on large-scale web infrastructures
  - Look at Facebook Scribe and integration with Hadoop HDFS
  - Continue prototyping with the Python libraries

Install it

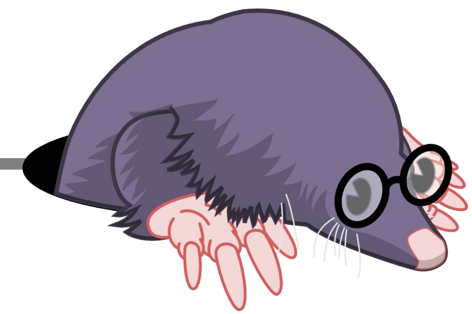
- Packages for your distro (`lttng-modules`, `lttng-ust`, `lttng-tools`, `userspace-rcu`, `babeltrace`)
- For Ubuntu : PPA for daily build (`lttngtop`)
- Or from the source, see `http://git.lttng.org`

LTTng 2.5 features

- Save/Restore sessions
  - lttng save
  - lttng restore
- Configuration file (lttng.conf)
  - System-wide : /etc/lttng/lttng.conf
  - User-specific : $HOME/.lttng/lttng.conf
  - Run-time
- Perf UST
- User-defined modules on lttng-sessiond startup
- lttng --version with git commit id

Questions ?

Virtual machine CPU monitoring with Kernel Tracing



Mohamad Gebai
Michel Dagenais

15 May, 2014
École Polytechnique de Montreal

Content

- General objectives
- Current approaches
- Kernel tracing
- Trace synchronization
- Virtual Machine Analysis
- Execution flow recovery

General objectives

- Getting the state of a virtual machine at a certain point in time
- Quantifying the overhead added by virtualization
- Track the execution of processes inside a VM
- Aggregate information from host and guests
- Monitoring multiple VMs on a single host OS
- Finding performance setbacks due to resource sharing among VMs

Current approaches

- Top
- Steal time: percentage of vCPU preemption for the last second

```
top - 11:39:00 up 29 min,  1 user,  load average: 0.60, 0.16, 0.09
Tasks:  73 total,   3 running,  70 sleeping,   0 stopped,   0 zombie
%Cpu(s): 63.1 us,  0.2 sy,  0.0 ni,  8.8 id,  0.0 wa,  0.0 hi,  0.0 si, 28.0 st
KiB Mem:   1964140 total,   149216 used,  1814924 free,     8736 buffers
KiB Swap:   604156 total,        0 used,   604156 free,    97540 cached
```

- Does not reflect the effective load on the host
  - 0% for idle VMs even if the physical CPU is busy
- Not enough information

Current approaches

- Perf kvm
- Information about VM exits, performance counters

```
14:13:28.000181

Analyze events for all VMs, all VCPUs:

             VM-EXIT    Samples  Samples%     Time%   Min Time   Max Time         Avg time

      IO_INSTRUCTION        145    70.05%     0.26%        2us     1478us      14.04us ( +-  72.47% )
        EPT_MISCONFIG         25    12.08%     0.01%        1us        6us       3.79us ( +-  10.03% )
         APIC_ACCESS         21    10.14%     0.02%        2us       13us       6.12us ( +-  12.36% )
                 HLT          8     3.86%    99.71%     2420us   248023us   99141.25us ( +-  28.92% )
              VMCALL          5     2.42%     0.00%        0us        1us       1.20us ( +-  20.79% )
       EXCEPTION_NMI          2     0.97%     0.00%        0us        1us       1.31us ( +-  25.97% )
  EXTERNAL_INTERRUPT          1     0.48%     0.00%       32us       32us      32.37us ( +-   0.00% )

Total Samples:207, Total events handled time:795429.44us.
```

- No information from inside the VM
- No information about VM interactions

Kernel tracing

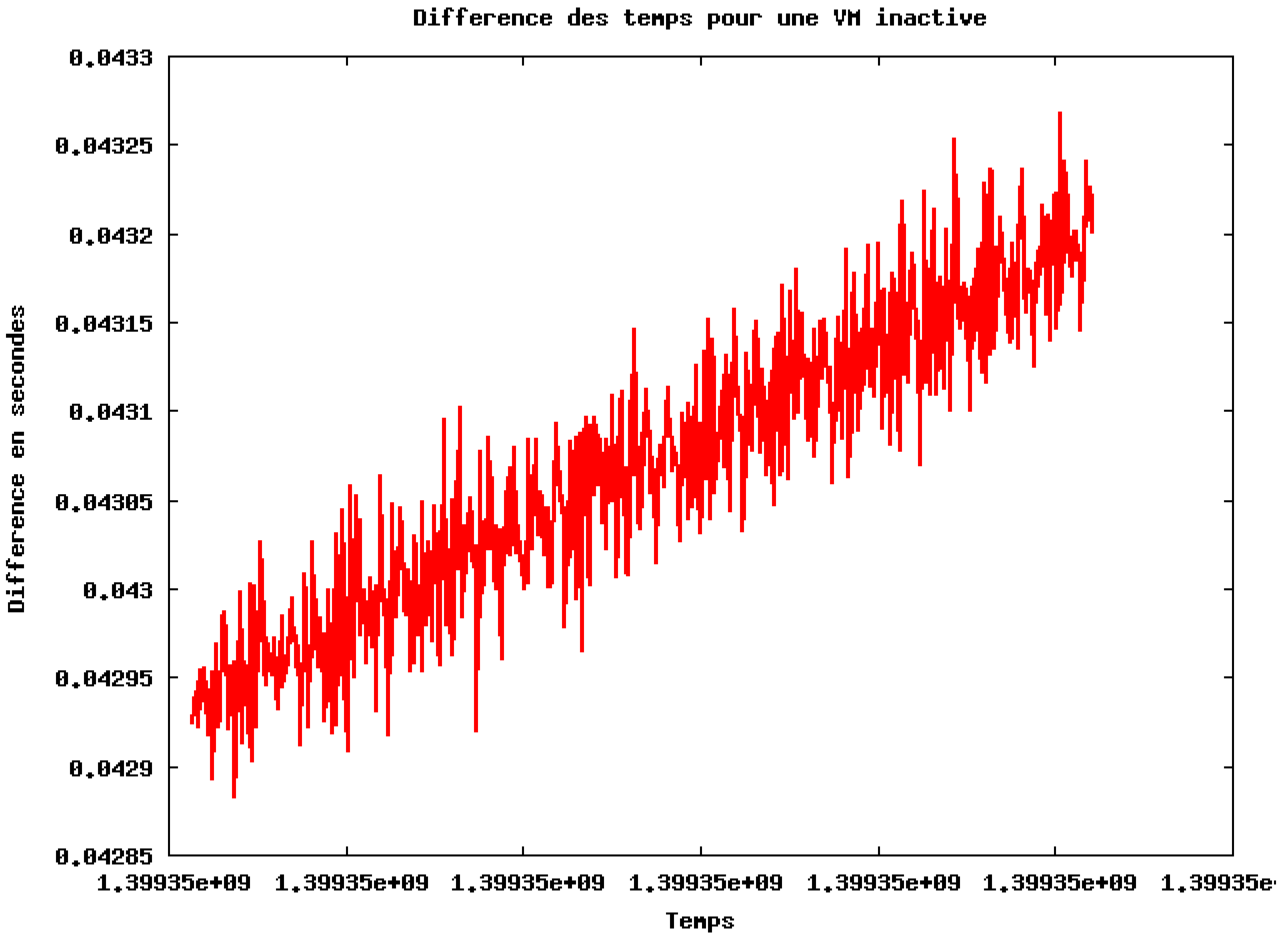
- Trace scheduling events
  - `sched_switch` for context switches
  - `sched_migrate_task` for thread migration between CPUs (optional)
  - `sched_process_fork`, `sched_process_exit`
- Trace VMENTRY and VMEXIT on the hypervisor (hardware virtualization)
  - `kvm_entry`
  - `kvm_exit`

Tracing virtual machines

- Each VM is a process
- Each vCPU is 1 thread
  - Per-thread state can be rebuilt
- A vCPU can be in VMX root mode or VMX non-root mode
- A vCPU can be preempted on the host
- The VM can't know when it is preempted or in VMX root mode
- Processes in the VM seem to take more time
- Trace host and guests simultaneously

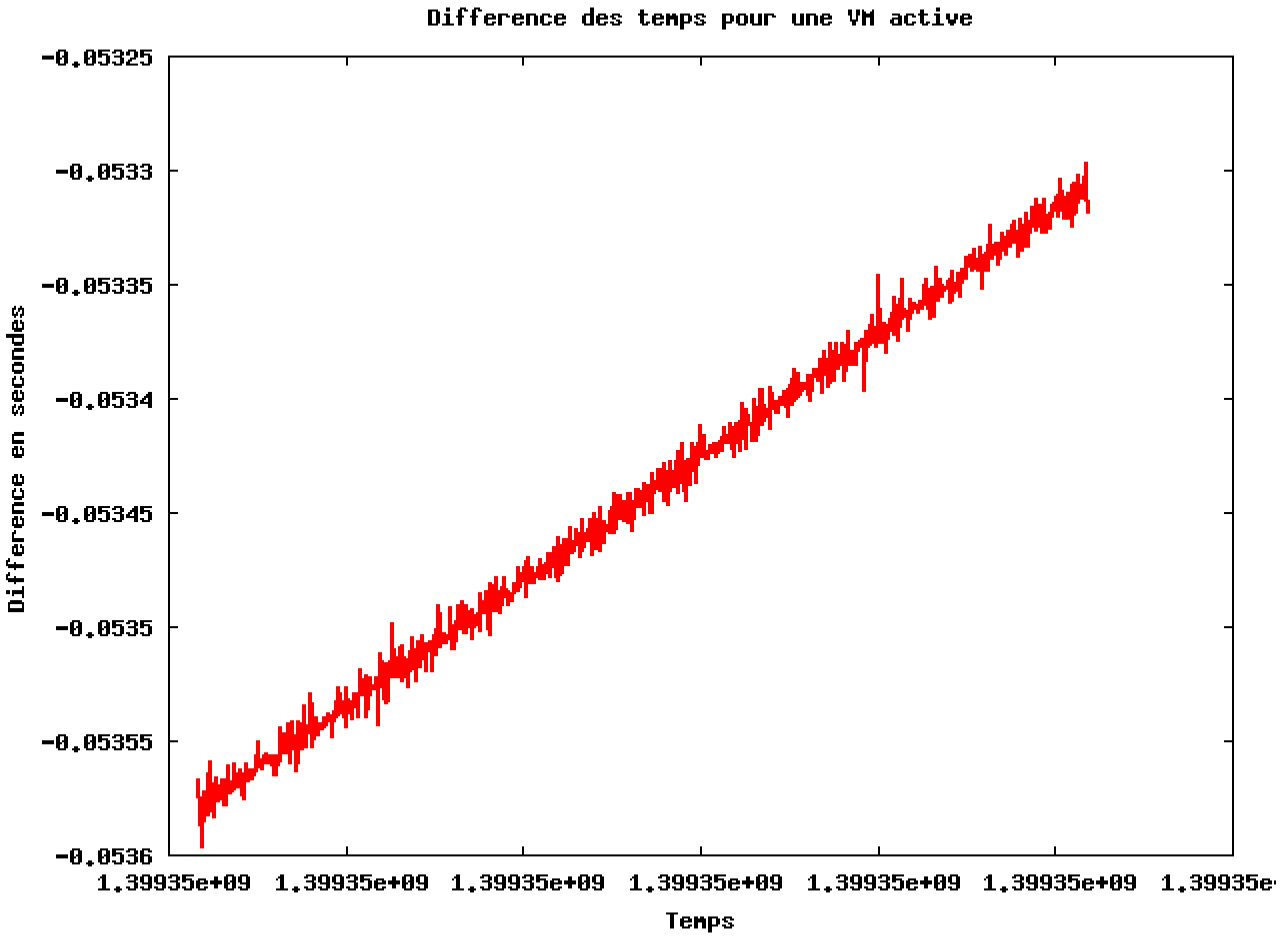
Trace synchronization

- Time difference between host and an idle VM



Trace synchronization

- Time difference between host and an active VM

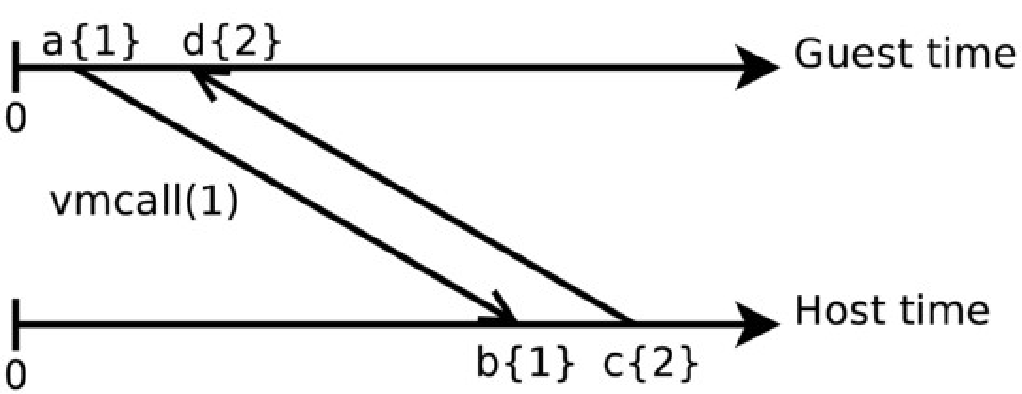


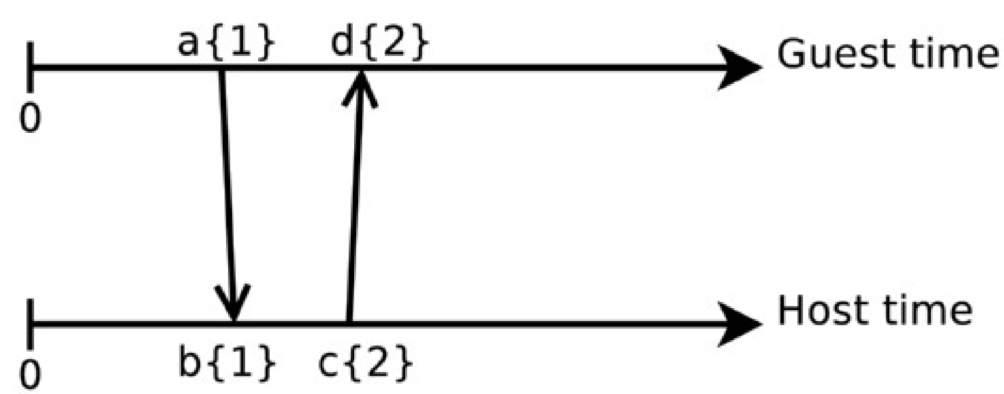
Trace synchronization

- Based on the fully incremental convex hull synchronization algorithm
- 1-to-1 relation required between events from guest and host
- Tracepoint is added to the guest kernel
- Executed on the system timer interrupt softirq
- Triggers a hypercall which is traced on the host
- Resistant to vCPU migrations and time drifts

Trace synchronization

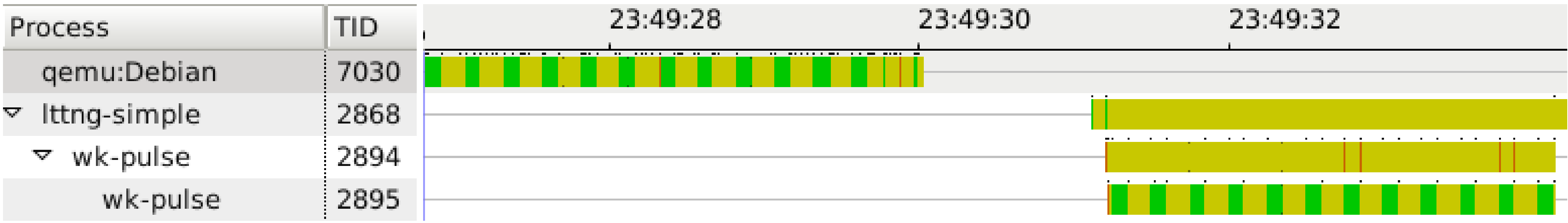
- Kernel module added to LTTng as an addon
- In the guest:
  - Trigger a hypercall (event `a`)
- On the host:
  - Acknowledge the hypercall (event `b`)
  - Give control back to the guest (event `c`)
- In the guest:
  - Acknowledge the control (event `d`)



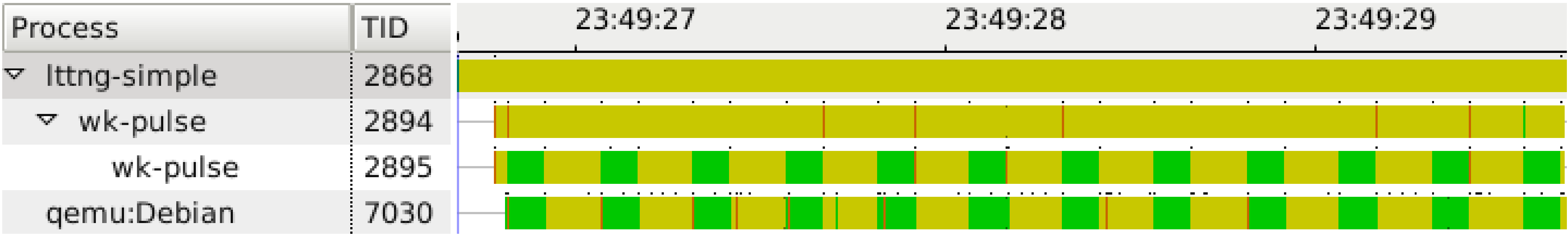


Trace synchronization

- Host and guest threads, as seen before..

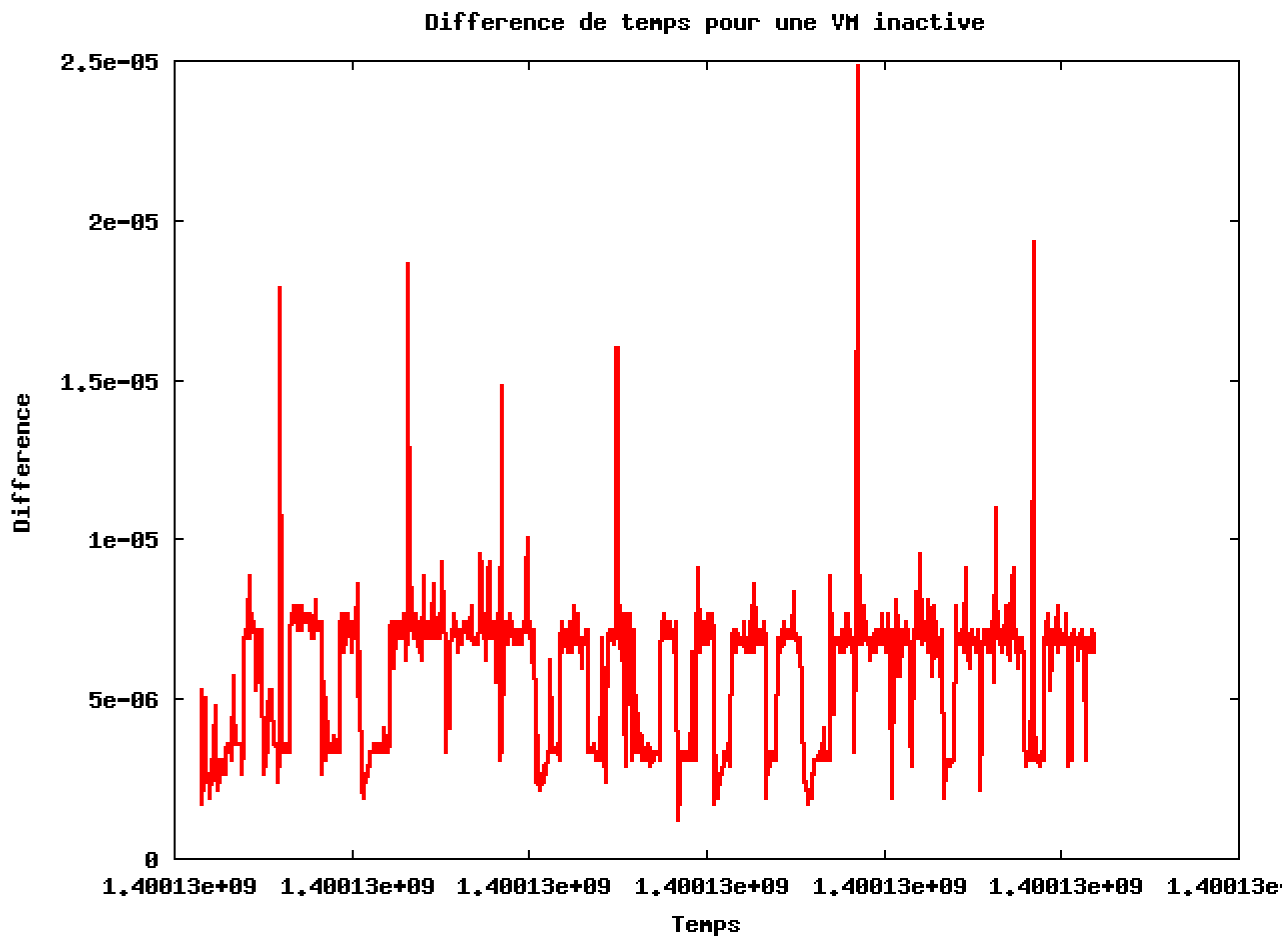


- ..and after synchronization



Trace synchronization

- Time difference between host and VM after synchronization



TMF Virtual Machine View

- Shows the state of each vCPU of a VM
- Aggregation of traces from the host and the guests

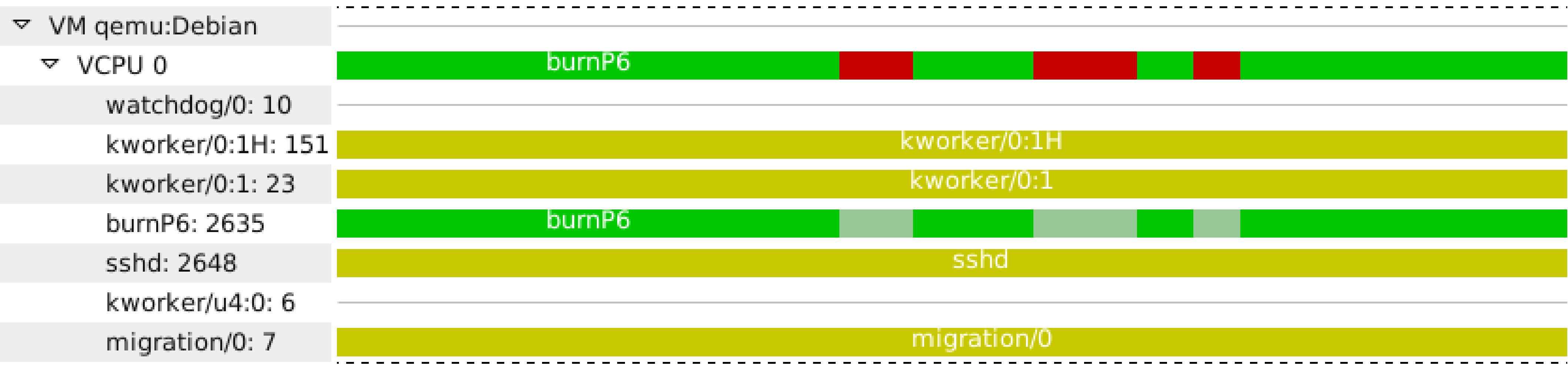
- 2 VM:
  - Debian and Ubuntu
  - vCPU 0 and vCPU 1 are complementary; fighting over the same pCPU

TMF Virtual Machine View

- Detailed information of execution inside the VM
- Process burnP6 (TID 2635) is deprived from the pCPU while the CPU time is still accounted for

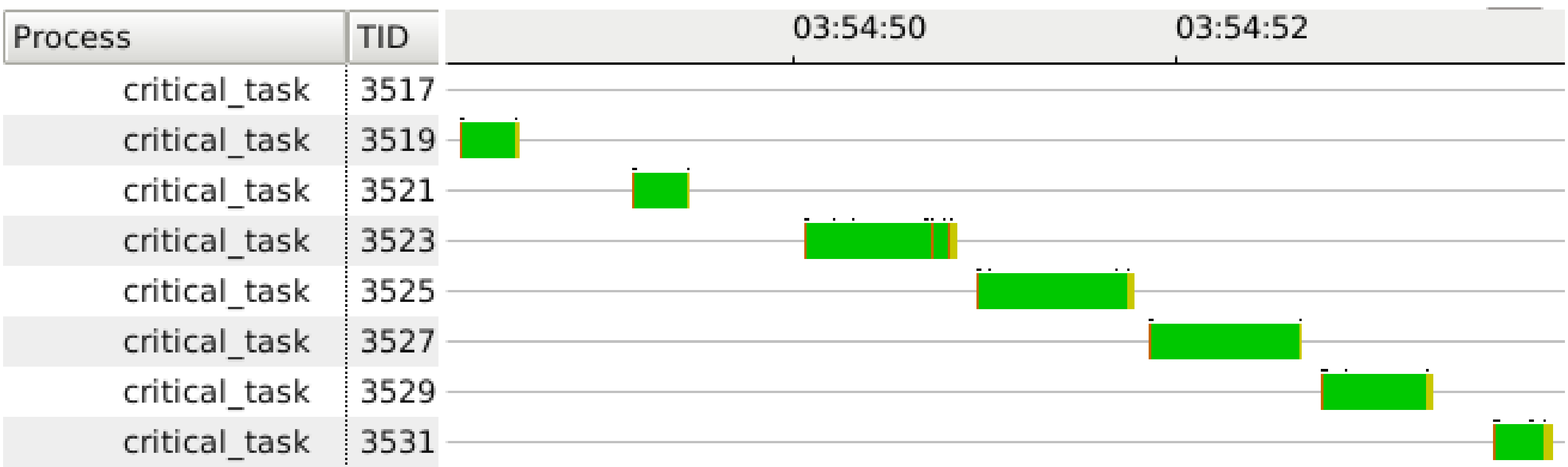
TMF Virtual Machine View

- Shows latency introduced by the hypervisor (ie. emulation in KVM) to the nanosecond scale



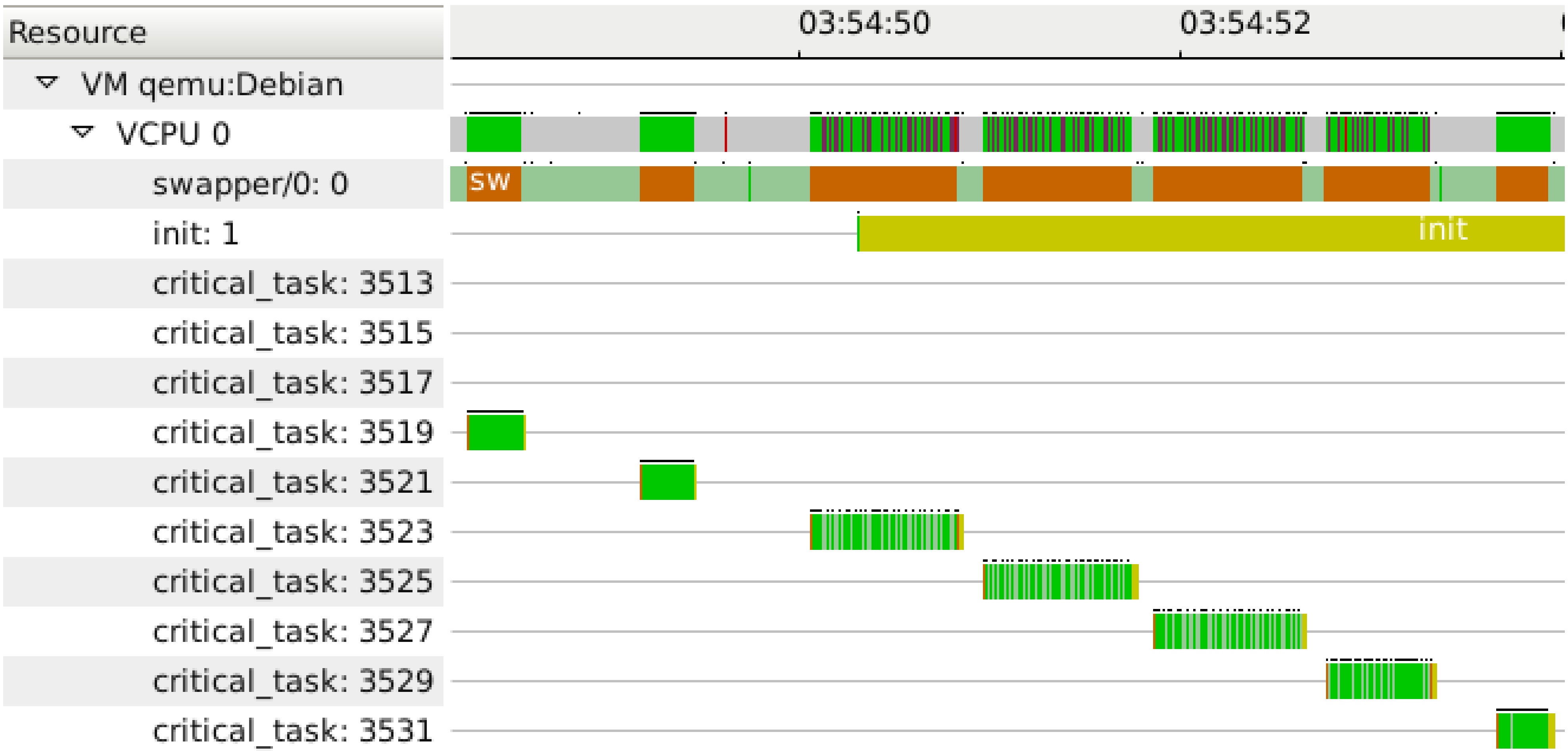
Use case

- Periodic critical task
- Inexplicably takes longer on some executions
- 100% CPU usage from the guest's point of view



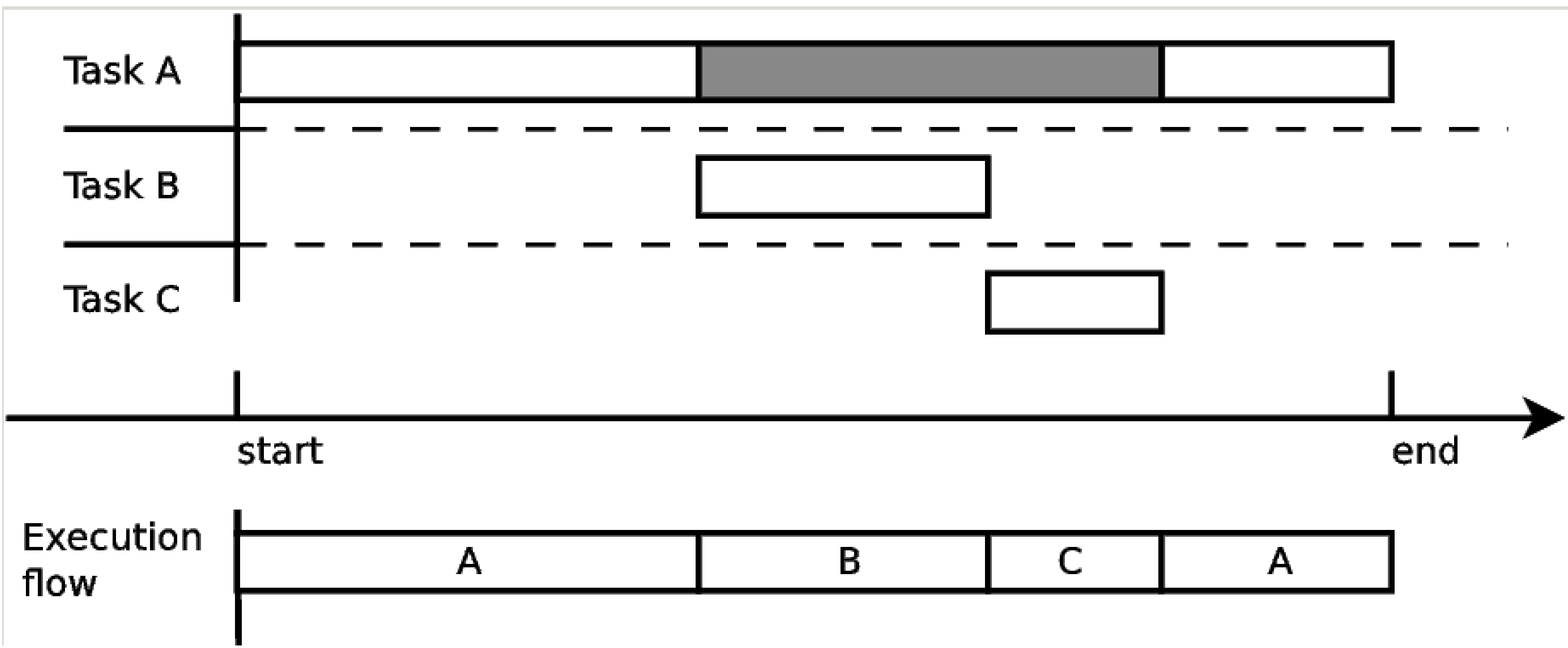
Use case

- VCPU is preempted on the host
- Invisible to the VM
- Duration of preemption is easily measurable



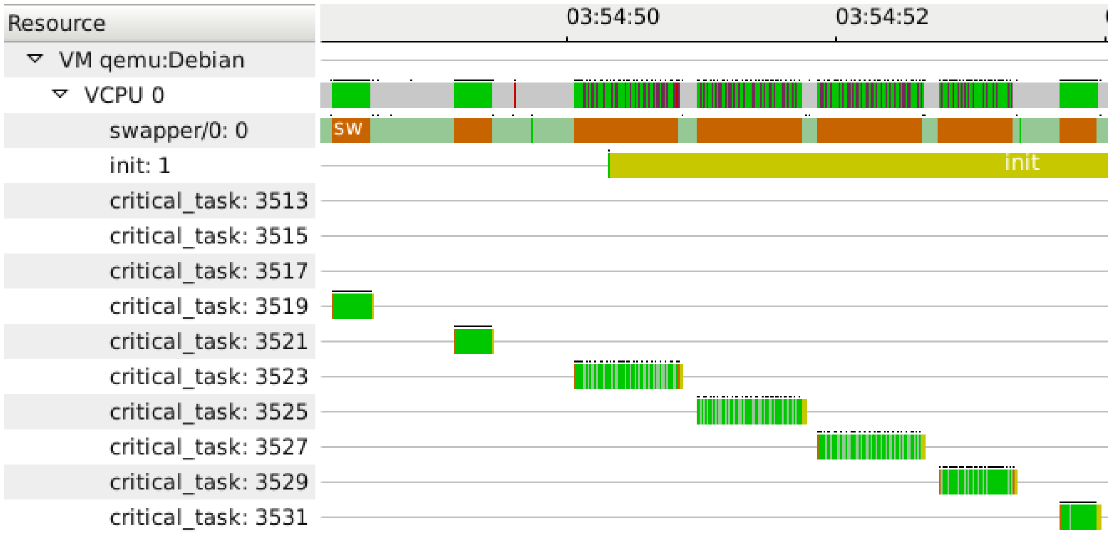
Execution flow recovery

- Build the execution flow centered around a certain task *A*
- List of execution intervals affecting the completion time of *A*
- Find the source of preemption across systems
- Example:

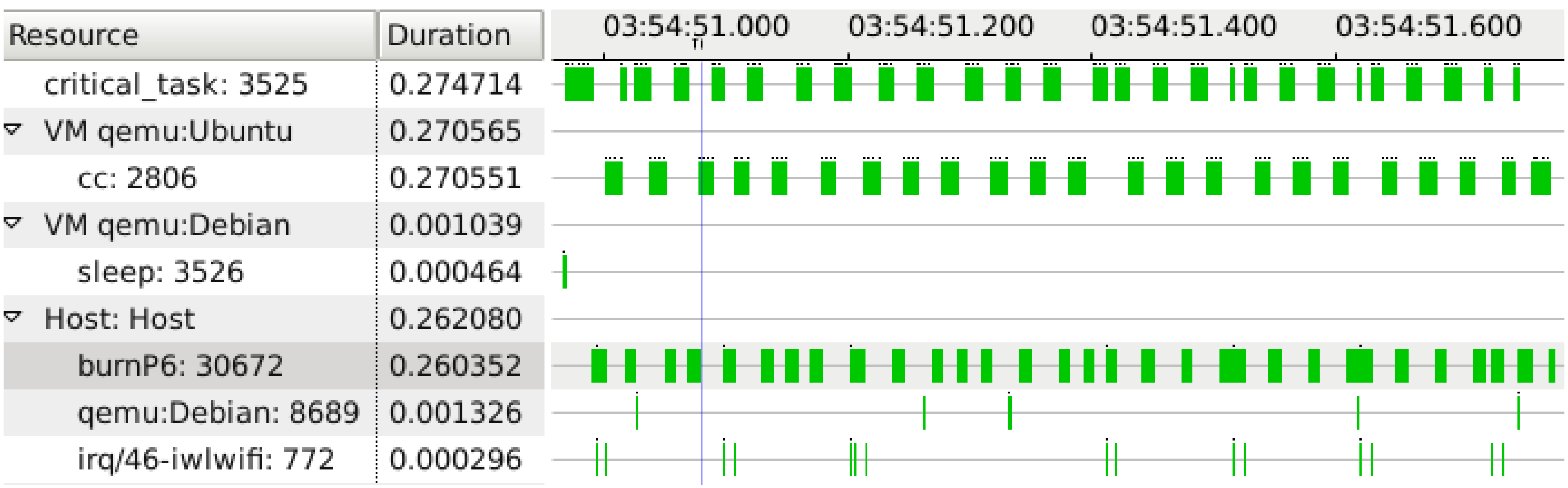


Execution flow recovery

- Previous example:



- Execution flow centered around task 3525:



Acknowledgements

- Ericsson
- CRSNG
- Professor Michel Dagenais
- Geneviève Bastien
- Francis Giraldeau
- DORSAL Lab