



SOFTWARE DEBUGGING AND MONITORING FOR MULTI-CORE SYSTEMS

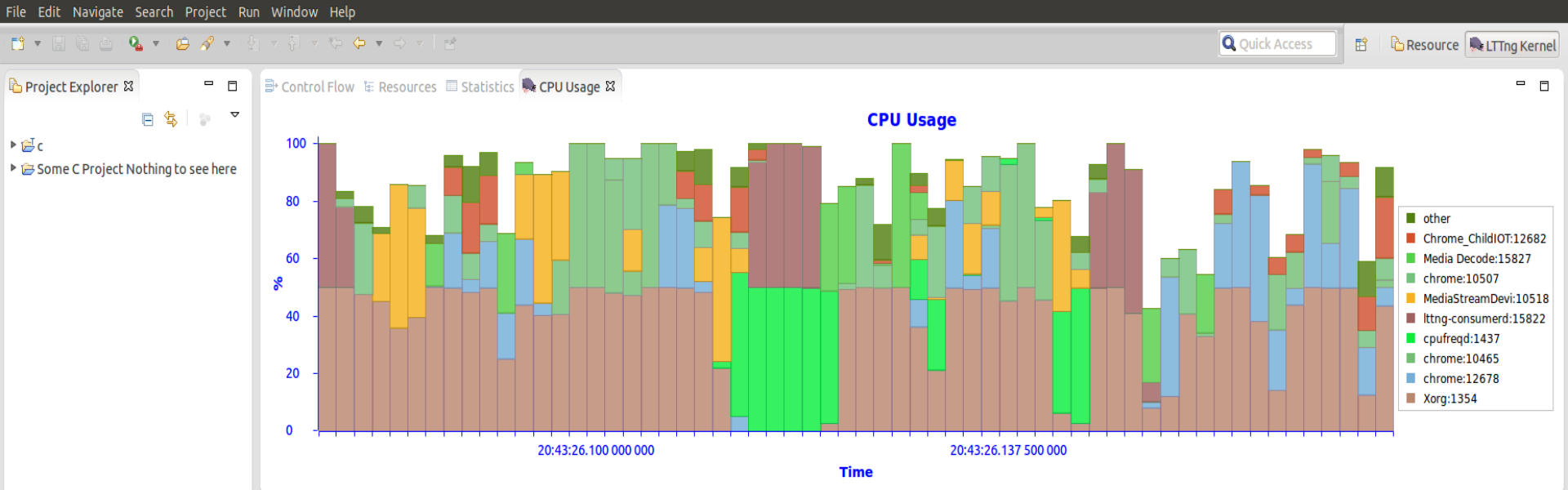
Naser Ezzati
DORSAL Lab
Polytechnique Montreal, Canada

Sharif-DNSL
Jan 2016

AGENDA

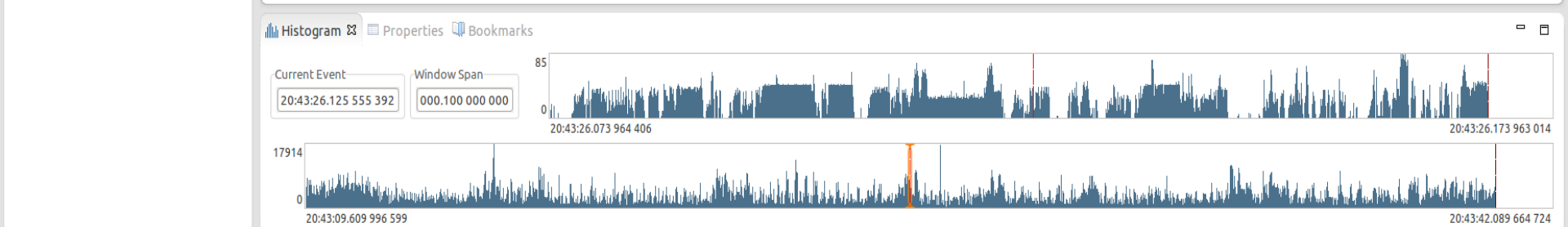
- Problem Faced
- Static and Dynamic Analysis
- Debugging vs Tracing
- Trace Analysis Tools
- Research Tracks
- Conclusion





YoutubeHD

Timestamp	Channel	Event Type	Content
<srch>	<srch>	<srch>	<srch>
20:43:26.125 534 23	channel0_0	exit_syscall	ret=0
20:43:26.125 537 72	channel0_0	sys_clock_gettime	which_clock=1, tp=0xbfdb8728
20:43:26.125 540 16	channel0_0	exit_syscall	ret=0
20:43:26.125 543 72	channel0_0	sys_clock_gettime	which_clock=1, tp=0xbfdb8778
20:43:26.125 545 96	channel0_0	exit_syscall	ret=0
20:43:26.125 553 01	channel0_0	sys_clock_gettime	which_clock=1, tp=0xbfdb8758
20:43:26.125 555 35	channel0_0	exit_syscall	ret=0
20:43:26.125 559 72	channel0_0	sys_clock_gettime	which_clock=1, tp=0xbfdb8728
20:43:26.125 562 02	channel0_0	exit_syscall	ret=0
20:43:26.125 565 93	channel0_0	sys_clock_gettime	which_clock=1, tp=0xbfdb8828
20:43:26.125 568 17	channel0_0	exit_syscall	ret=0
20:43:26.125 570 05	channel0_0	sys_clock_gettime	which_clock=1, tp=0xbfdb8728
20:43:26.125 572 43	channel0_0	exit_syscall	ret=0
20:43:26.125 574 52	channel0_0	sys_clock_gettime	which_clock=1, tp=0xbfdb8778



PROBLEM FACED

- Today's systems are composed of computers or virtual machines that interact between themselves.
- Understand the running behavior of those systems or find suspicious activities is more difficult.
- How to verify if the system is working as intended?
- Why is the system slow? Where is the bottleneck?
- Why do we get this incorrect answer once in a billion times?
- Are there intrusion attempts? Did they succeed?
- Are we leaking information?



STATIC VS. DYNAMIC ANALYSIS

○ Static Analysis

- Source codes and other artifacts
 - Not available
 - Outdated
 - Difficult to analysis

○ Dynamic Analysis

- Runtime behaviour analysis
- Tracing
 - Performance bottlenecks



TRACING

○ *What is Tracing?*

- Process of collecting information about the program's execution
 - Trace-points
 - Inserted before *compile-time, enabled/disabled at run-time*
 - People can use them to extract useful information without having to know the code
- The later analysis of this information may help us understand why a part of the software is not behaving as it is expected to.
- ‘Heisenbugs’ detection, hard to detect errors
 - Race conditions, Deadlocks, Non-deterministic behavior
 - Multiple layers
 - Middleware, VM, OS, hypervisor
 - Performance Problems
 - problems are not reproducible in the developer’s environment!

○ Debuggers:

- Debuggers are indispensable, but they only show a snapshot.



TRACING USECASES

- Finding cause of
 - Performance issues
 - Concurrency issues
 - Failures, crashes
- System-wide troubleshooting
 - Multiple layers, multi-core, multi-processor, multiple nodes, etc.
- Live monitoring of system in production
 - Resource usage (e.g. CPU load)
 - Raising alarms, warnings
 - Overload protection



TRACING TOOLS

- Classification:

- Userspace Tracing
 - Chrome://tracing
- Kernel Tracing
- Hardware Tracing

- ETW: Event Tracing for Windows

- Linux Tracing Tools:

- SystemTap
- Perf
- DTrace
- LTTng



USERSPACE TRACING

- UST Trace Library
- Example:
 - LTTng UST
 - Chrome:tracing



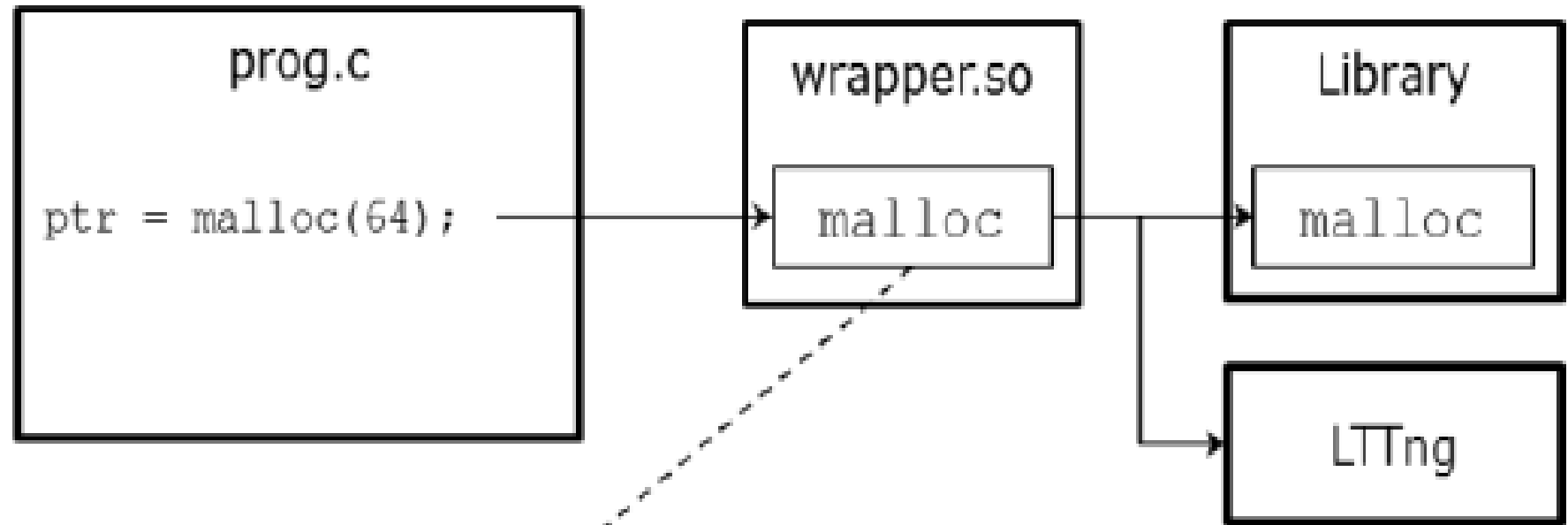
```
void function(void)
{
    int i = 0;
    long vals[3] = { 0x42, 0xCC, 0xC001CAFE };
    float flt = M_PI;

    [...]
    tracepoint(ust_tests_hello,
               tpctest,
               i,
               &vals,
               flt);

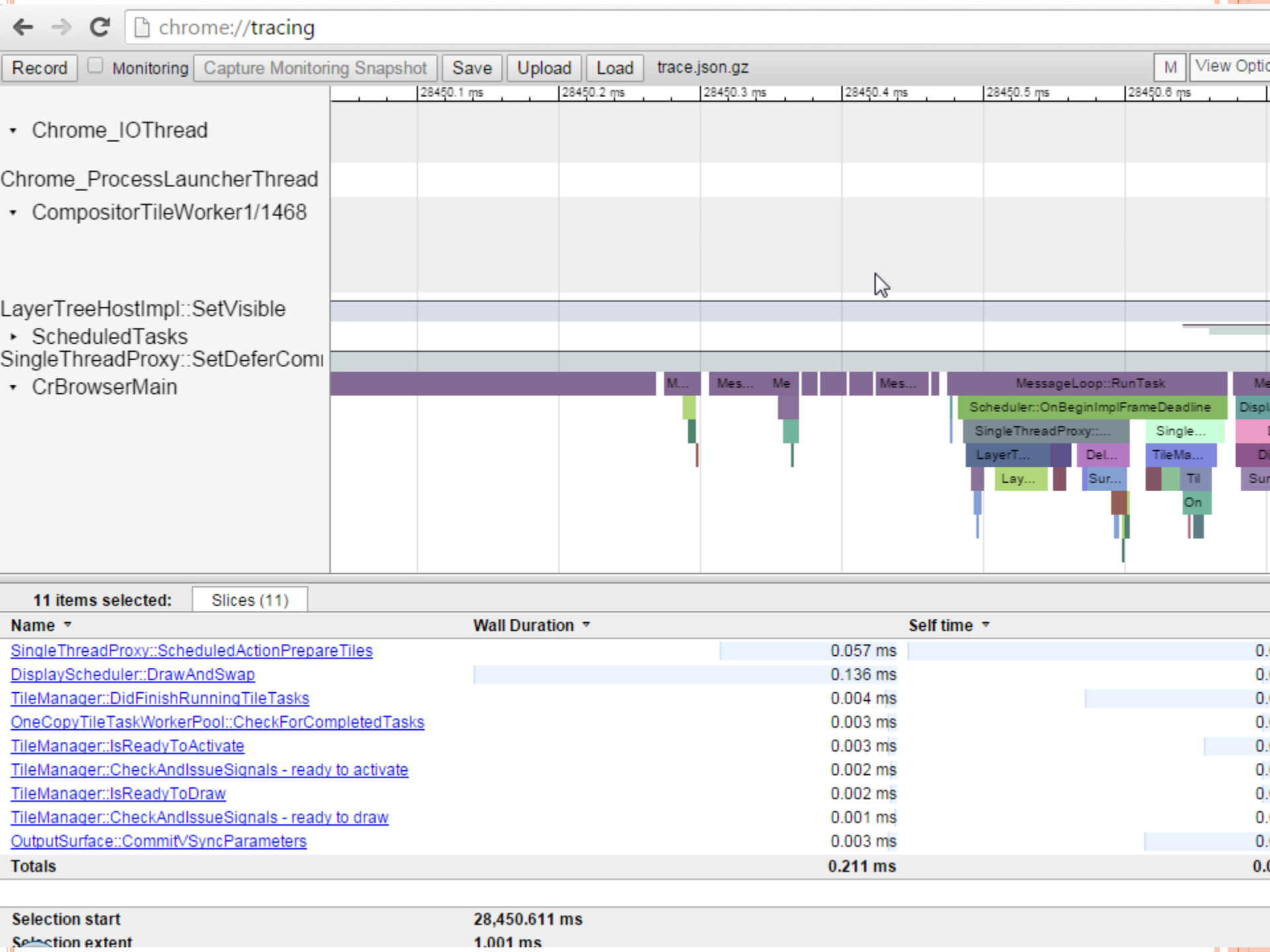
    [...]
}
```

```
TRACEPOINT_EVENT(  
    /* Provider name */  
    ust_tests_hello,  
  
    /* Tracepoint name */  
    tptest,  
  
    /* Type, variable name */  
    TP_ARGS(int, anint,  
            long *, values,  
            float, floatarg),  
  
    /* Type, field name, expression */  
    TP_FIELDS(ctf_integer(int, intfield, anint),  
              ctf_array(long, arrfield1, values, 3),  
              ctf_float(float, floatfield, floatarg))  
)
```

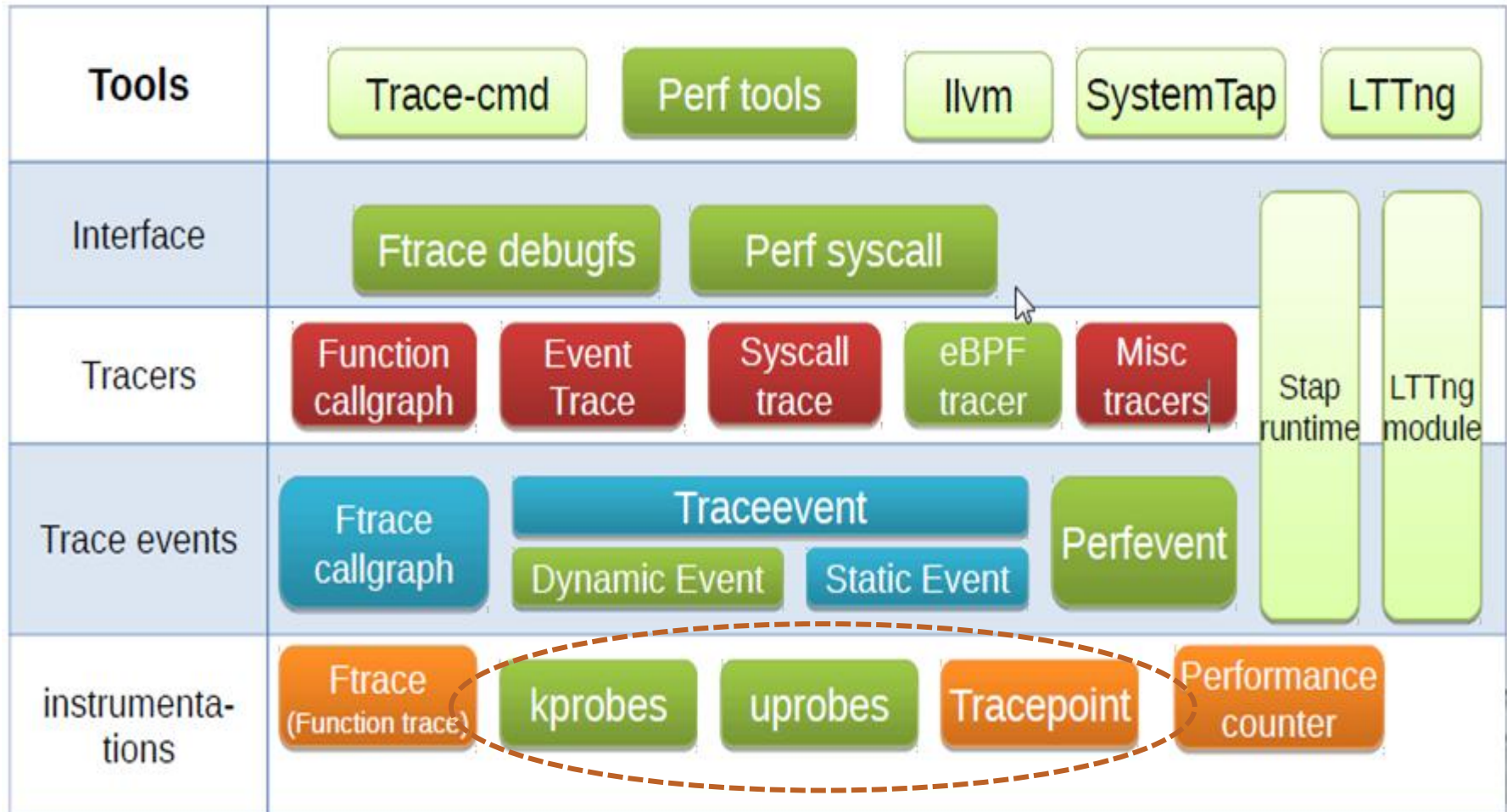
```
$ LD_PRELOAD=./wrapper.so ./prog
```



```
void* malloc(size_t size) // wrapper for malloc
{
    void* ret;
    static void* (*realmalloc)(size_t size) = NULL;
    if (realmalloc == NULL)
        realmalloc=dlsym(RTLD_NEXT, "malloc");
    ret = realmalloc(size);
    tracepoint(percepio, malloc, size, ret);
    return ret;
}
```

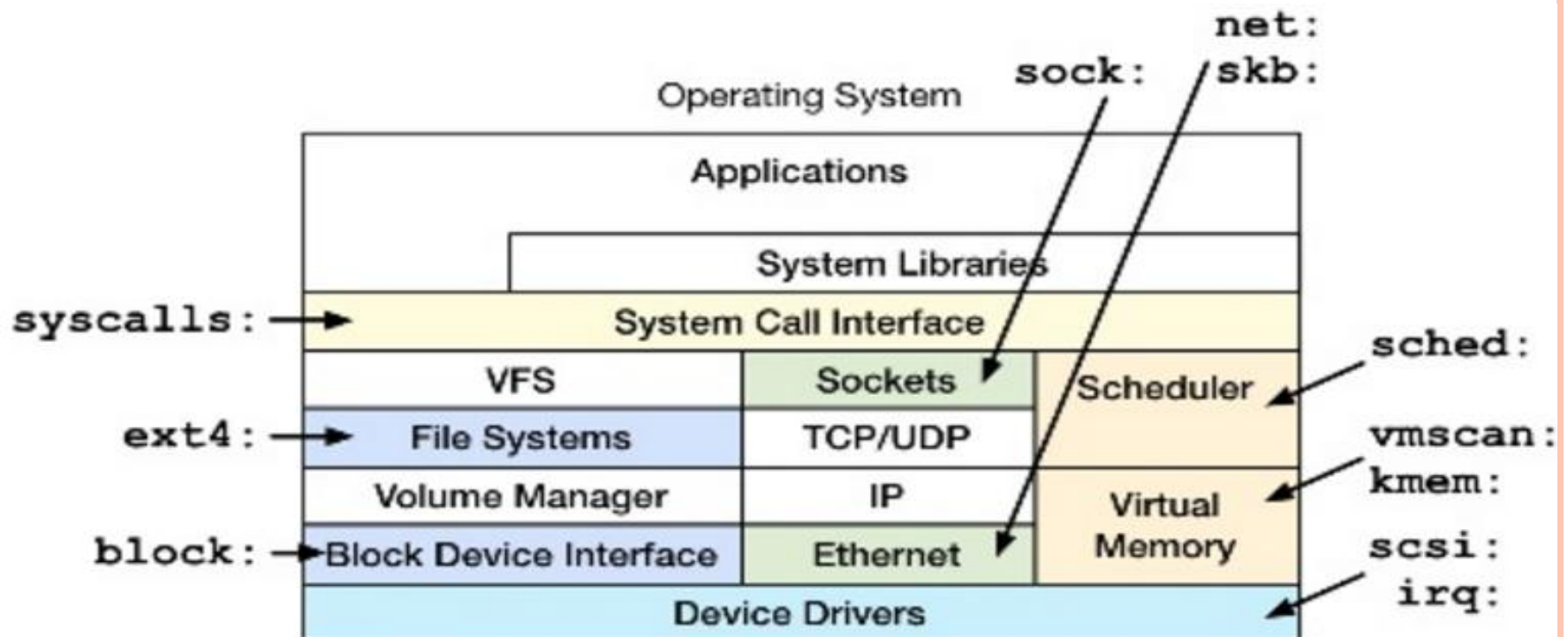


LINUX KERNEL TRACING



TRACEPOINT

- Statically placed at different logical places in the kernel
- More than 250 tracepoints
- `TRACE_EVENT()` macro



TRACE_EVENT MACRO

include/trace/events/sched.h

```
TRACE_EVENT(sched_switch,  
            TP_PROTO(struct task_struct *prev,  
                    struct task_struct *next),  
            TP_ARGS(prev, next),  
            TP_STRUCT__entry(  
                __array(char, prev_comm, TASK_COMM_LEN)  
                __field(pid_t, prev_pid)  
                __field(int, prev_prio)  
                __field(long, prev_state)  
            ),  
            ...  
            TP_fast_assign(  
                ...  
            );
```

Event Name

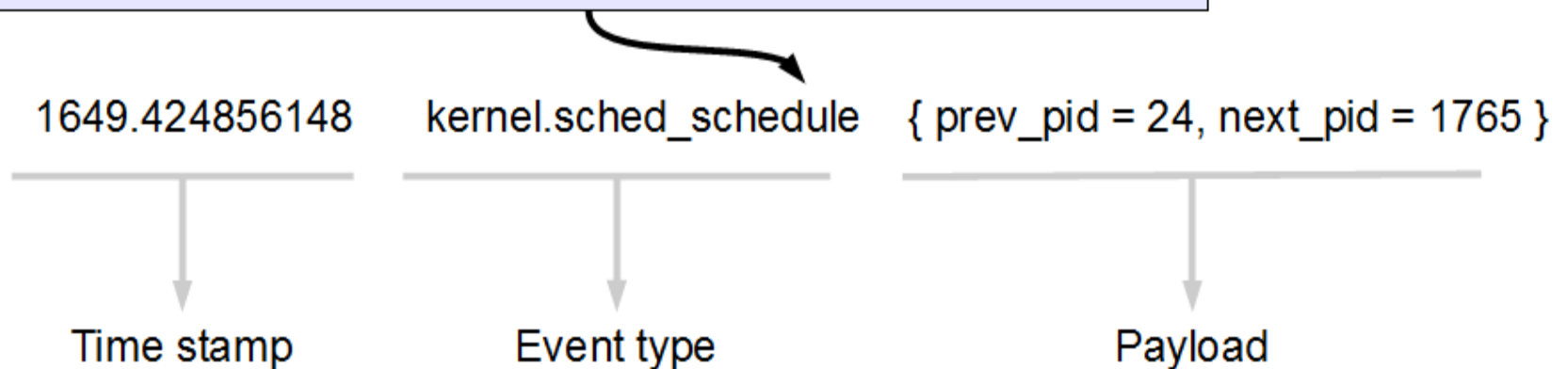
Trace function prototype

How to store data in tracer's ringbuffer

Fill the fields above

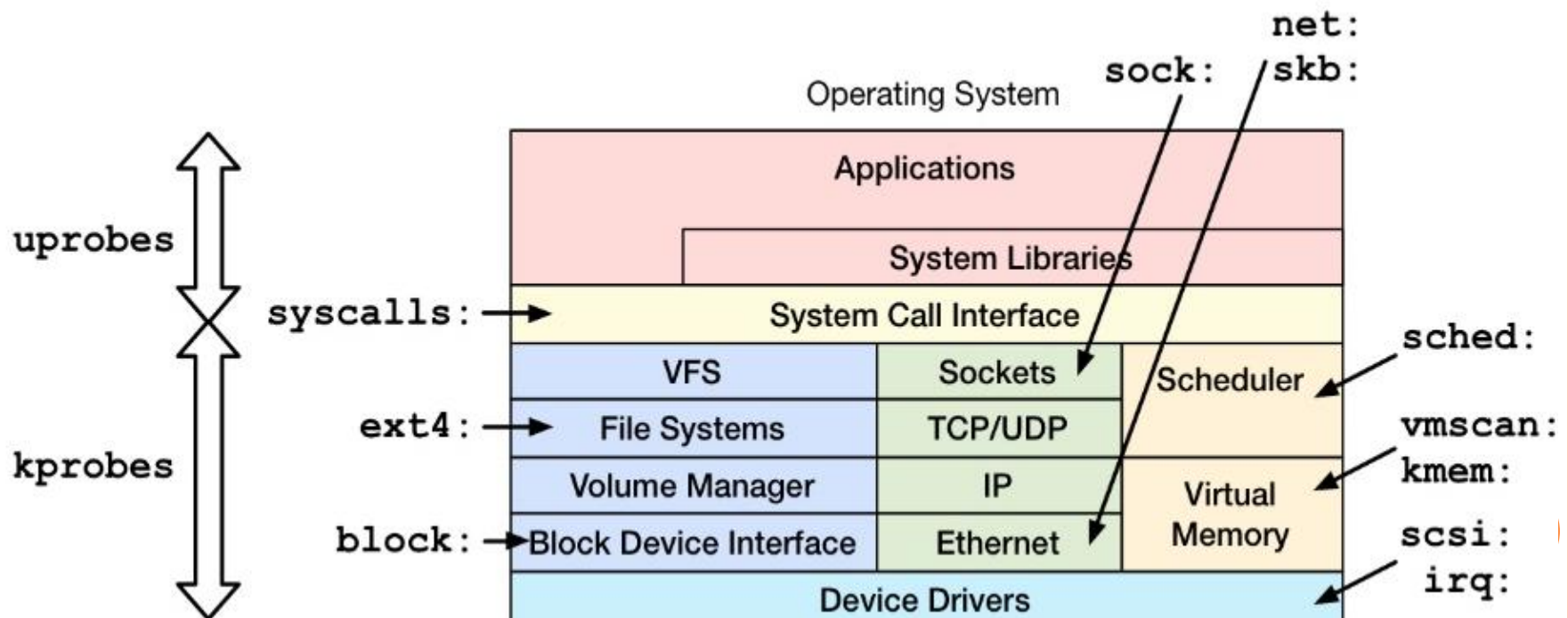
Linux/kernel/sched.c

```
/*  
 * context_switch - switch to the new MM and the new  
 * thread's register state.  
 */  
static inline void  
context_switch(struct rq *rq, struct task_struct *prev,  
              struct task_struct *next)  
{  
    struct mm_struct *mm, *oldmm;  
  
    prepare_task_switch(rq, prev, next);  
    trace_sched_switch(prev, next);  
    mm = next->mm;  
    oldmm = prev->active_mm;  
    ...  
}
```



PROBES

- Kprobes: dynamic kernel tracing
 - Function calls, returns, line numbers
- Uprobes: dynamic user-level tracing



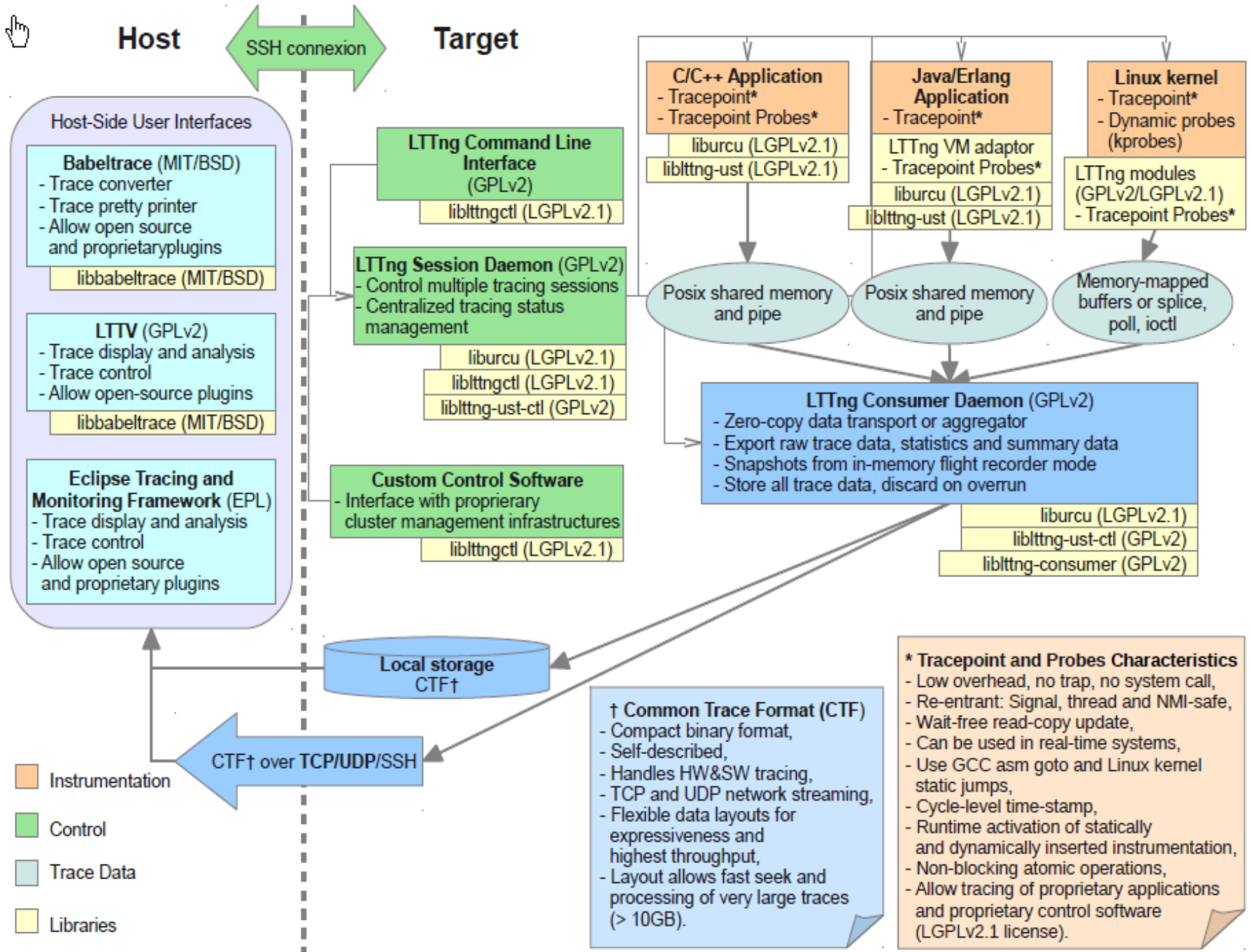
LTTNG + TRACE COMPASS



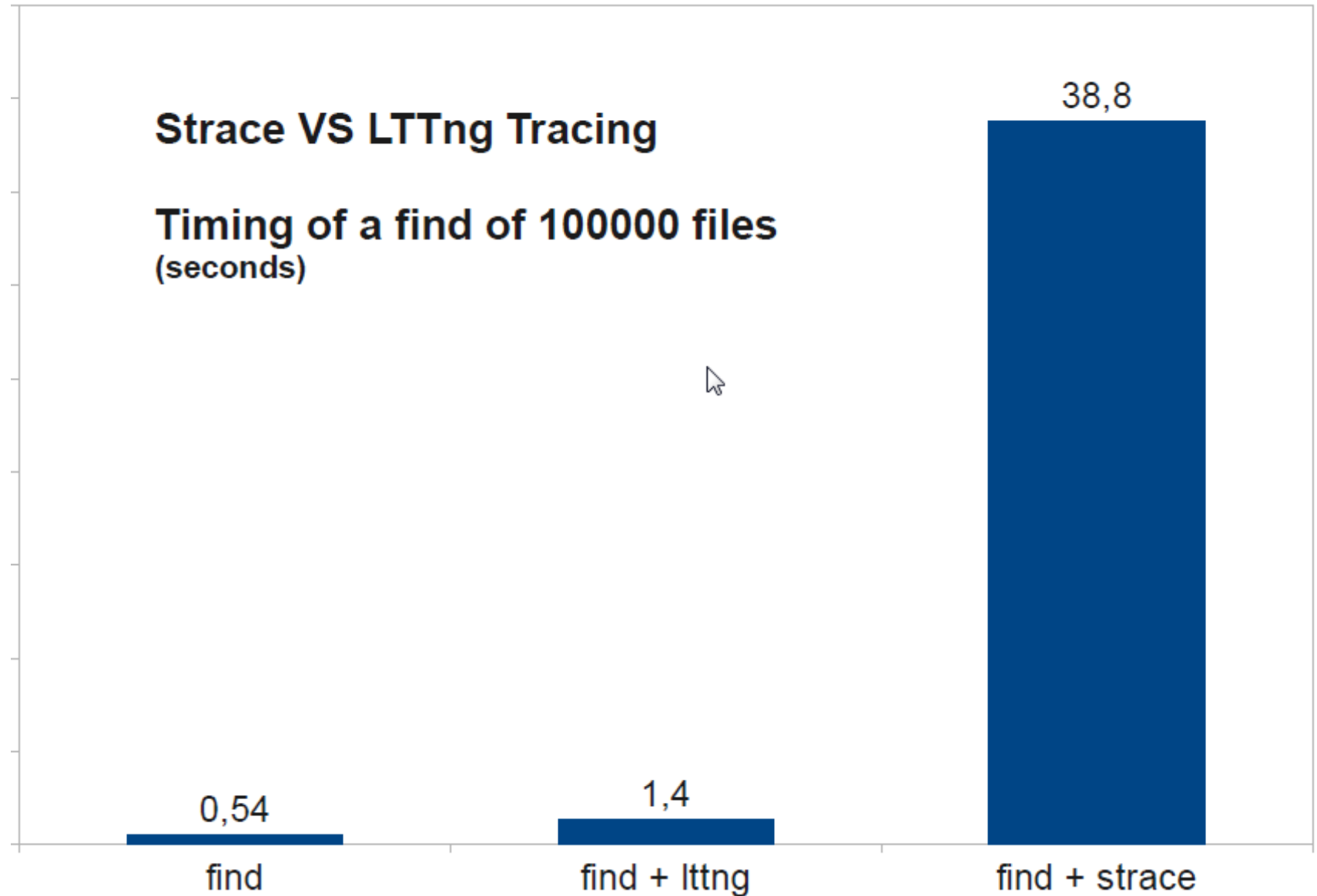
LTTNG: LINUX TRACE TOOLKIT NEXT GENERATION

- Scalable tracer
- Fast tracer
- Minimal impact and overhead on the target
- Output data in unified format (CTF)
- Flight-recorder
- Support kernel and user-space tracing
- Easy installation:
 - Support kernel from 2.6.38 +
- Linux Distribution:
 - Ubuntu
 - Debian
 - Fedora
 - Arch
 - Suse
 - Red Hat
 - Other OS:
 - Android, FreeBSD, Cygwin

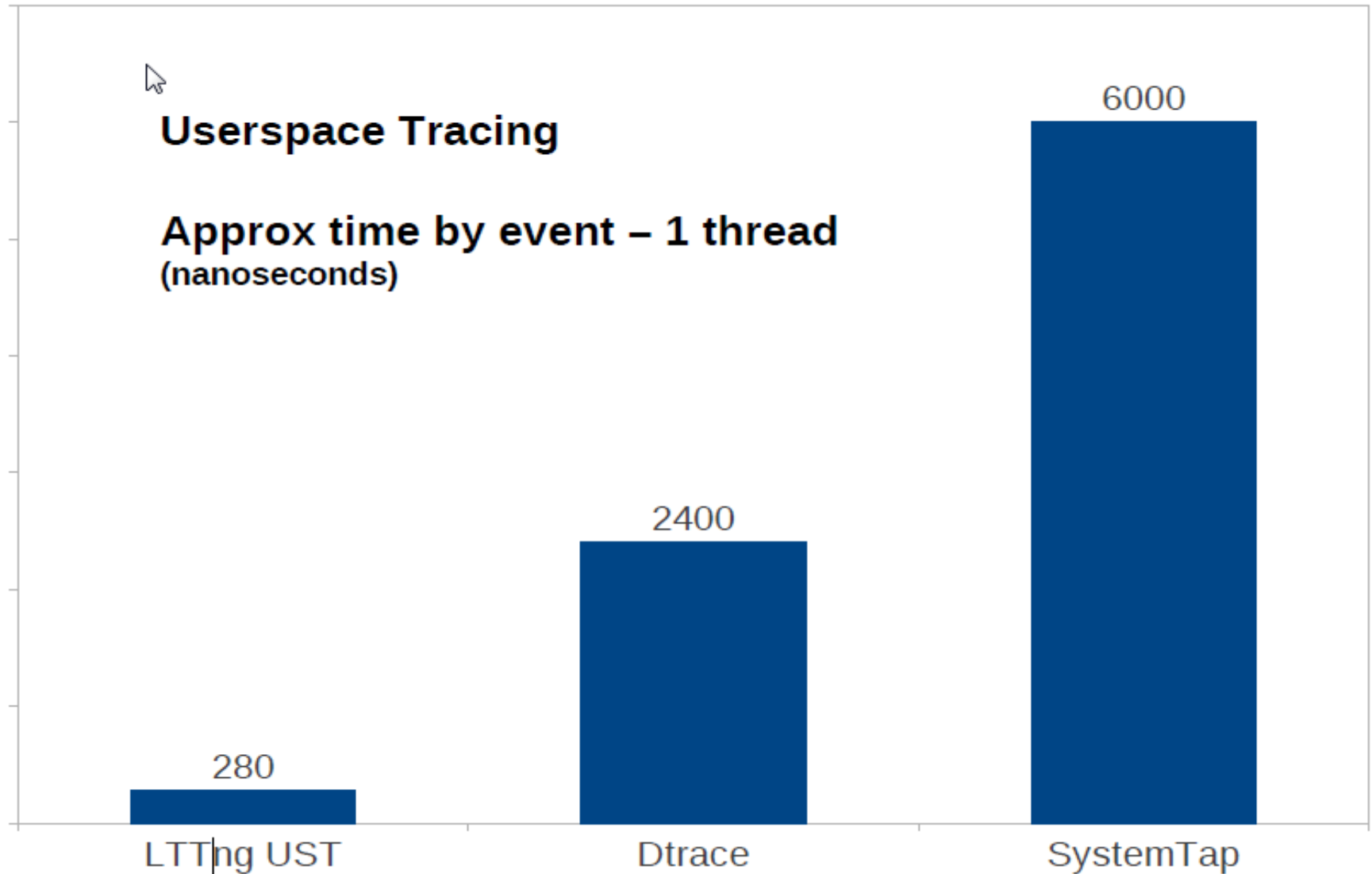




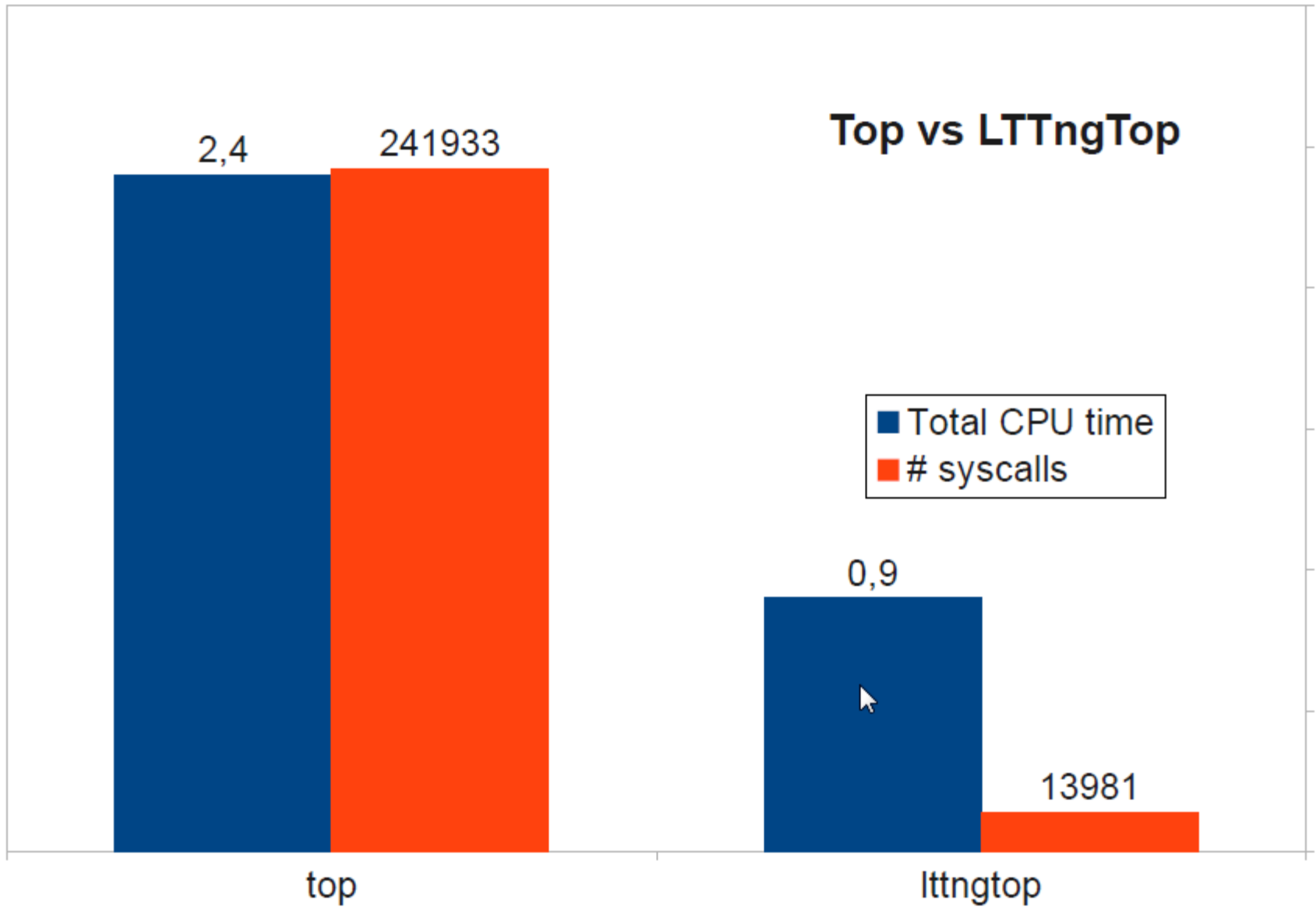
LTTNG IS FAST! (KERNEL)



LTTNG IS FAST! (UST)



LTTNG IS FAST!



EXAMPLE KERNEL TRACE SESSION: SIMPLE AND UNIFIED COMMAND LINE

\$lttng create session

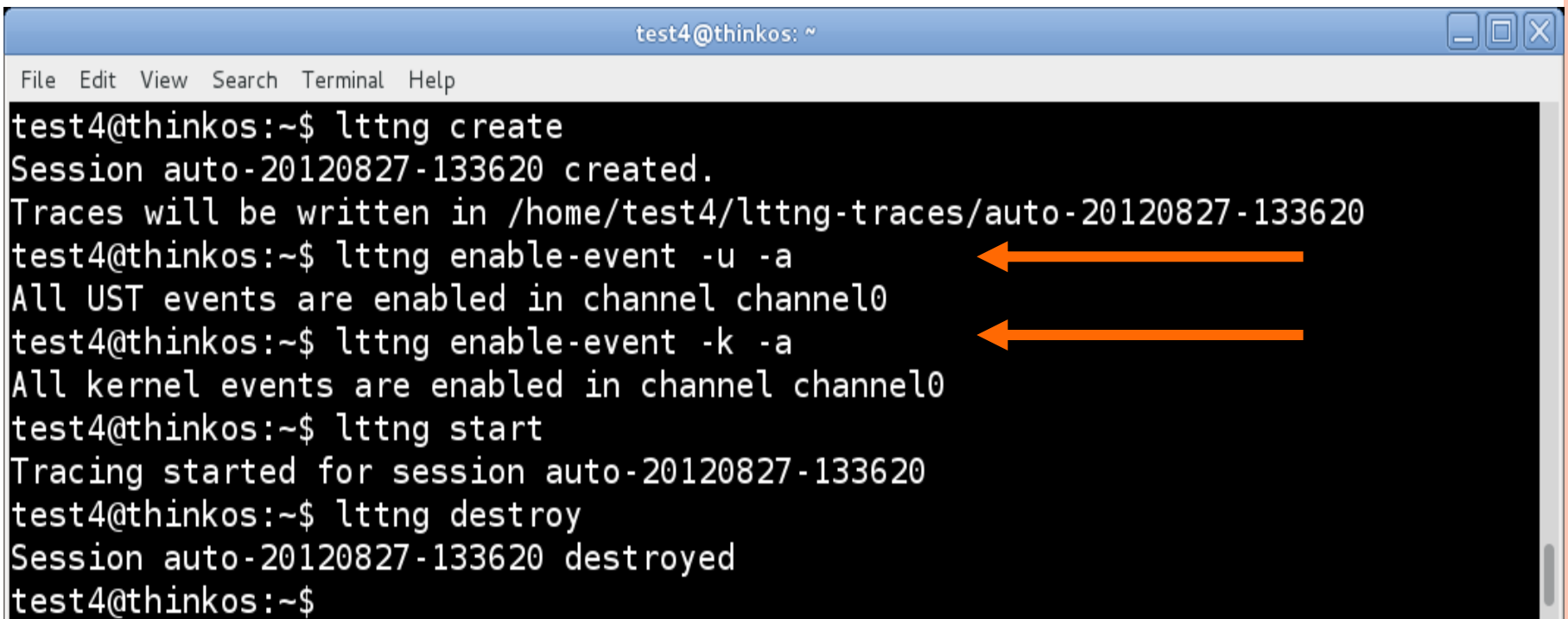
\$lttng enable-event -k-a (or -k sched_switch)

\$lttng enable-event -u -a

\$lttng start

\$lttng stop

\$lttng view

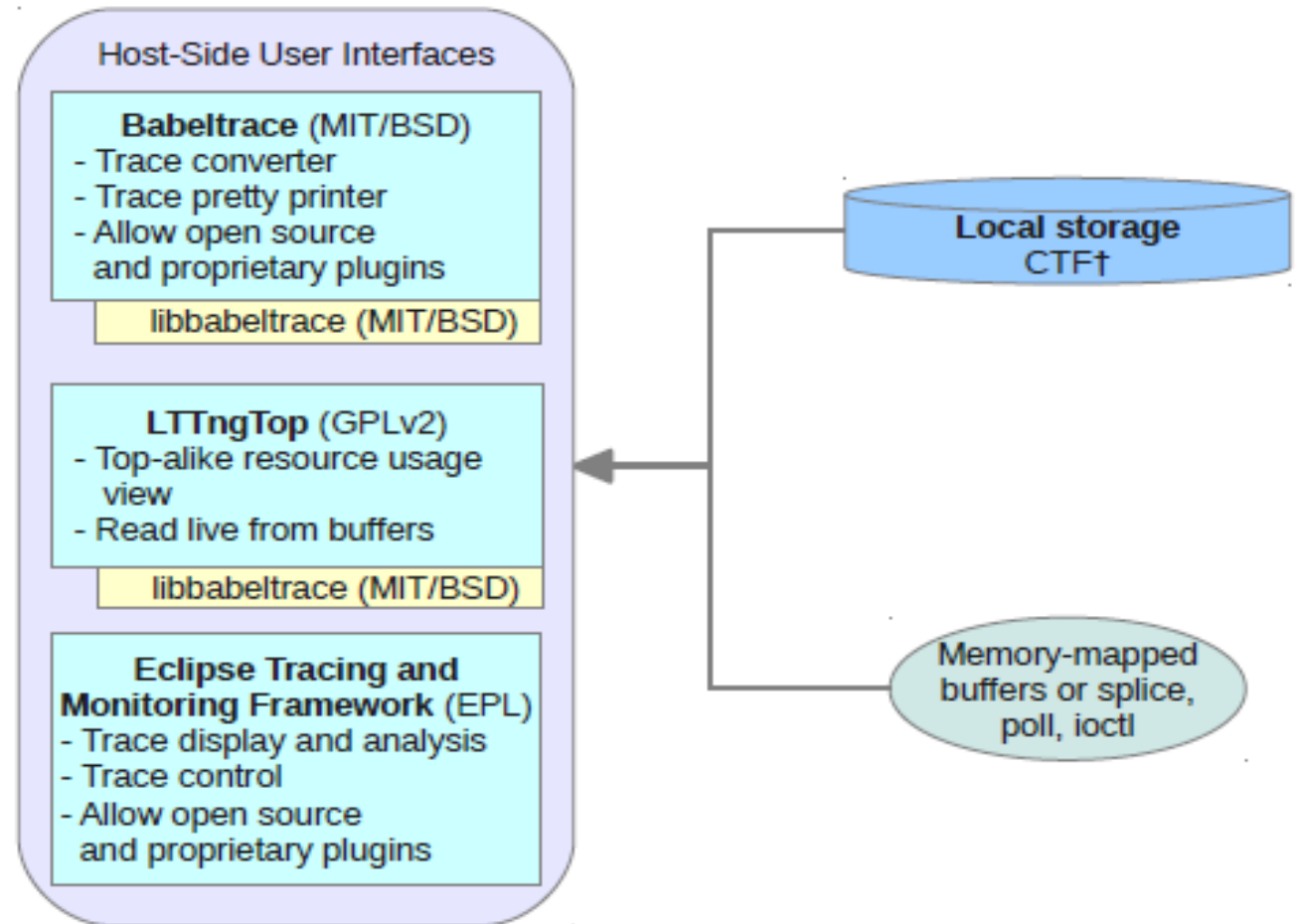


The screenshot shows a terminal window titled 'test4@thinkos: ~'. The terminal contains the following commands and output:

```
test4@thinkos:~$ lttng create
Session auto-20120827-133620 created.
Traces will be written in /home/test4/lttng-traces/auto-20120827-133620
test4@thinkos:~$ lttng enable-event -u -a
All UST events are enabled in channel channel0
test4@thinkos:~$ lttng enable-event -k -a
All kernel events are enabled in channel channel0
test4@thinkos:~$ lttng start
Tracing started for session auto-20120827-133620
test4@thinkos:~$ lttng destroy
Session auto-20120827-133620 destroyed
test4@thinkos:~$
```

Two orange arrows point to the output of the 'lttng enable-event -u -a' and 'lttng enable-event -k -a' commands.

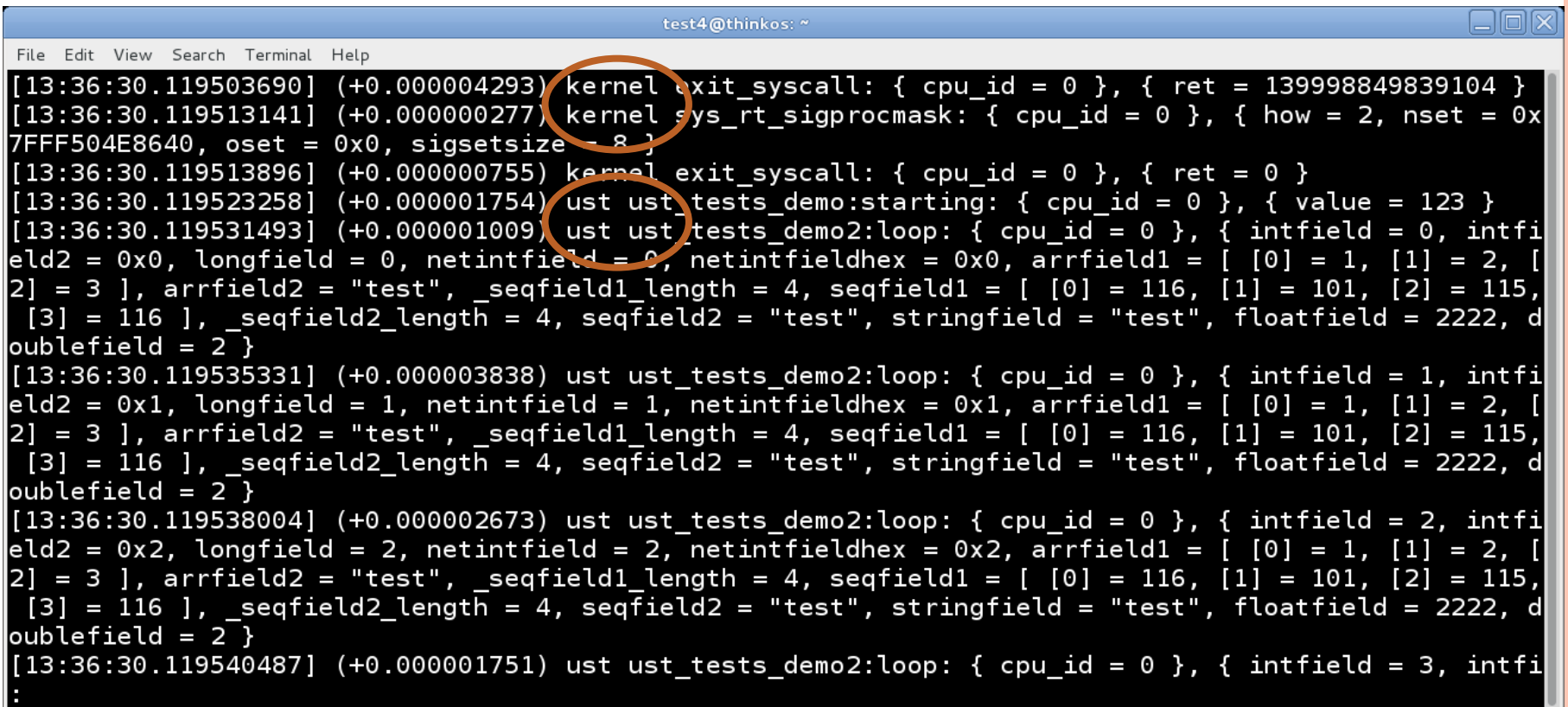
LTTNG COMMON TRACE FORMAT VIEWERS



Trace Data

Libraries

OUTPUT TRACE: BABELTRACE



```
test4@thinkos: ~
File Edit View Search Terminal Help
[13:36:30.119503690] (+0.000004293) kernel_exit_syscall: { cpu_id = 0 }, { ret = 139998849839104 }
[13:36:30.119513141] (+0.000000277) kernel_sys_rt_sigprocmask: { cpu_id = 0 }, { how = 2, nset = 0x
7FFF504E8640, oset = 0x0, sigsetsize = 8 }
[13:36:30.119513896] (+0.000000755) kernel_exit_syscall: { cpu_id = 0 }, { ret = 0 }
[13:36:30.119523258] (+0.000001754) ust_ust_tests_demo:starting: { cpu_id = 0 }, { value = 123 }
[13:36:30.119531493] (+0.000001009) ust_ust_tests_demo2:loop: { cpu_id = 0 }, { intfield = 0, intfi
eld2 = 0x0, longfield = 0, netintfield = 0, netintfieldhex = 0x0, arrfield1 = [ [0] = 1, [1] = 2, [
2] = 3 ], arrfield2 = "test", _seqfield1_length = 4, seqfield1 = [ [0] = 116, [1] = 101, [2] = 115,
[3] = 116 ], _seqfield2_length = 4, seqfield2 = "test", stringfield = "test", floatfield = 2222, d
oublefield = 2 }
[13:36:30.119535331] (+0.000003838) ust_ust_tests_demo2:loop: { cpu_id = 0 }, { intfield = 1, intfi
eld2 = 0x1, longfield = 1, netintfield = 1, netintfieldhex = 0x1, arrfield1 = [ [0] = 1, [1] = 2, [
2] = 3 ], arrfield2 = "test", _seqfield1_length = 4, seqfield1 = [ [0] = 116, [1] = 101, [2] = 115,
[3] = 116 ], _seqfield2_length = 4, seqfield2 = "test", stringfield = "test", floatfield = 2222, d
oublefield = 2 }
[13:36:30.119538004] (+0.000002673) ust_ust_tests_demo2:loop: { cpu_id = 0 }, { intfield = 2, intfi
eld2 = 0x2, longfield = 2, netintfield = 2, netintfieldhex = 0x2, arrfield1 = [ [0] = 1, [1] = 2, [
2] = 3 ], arrfield2 = "test", _seqfield1_length = 4, seqfield1 = [ [0] = 116, [1] = 101, [2] = 115,
[3] = 116 ], _seqfield2_length = 4, seqfield2 = "test", stringfield = "test", floatfield = 2222, d
oublefield = 2 }
[13:36:30.119540487] (+0.000001751) ust_ust_tests_demo2:loop: { cpu_id = 0 }, { intfield = 3, intfi
:
```

OUTPUT TRACE: SYSTEM CALL TRACE

lttng enable-event --syscall -a

```
compudj@squeeze-amd64: ~  
File Edit View Terminal Help  
timestamp = 3325007286161, name = sys_brk, stream.packet.context = { cpu_id = 1 }, event.fields = { brk = 28622848 }  
timestamp = 3325007293436, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 28622848 }  
timestamp = 3325007310064, name = sys_read, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 3, buf = 0x1B48008, count = 9645 }  
timestamp = 3325007322188, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 9645 }  
timestamp = 3325007323370, name = sys_close, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 3 }  
timestamp = 3325007325129, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }  
timestamp = 3325007362552, name = sys_open, stream.packet.context = { cpu_id = 1 }, event.fields = { filename = "/root/.bash_history", flags = 513, mode = 3 }  
timestamp = 3325007408083, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 3 }  
timestamp = 3325007409590, name = sys_write, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 3, buf = 0x1B48081, count = 9524 }  
timestamp = 3325007468932, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 9524 }  
timestamp = 3325007470405, name = sys_close, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 3 }  
timestamp = 3325007472410, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }  
timestamp = 3325007477613, name = sys_rt_sigprocmask, stream.packet.context = { cpu_id = 1 }, event.fields = { how = 0, nset = 0x7FFF28A2A040, oset = 0x7FFF }  
timestamp = 3325007479187, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }  
timestamp = 3325007480290, name = sys_ioctl, stream.packet.context = { cpu_id = 1 }, event.fields = { fd = 255, cmd = 21520, arg = 140733875134380 }  
timestamp = 3325007484139, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }  
timestamp = 3325007485012, name = sys_rt_sigprocmask, stream.packet.context = { cpu_id = 1 }, event.fields = { how = 2, nset = 0x7FFF28A29FC0, oset = 0x0, s }  
timestamp = 3325007485944, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }  
timestamp = 3325007488210, name = sys_setpgid, stream.packet.context = { cpu_id = 1 }, event.fields = { pid = 0, pgid = 4235 }  
timestamp = 3325007490747, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }  
timestamp = 3325007581235, name = sys_exit_group, stream.packet.context = { cpu_id = 1 }, event.fields = { error_code = 0 }  
timestamp = 3325008132938, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 1 }  
timestamp = 3325008156466, name = sys_gettimeofday, stream.packet.context = { cpu_id = 1 }, event.fields = { tv = 0x7FFF0E61DC10, tz = 0x0 }  
timestamp = 3325008158565, name = exit_syscall, stream.packet.context = { cpu_id = 1 }, event.fields = { ret = 0 }  
:
```

LTTNGTOP

```
sinkpad:/home/julien 82x42
Statistics for interval [20:12:14.225117603, 20:12:15.225118300[
  CPUs      2      (max/cpu : 50.00%)
  Threads   367    (+1, 0)
  FDs       1651   (+5, -8)          5KB/sec

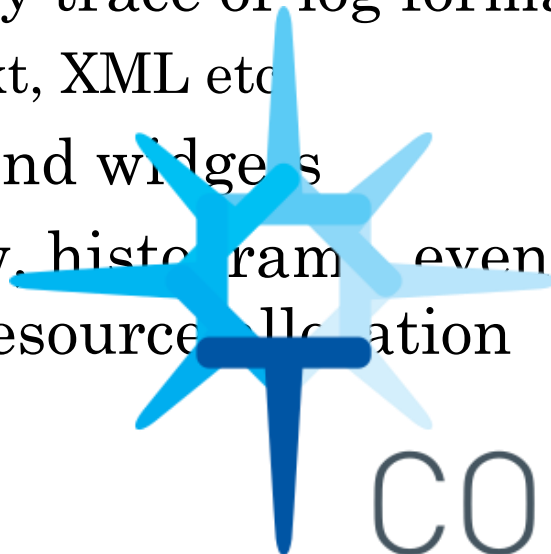
CPU Top
CPU(%)  PID      TID      NAME
0.31    4129     4129     firefox-bin
0.28    7709     7709     ifconfig
0.22    4196     4196     wicd
0.12    4971     4971     kworker/1:2
0.12    7447     7447     kworker/0:1
0.11    7373     7373     /usr/bin/x-term
0.07    2580     2580     Xorg
0.07    2441     2441     dbus-daemon
0.07    4227     4227     wicd-monitor
0.03    4075     4075     tor
0.03    6021     6021     xscreensaver
0.01    7298     7298     kworker/u:0
0.01    6808     6808     kworker/u:2
0.01    2498     2498     acpid
0.01    5957     5957     awesome
0.00    4114     4114     uml_switch
0.00    2585     2585     wpa_supplicant
0.00    7682     7682     lttngtop
0.00    18675    18675     migration/1
0.00    28967    28967     watchdog/0
0.00    28969    28969     watchdog/1
0.00    7682     7683     lttngtop
0.00    7682     7684     lttngtop
0.00    7588     7648     lttng-sessiond
0.00    7588     7588     lttng-sessiond
0.00    0        0        swapper/1
0.00    7274     7274     kworker/0:2

Status
Going forward in time
Manually moving forward
Manually moving forward
Manually moving forward

F2:CPUTop  F3:PerfTop  F4:IOTop  Enter:Details  Space:Highlight  q:Quit  r:Pref  :
```

TRACE COMPASS

- Eclipse IDE integration or RCP app
- Framework to build trace visualization and analysis tools
 - Mix kernel and userspace trace analysis
- Scalable: handle traces exceeding memory
- Extensible for any trace or log format
 - CTF, Binary, text, XML etc
- Reusable views and widgets
- Control flow view, histogram, event list, trace statistics, CPU/resource allocation



TRACE
COMPASS



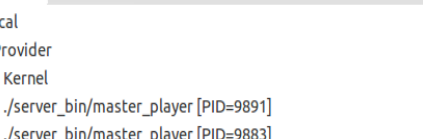
- ▼ Remote
 - 📁 Experiments [0]
 - ▼ 📁 Traces [4]
 - 🔗 seqSession-20140610-101412 [4]
 - ▼ 📁 ust [3]
 - ▼ 📁 pid [3]
 - ▶ 🖱 challenger-4708-20140610-101412
 - ▶ 🖱 master_player-4707-20140610-101412
 - ▶ 🖱 master_player-4715-20140610-101412
 - ▶ 🖱 kernel
- ▶ 📁 Tracing

Statistics **Sequence Diagram**

Component Interactions

```
sequenceDiagram
    participant Challenger
    participant Master
    Challenger->>Master: 
    Master-->>Challenger: BALL_REPLY
    Challenger->>Master: BALL_REQUEST
    Master-->>Challenger: BALL_REPLY
    Challenger->>Master: BALL_REQUEST
```

The diagram illustrates the sequence of messages between a Challenger and a Master. The Challenger initiates the interaction with an outgoing message to the Master. The Master responds with a `BALL_REPLY` message back to the Challenger. This sequence of messages repeats twice more, with the Challenger sending `BALL_REQUEST` and the Master replying with `BALL_REPLY`.



Control

- Local
 - Provider
 - Kernel
 - ./server_bin/master_player [PID=9891]
 - ./server_bin/master_player [PID=9883]
 - ./client_bin/challenger [PID=9884]
 - Sessions
 - seqSession
 - Kernel
 - UST global
 - channel0

```

int main()
{
    try
    {
        while (true)
        {
            seqTest();
        }
    }
    catch (std::exception &e)
    {
        std::cout << "Exception: " << e.what() << std::endl;
    }
}

```


Call Stack

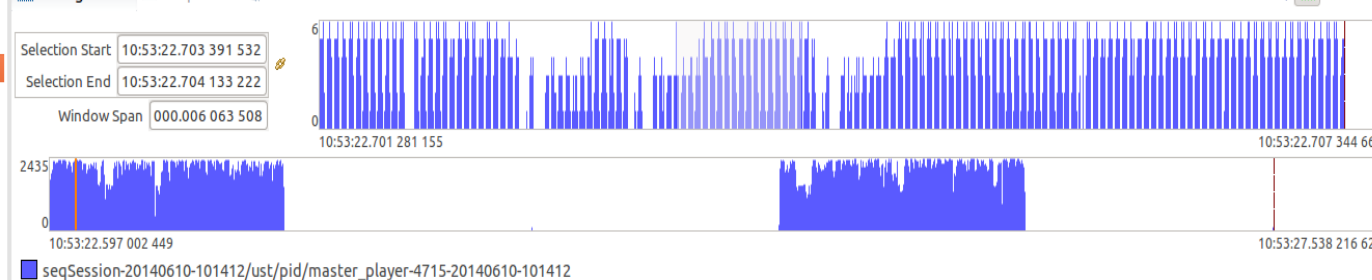
[illegible]

Ball Game View ✕

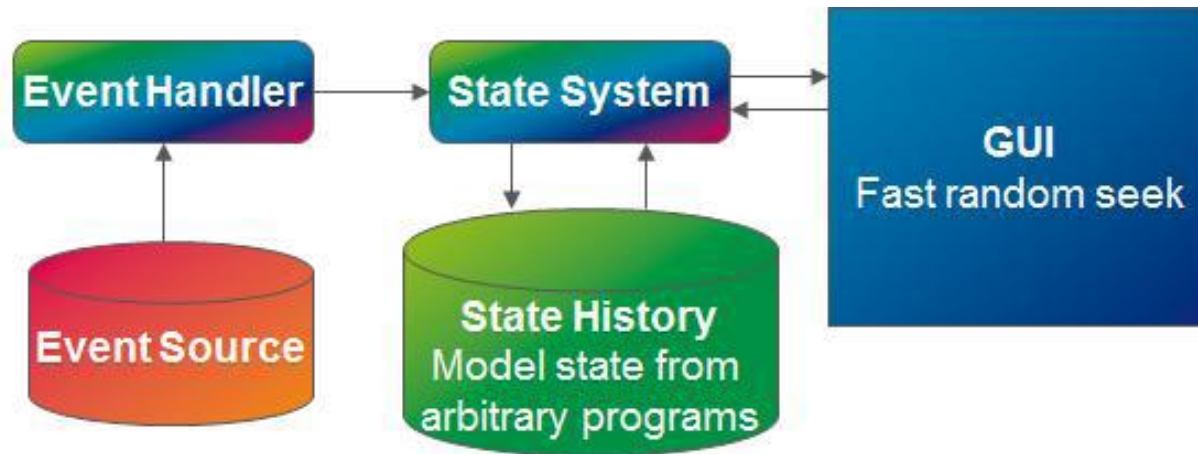
Name	10:53:22.702	10:53:22.704	10:53:22.706
seqSession-20140610-101412/ust/pid/master_player-4715-20140610-101412			
Challenger			

```
seqSession-20140610-101412/ust/pid/master_player-4715-20140610-101412
```

Timestamp	Source	Type	File	Content
 <srch>	<srch>	<srch>	<srch>	<srch>
10:53:22.703 381 997	2	lttnq_ust_cyg_profile:func_exit	channel0_2	addr=0x403520, call_site=0x402a32, context_vtid=4715, context_procname=master_player
10:53:22.703 383 791	2	lttnq_ust_cyg_profile:func_entry	channel0_2	addr=0x401ea0, call_site=0x402a32, context_vtid=4715, context_procname=master_player
10:53:22.703 385 575	2	lttnq_ust_cyg_profile:func_entry	channel0_2	addr=0x4021c0, call_site=0x402a32, context_vtid=4715, context_procname=master_player
10:53:22.703 387 369	2	lttnq_ust_cyg_profile:func_exit	channel0_2	addr=0x4021c0, call_site=0x402a32, context_vtid=4715, context_procname=master_player
10:53:22.703 389 238	2	lttnq_ust_cyg_profile:func_exit	channel0_2	addr=0x401ea0, call_site=0x402a32, context_vtid=4715, context_procname=master_player
10:53:22.703 391 599	2	lttnq_ust_cyg_profile:func_entry	channel0_2	addr=0x4040e0, call_site=0x40c255, context_vtid=4715, context_procname=master_player
10:53:22.703 393 358	2	lttnq_ust_cyg_profile:func_exit	channel0_2	addr=0x4040e0, call_site=0x40c255, context_vtid=4715, context_procname=master_player
10:53:22.703 395 278	2	lttnq_ust_cyg_profile:func_entry	channel0_2	addr=0x401e20, call_site=0x40c255, context_vtid=4715, context_procname=master_player
10:53:22.703 396 886	2	lttnq_ust_cyg_profile:func_exit	channel0_2	addr=0x401e20, call_site=0x40c255, context_vtid=4715, context_procname=master_player
10:53:22.703 398 575	2	ust_sequence:STATE	channel0_2	file=simple_server_main.cpp, line=198, name=Challenger, state=4, content=BALL_RECVD, context_vtid=4715
10:53:22.703 401 402	2	lttnq_ust_cyg_profile:func_entry	channel0_2	addr=0x409e0, call_site=0x40bf96, context_vtid=4715, context_procname=master_player
10:53:22.703 403 612	2	lttnq_ust_cyg_profile:func_entry	channel0_2	addr=0x408f90, call_site=0x409f27, context_vtid=4715, context_procname=master_player

Histogram 

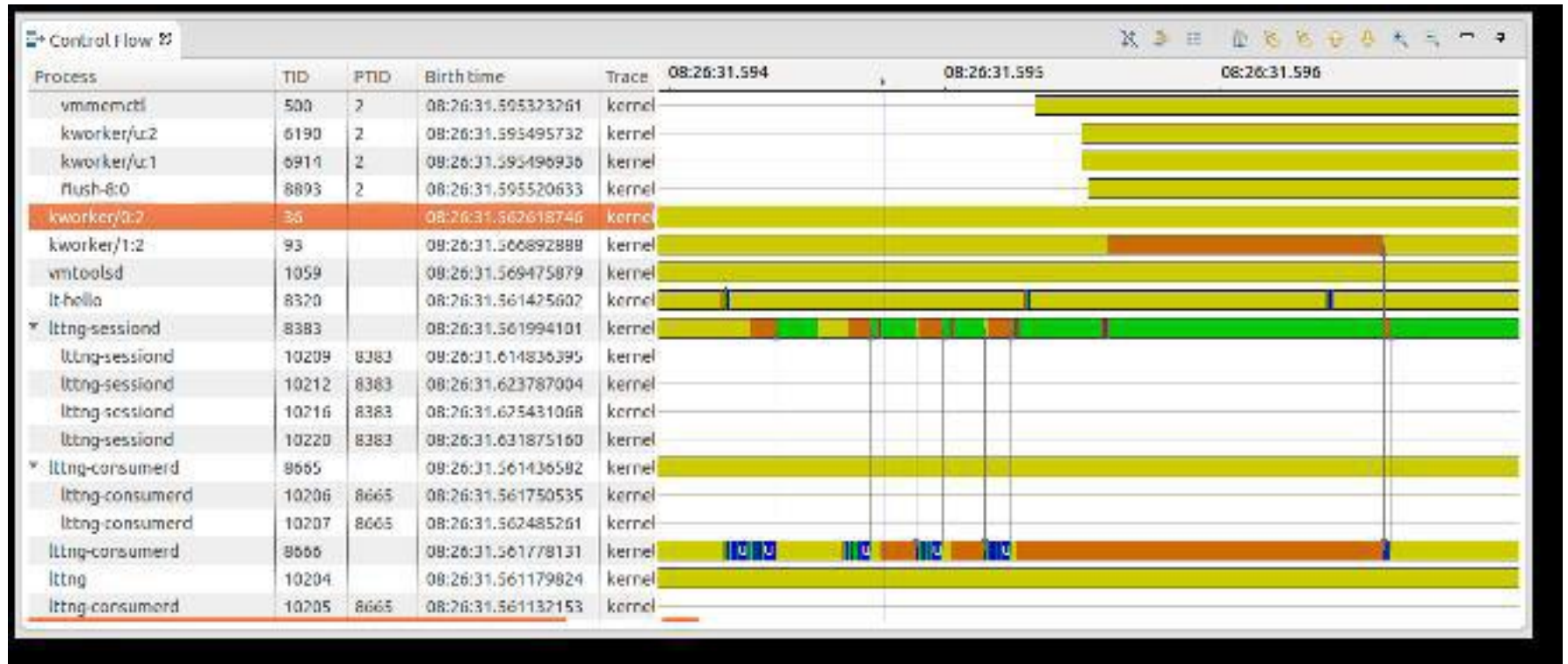
STATEFUL ANALYSIS: STATE SYSTEM



- State system abstracts events, analyses traces and creates models to be displayed



CONTROL FLOW VIEW

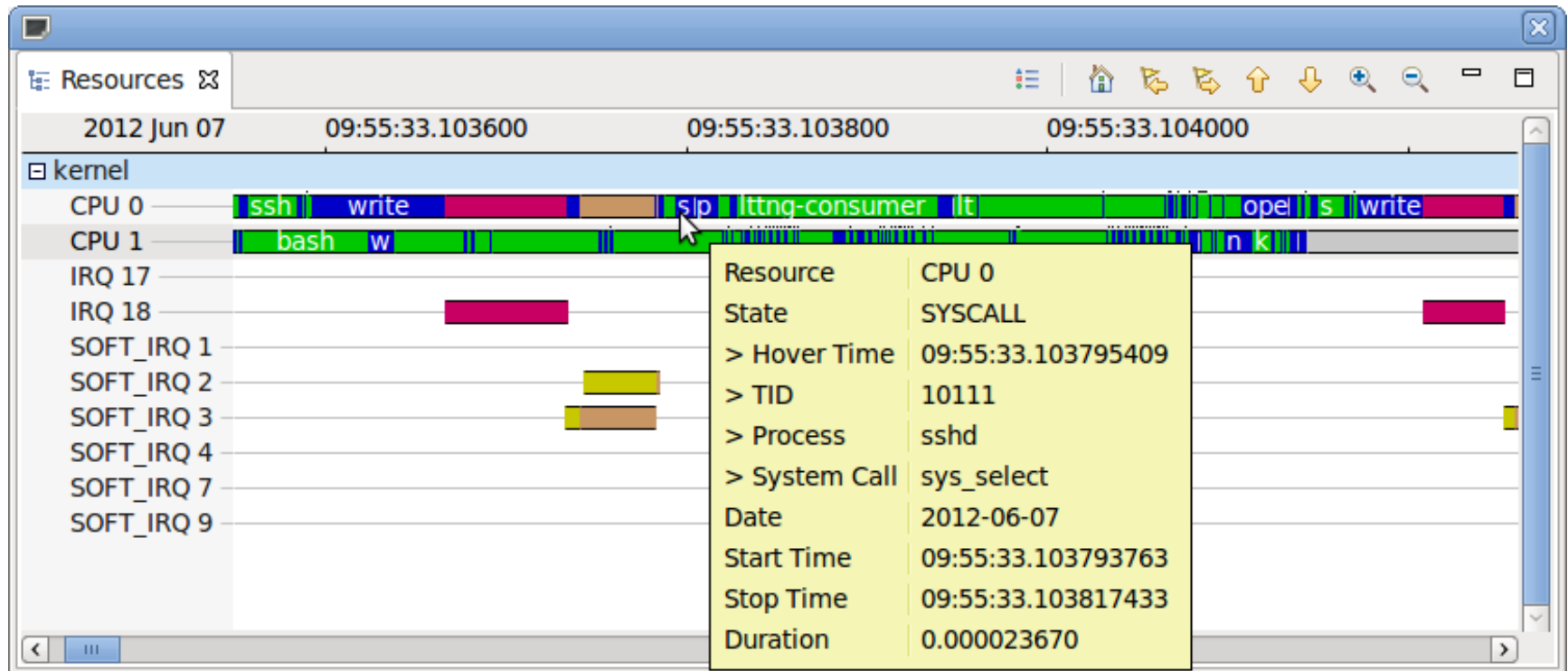


○ Display processes state changes (color-coded) over time

- USERMODE, SYSCALL, INTERRUPTED, WAIT_FOR_CPU, etc



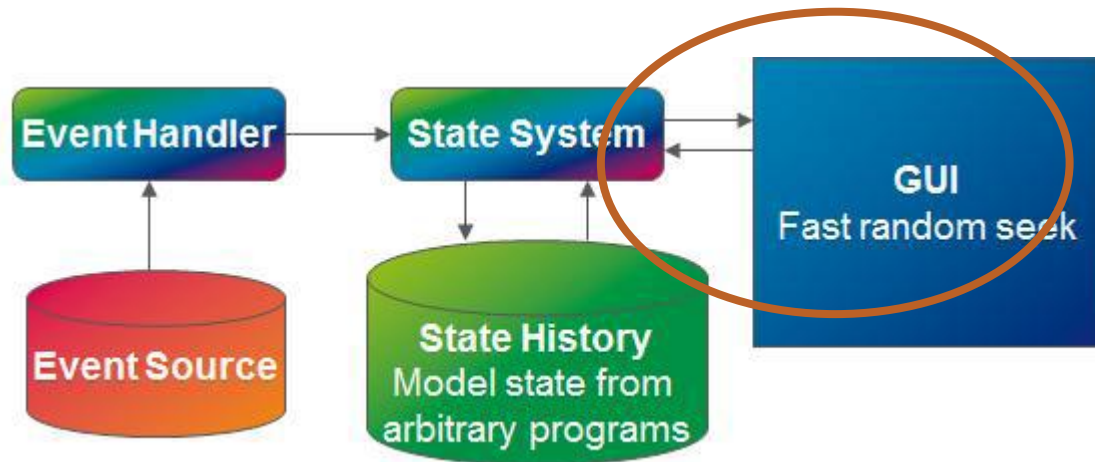
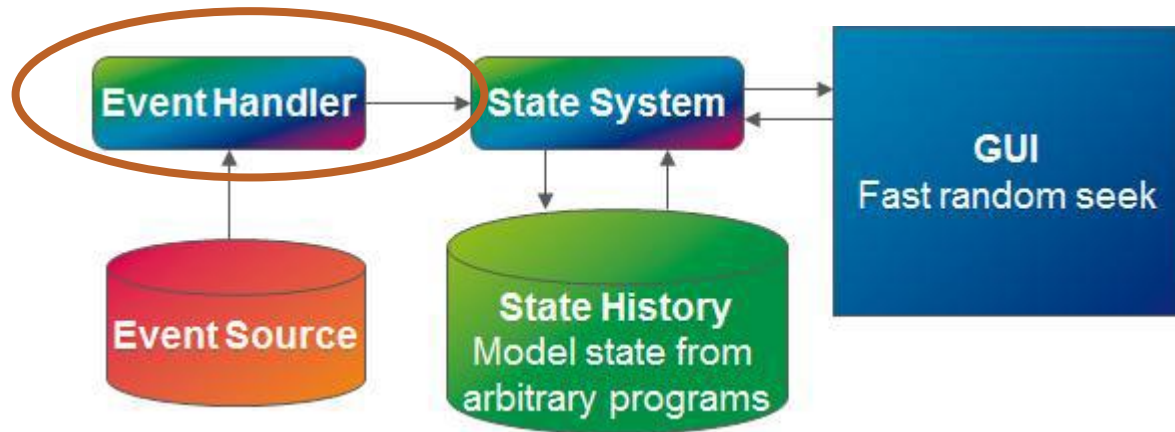
RESOURCEVIEW



- Display system resource states (color-coded) over time



DATA DRIVEN ANALYSIS



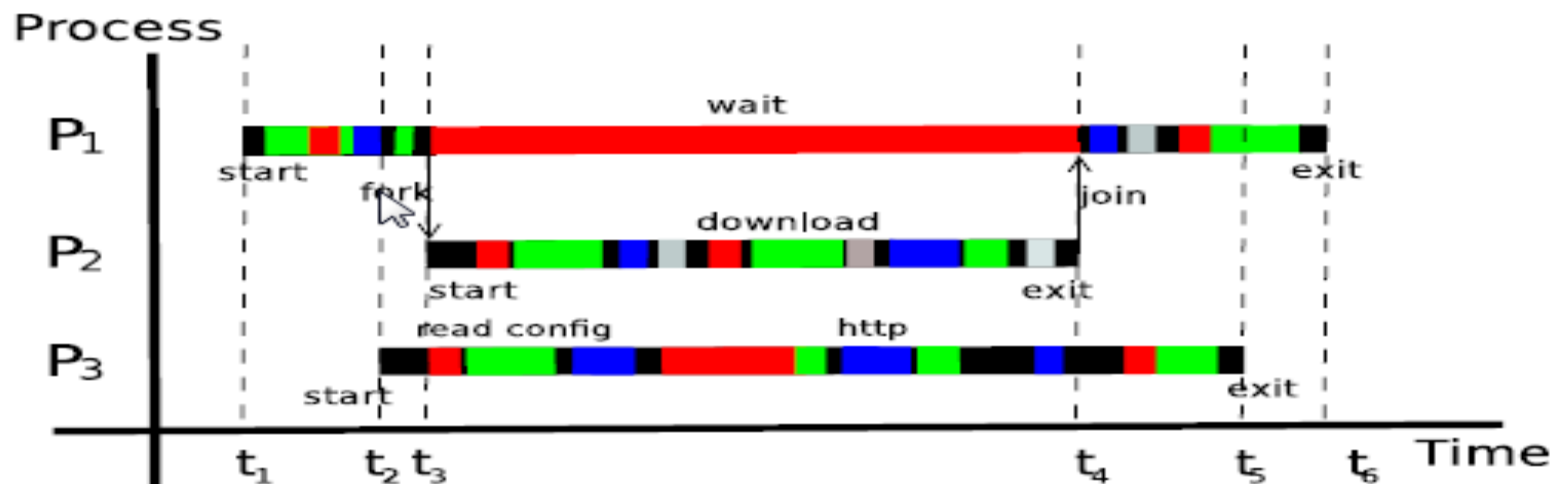
HOW TO USE TRACING?

- Learning
 - OS concepts teaching by LTTng traces
- Debugging
 - Program comprehension
 - Bug finding
 - Root-cause analysis
- Anomaly Detection
- Dependency Analysis
- Network Analysis



RESEARCH TRACKS

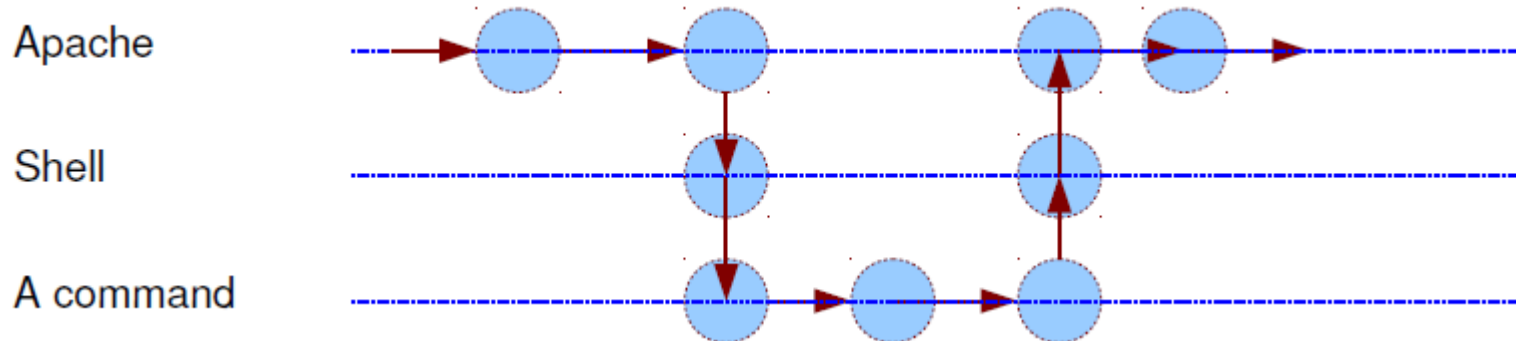
- Multi level trace analysis:
 - Multi Level Trace Abstraction
 - Metric based, Data driven, Visual abstraction, Resource abstraction
 - Stateful trace aggregation
 - Multi Level Trace Visualization
 - Label Placement
 - Statistics Framework (offline and online data)



RESEARCH TRACKS (2)

○ Automated Fault Identification

- Pattern Library
 - Various Attack Patterns
- Kernel Execution Path
 - example: apache, shell



File Edit View Search Terminal Help

```

apache2 (4707): Socket Accept - [syscallID= 43, start= 6164785038189, end= 6161630675366], Ret= 9
apache2 (4707): Get Socket Name - [syscallID= 51, start= 6161630689980, end= 6161630692919], Ret= 0
apache2 (4707): Get File Status - [syscallID= 4, start= 6161630778175, end= 6161630787262], Ret= 0
apache2 (4707): Set Value of an Interval Timer - [syscallID= 38, start= 6161630989131, end= 6161630994388], Ret= 0
apache2 (4707): Set Value of an Interval Timer - [syscallID= 38, start= 6161631288272, end= 6161631291791], Ret= 0
apache2 (4707): Get File Status - [syscallID= 5, start= 6161631388093, end= 6161631388776], Ret= 0
apache2 (4707): Open.Close File Operation - \var\www\test.php (6161631363661 - 6161631507683)
apache2 (4707): Create a Child Process - [syscallID= 56, start= 6161631571424, end= 6161631888041, PARENT_PID= 4707, CHILD_PID= 6782], Ret= 6782
apache2 (4707): Open.Close File Operation - pipe (6161631564786 - 6161631914508)
apache2 (6782): Open.Close File Operation - pipe (0 - 6161631990480)
chrome (1951): Recieve Data - [syscallID= 45, start= 6161631986540, end= 6161631990480, SOCK= 0xffff800007146300], Ret= 4
chrome (1951): Recieve Data - [syscallID= 45, start= 6161631991947, end= 6161631994520, SOCK= 0xffff800007146300], Ret= 4
apache2 (6782): Duplicate a FD - [syscallID= 33, start= 6161631991622, end= 6161631995134], Ret= 1

```

```

bin\sh (6782): Get File Status - [syscallID= 4, start= 6161633215760, end= 6161633218245], Ret= -2
bin\sh (6782): Get File Status - [syscallID= 4, start= 6161633219827, end= 6161633221180], Ret= 0
bin\sh (6782): Create a Child Process - [syscallID= 56, start= 6161633227804, end= 6161633276936, PARENT_PID= 6782, CHILD_PID= 6783], Ret= 6783

```

```

\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libacl.so.1 (832 bytes) (6161633947043 - 6161633984566)
\bin\ls (6783): Check User's Permissions for File - [syscallID= 21, start= 6161633991358, end= 61616340006094], Ret= -2
\bin\ls (6783): Get File Status - [syscallID= 5, start= 6161634005244, end= 6161634006094], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libc.so.6 (832 bytes) (6161634000432 - 6161634035794)
\bin\ls (6783): Check User's Permissions for File - [syscallID= 21, start= 6161634042953, end= 6161634045923], Ret= -2
\bin\ls (6783): Get File Status - [syscallID= 5, start= 6161634066025, end= 6161634066955], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libdl.so.2 (832 bytes) (6161634060803 - 6161634093178)
\bin\ls (6783): Check User's Permissions for File - [syscallID= 21, start= 6161634099782, end= 6161634102739], Ret= -2
\bin\ls (6783): Get File Status - [syscallID= 5, start= 6161634114753, end= 6161634115703], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libpthread.so.0 (832 bytes) (6161634109768 - 6161634155166)
\bin\ls (6783): Check User's Permissions for File - [syscallID= 21, start= 6161634162985, end= 6161634165867], Ret= -2
\bin\ls (6783): Get File Status - [syscallID= 5, start= 6161634177919, end= 6161634178808], Ret= 0
\bin\ls (6783): Sequential File Read - \lib\x86_64-linux-gnu\libattr.so.1 (832 bytes) (6161634172999 - 6161634204888)
\bin\ls (6783): Get File Status - [syscallID= 5, start= 6161634662387, end= 6161634663635], Ret= 0
\bin\ls (6783): Sequential File Read - \proc\filesystems (315 bytes) (6161634644712 - 6161634712957)
\bin\ls (6783): Open.Close File Operation - . (6161634833462 - 6161634864928)
\bin\ls (6783): Get File Status - [syscallID= 5, start= 6161634890756, end= 6161634891843], Ret= 0
\bin\ls (6783): Sequential File Write - pipe (16 bytes) (6161634913338 - 6161634916753)

```

```

\bin\ls (6783): Open.Close File Operation - pipe (0 - 6161634931119)
\bin\sh (6782): Wait for Process to Change State - [syscallID= 61, start= 6161633309590, end= 6161635084730, PID= 0], Ret= 6783
apache2 (4707): Sequential File Read - pipe (16 bytes) (6161631564785 - 6161635190274)
apache2 (4707): Wait for Process to Change State - [syscallID= 61, start= 6161635191059, end= 6161635196798, PID= 6782], Ret= 6782
apache2 (4707): Sequential File Read - \dev\urandom (8 bytes) (6161635412773 - 6161635429084)
apache2 (4707): Sequential File Read - \dev\urandom (8 bytes) (6161635433058 - 6161635446302)
apache2 (4707): Sequential File Read - \dev\urandom (8 bytes) (6161635449624 - 6161635462920)
apache2 (4707): Set Value of an Interval Timer - [syscallID= 38, start= 6161635464907, end= 6161635468712], Ret= 0
apache2 (4707): Get Process Times - [syscallID= 100, start= 6161635682919, end= 6161635684019], Ret= 4295549864
apache2 (4707): Get File Status - [syscallID= 4, start= 6161648146207, end= 6161648151184], Ret= -2
apache2 (4707): Get File Status - [syscallID= 6, start= 6161648153604, end= 6161648155576], Ret= 0
apache2 (4707): Get File Status - [syscallID= 6, start= 6161648156928, end= 6161648158391], Ret= 0
apache2 (4707): Get File Status - [syscallID= 6, start= 6161648162538, end= 6161648164345], Ret= -2
apache2 (4707): Open.Close File Operation - \var\www\ (6161648192332 - 6161648214782)
apache2 (4707): Get Process Times - [syscallID= 100, start= 6161648389334, end= 6161648390386], Ret= 4295549865

```


RESEARCH TRACKS (3)

- Dependency Analysis
 - System-level critical path analysis
- *Provide trace analysis tools to understand the overall performance of a distributed application.*

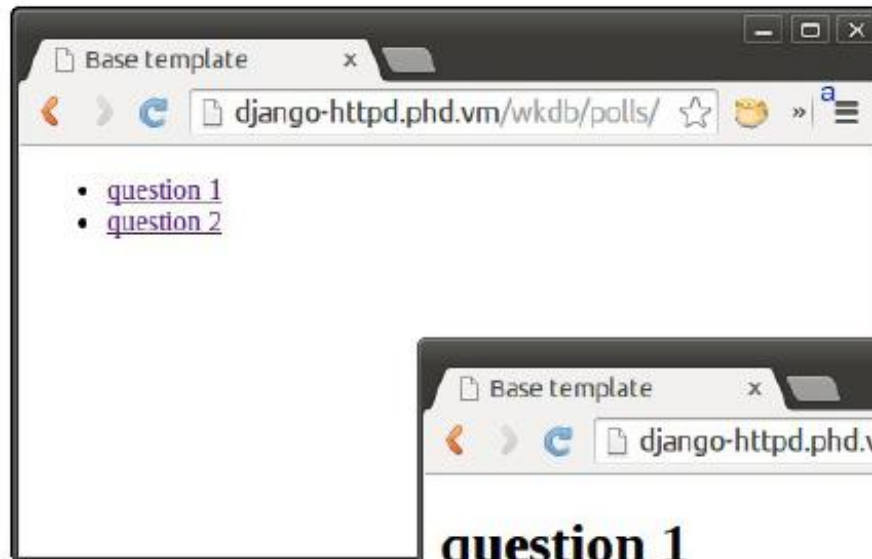


Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x0000000000001650	ld-2.17.so
99.87	0.00	1	0x000000000040a5a8	apt-get
99.87	0.00	1	(below main)	libc-2.17.so: libc-start.c
99.87	0.00	1	0x000000000040a0b0	apt-get
99.72	0.00	1	CommandLine:DispatchArg(CommandLin...	libapt-pkg.so.4.12.0
99.72	0.13	1	0x0000000000416ca0	apt-get
72.20	0.00	1	0x00000000004251e0	apt-get
63.28	0.00	1	pkgCacheFile::Open(OpProgress*, bool)	libapt-pkg.so.4.12.0
62.65	0.00	3	pkgCacheFile::BuildDepCache(OpProgress*)	libapt-pkg.so.4.12.0
62.65	0.51	1	pkgDepCache::Init(OpProgress*)	libapt-pkg.so.4.12.0
53.46	2.22	1	pkgDepCache::Update(OpProgress*)	libapt-pkg.so.4.12.0
41.23	3.66	609 598	pkgDepCache::DependencyState(pkgCac...	libapt-pkg.so.4.12.0
37.57	10.16	1 828 794	pkgDepCache::CheckDep(pkgCache::Depl...	libapt-pkg.so.4.12.0
26.49	2.24	1 130 829	debVersioningSystem::CheckDep(char con...	libapt-pkg.so.4.12.0

Traditional profiler output

What the app
is waiting for?

- 1) GET /wkdb/polls/1
- 2) POST vote
- 3) GET redirect

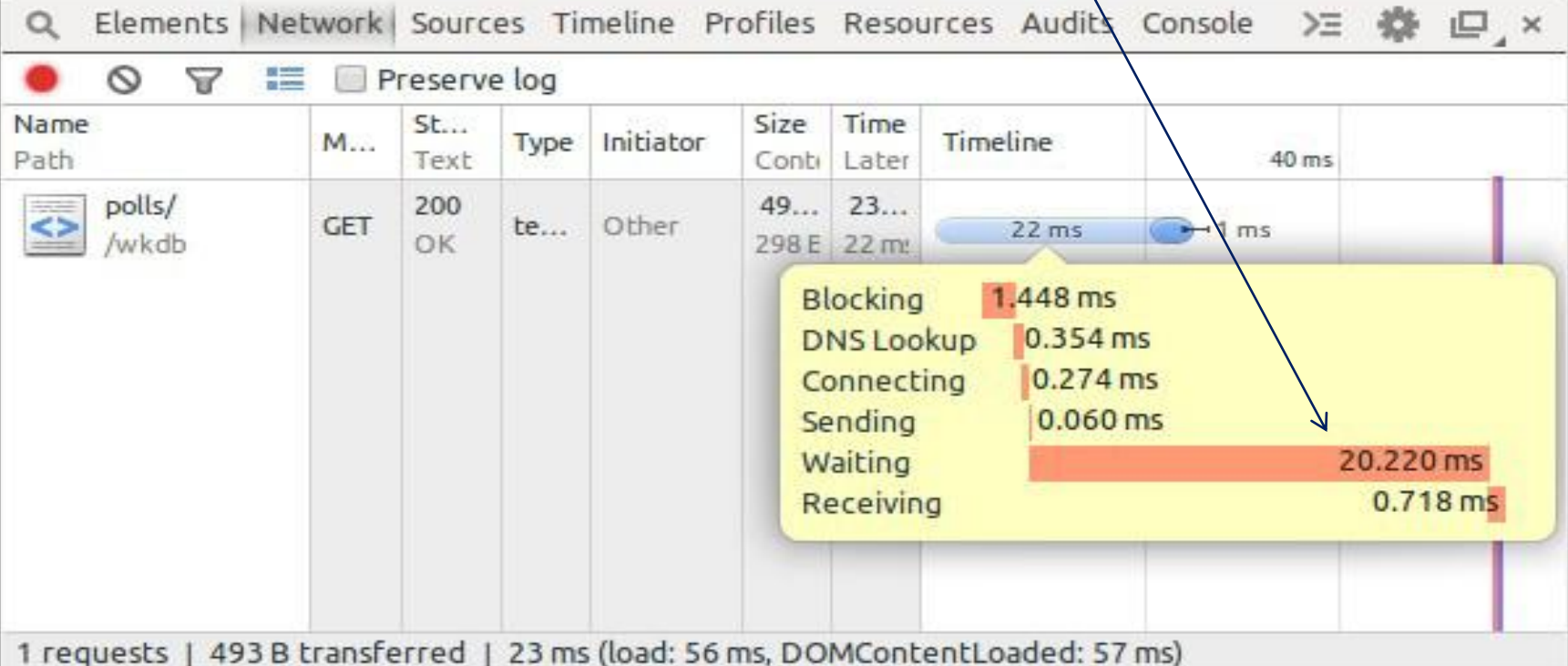


Base template

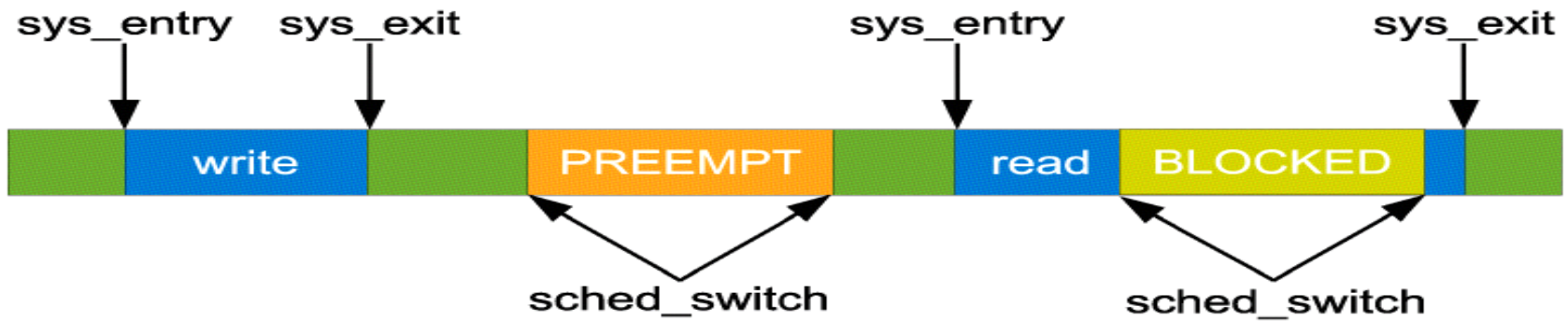
django-httpd.phd.vm/wkdb/polls/

- [question 1](#)
- [question 2](#)

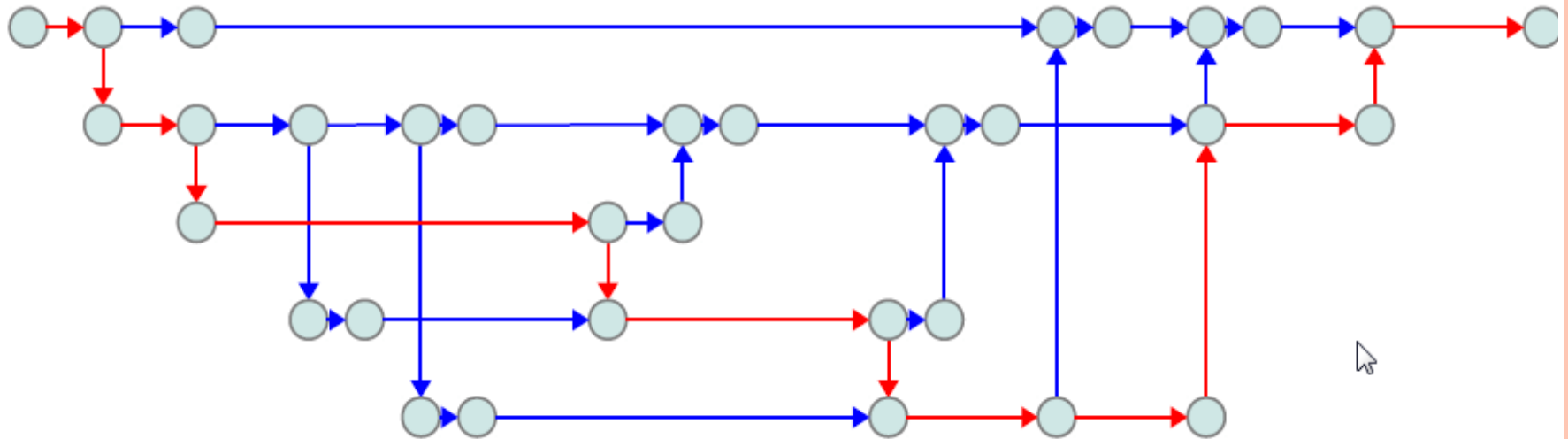
What is the server doing?



Task state



TASK_INTERRUPTIBLE
TASK_UNINTERRUPTIBLE



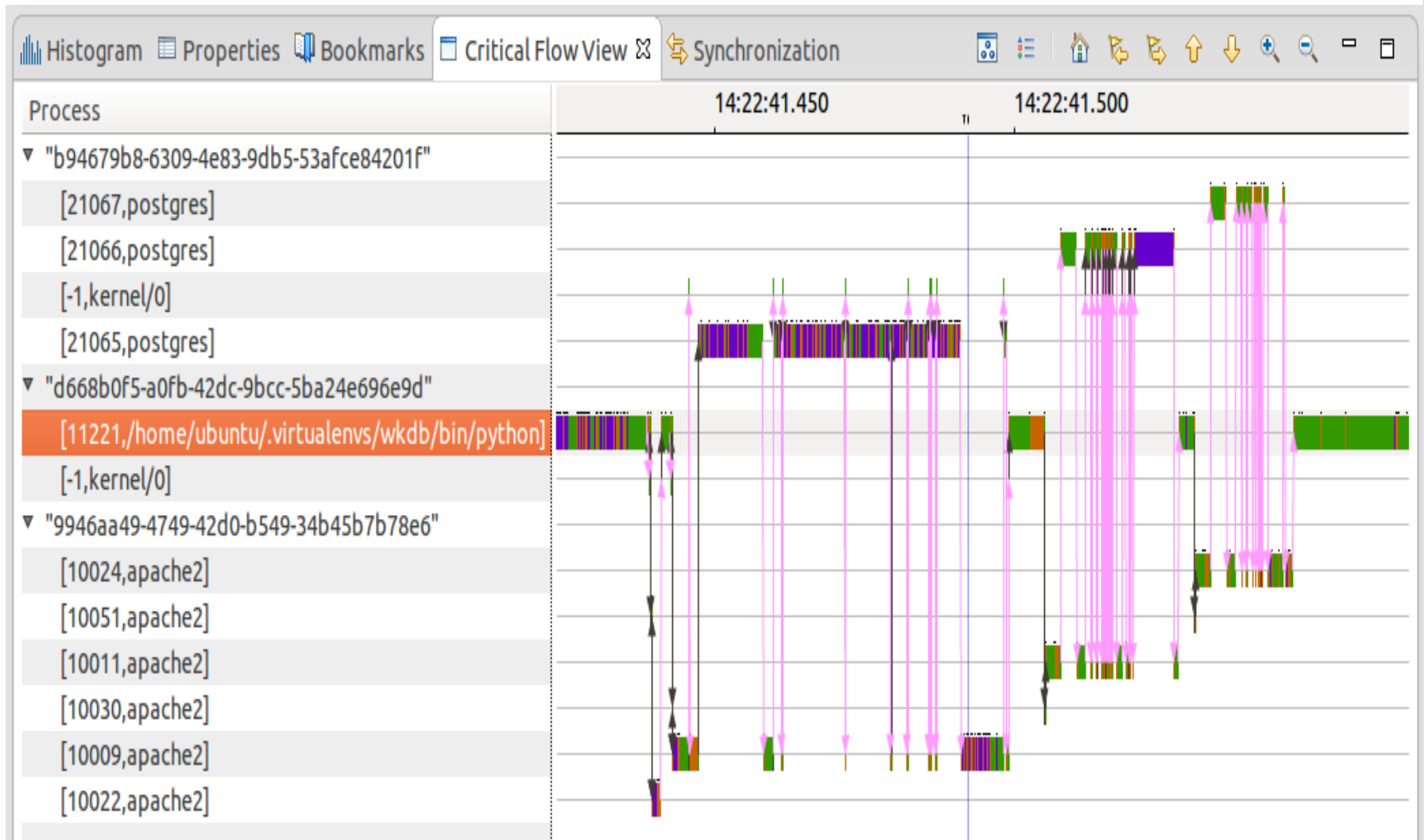
Device wake-up



Task wake-up



RESULT



RESEARCH TRACKS (3)

- Trace Compare: diagnose performance variations by comparing traces
 - Compare different parts of a trace
 - Compare different trace files
 - Normal or faulty execution
- Case1: a server has to read data from the disk to fulfill requests. At regular intervals, a request is about 5 times slower than usual.
- Case2: a client application generates data and inserts it into a MongoDB database (version 2.5.4). Most of the time, the whole operation takes around 10 ms. However, a fraction of the time, the operation takes more than 100 ms.
- Case 3: batch insert commands are sent to a MongoDB server. The commands are run in less than 700 μ s most of the time. However, about 1 in 10 000 commands takes between 3 and 5 seconds to complete.



RESEARCH TRACKS (4)

○ Data Driven Analysis

- Language
 - XML
- Aggregation
- Filtering
 - Debugging the patterns
- Program comprehension
- Attack detection:
 - SYN Flood Attack
- I/O latency analysis
 - Why some web requests take too long time?

```
$ while true; do time wget -q -p http://10.2.0.1/; sleep 0.5; done
...
real    0m0.017s
real    0m0.016s
real    0m0.454s
real    0m0.016s
real    0m0.015s
```



ONE MORE EXAMPLE

```
void workFor(int micros)
{
    clock_t start, end;
    start = clock();
    end = start + (micros / (CLOCKS_PER_SEC / 1000000));
    while (clock() < end) {};
}

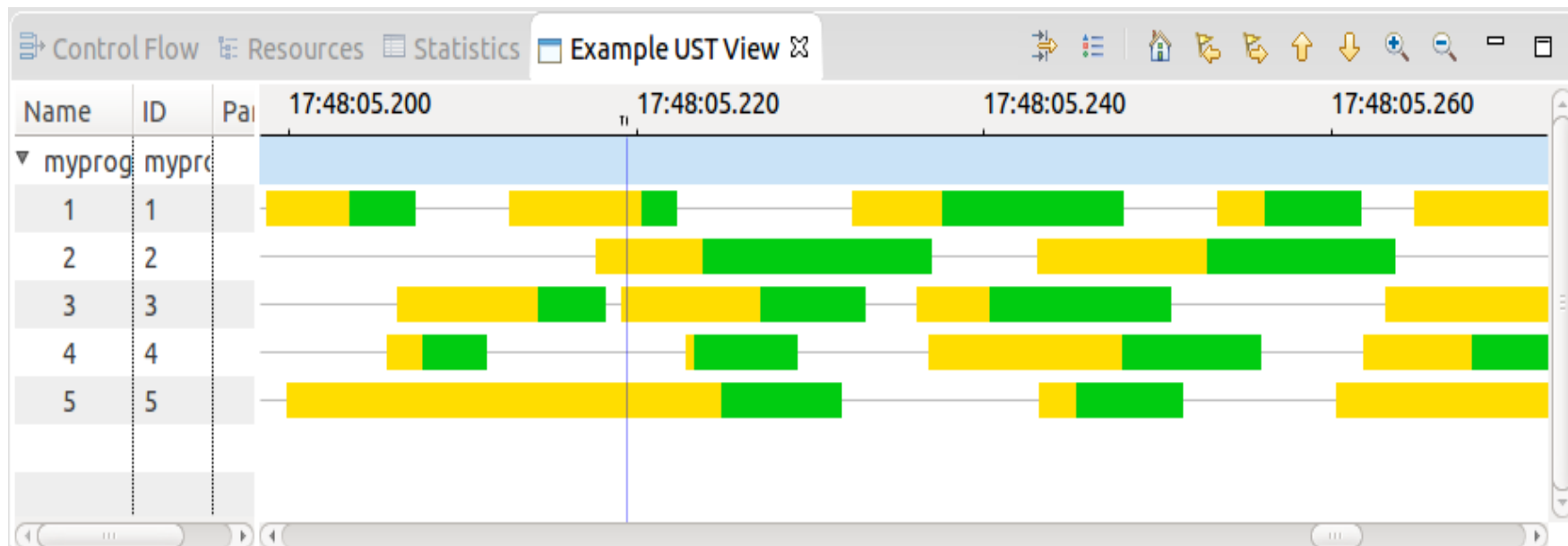
int main(int argc, char **argv)
{
    int nb_threads = 5;
    int nb_loops = 10;
    int i;
    srand(time(NULL));
    fprintf(stderr, "Tracing...\n");
    #pragma omp parallel private(i) num_threads(nb_threads)
    for (i = 0; i < nb_loops; i++) {
        int id = omp_get_thread_num() + 1;
        /* Loop starts here */
        workFor(rand() % 50000);
        //Connection attempted
        tracepoint(ust_myprog, connection_wait, id);

        workFor(rand() % 50000);
        //Connection is established
        tracepoint(ust_myprog, connection_start, id);

        workFor(rand() % 50000);
        //Connection ends
        tracepoint(ust_myprog, connection_end, id);
    }
    fprintf(stderr, "Done.\n");
    return 0;
}
```




```
<!-- Event handlers -->
<eventHandler eventName="ust_myprog:connection_wait">
    <stateChange>
        <stateAttribute type="constant" value="Threads" />
        <stateAttribute type="eventField" value="id" />
        <stateValue type="int" value="$STATE_CONNECTING" />
    </stateChange>
</eventHandler>
<eventHandler eventName="ust_myprog:connection_start">
    <stateChange>
        <stateAttribute type="constant" value="Threads" />
        <stateAttribute type="eventField" value="id" />
        <stateValue type="int" value="$STATE_ESTABLISHED" />
    </stateChange>
</eventHandler>
<eventHandler eventName="ust_myprog:connection_end">
    <stateChange>
        <stateAttribute type="constant" value="Threads" />
        <stateAttribute type="eventField" value="id" />
        <stateValue type="null" />
    </stateChange>
</eventHandler>
...
<!-- This is the definition of the time-graph view -->
<timeGraphView id="org.eclipse.linuxtools.tmf.analysis.xml.sstimeview">
...
    <definedValue name="STATE_CONNECTING" value="0" color="#FFDD00" />
    <definedValue name="STATE_ESTABLISHED" value="1" color="#00CC11" />
    <!-- Which attributes to "print" in the view -->
    <entry path="Threads/*">
        <display type="self" />
    </entry>
</timeGraphView>
```



myprog-trace

Timestamp	Channel	CPU	Event type	Contents
<srch>	<srch>	<srch>	<srch>	<srch>
17:48:05.214 424 109	channel0_3	3	ust_myprog:connection_start	id=3
17:48:05.217 732 378	channel0_0	0	ust_myprog:connection_wait	id=2
17:48:05.218 155 342	channel0_3	3	ust_myprog:connection_end	id=3
17:48:05.219 208 573	channel0_3	3	ust_myprog:connection_wait	id=3
17:48:05.220 341 267	channel0_2	2	ust_myprog:connection_start	id=1
17:48:05.222 290 003	channel0_2	2	ust_myprog:connection_end	id=1

CONCLUSION

- Many problems can only be studied live, in production.
- Tracing is a great solution.
- LTTng and TraceCompass provide an excellent platform to build advanced analysis.



THANK YOU

Downloads:

- LTTng: www.lttng.org
- TraceCompass: www.tracecompass.org
- Publications: www.dorsal.polymtl.ca

Me:

- email: n.ezzati@polymtl.ca, ezzati@gmail.com

