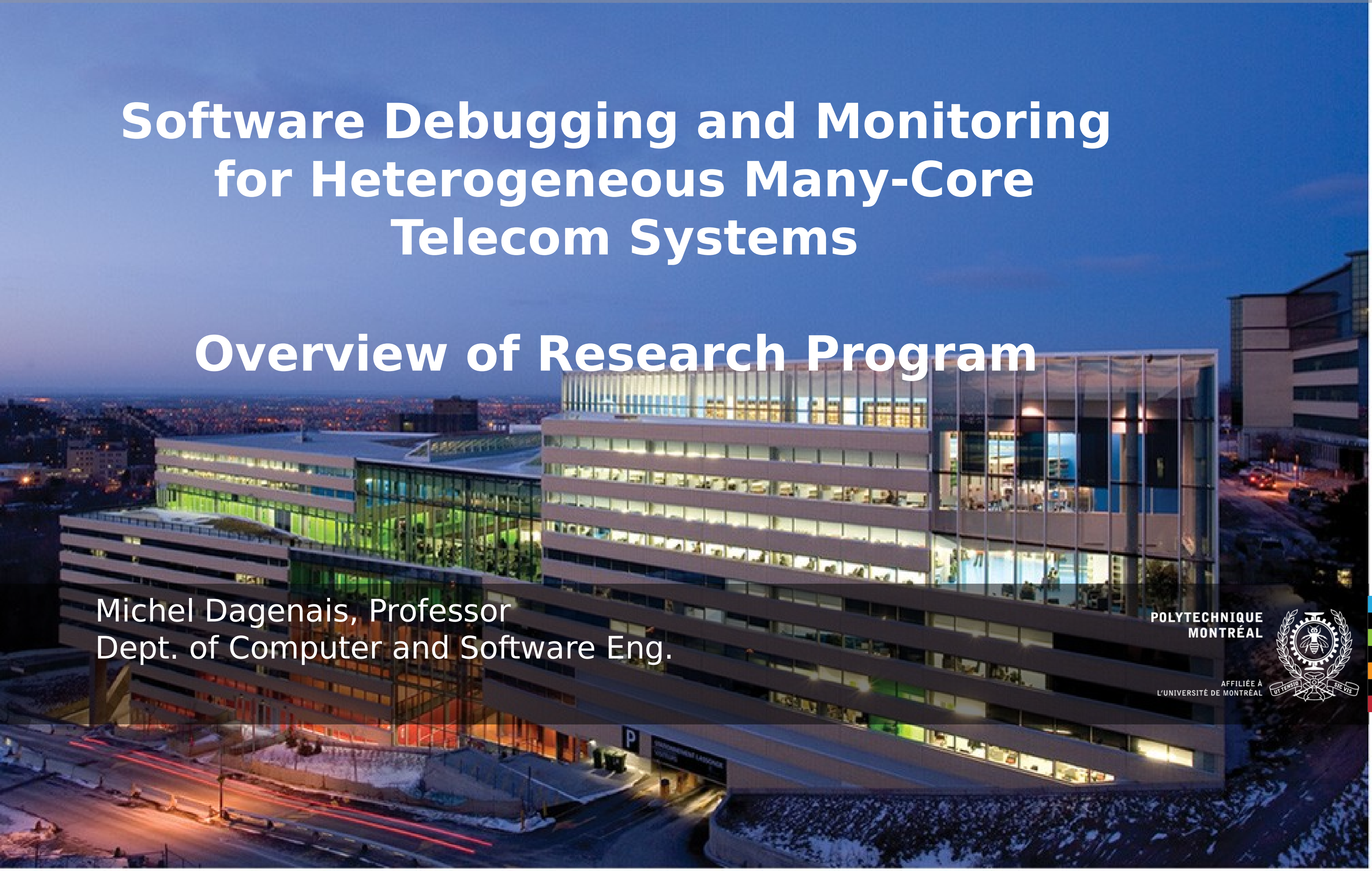


Software Debugging and Monitoring for Heterogeneous Many-Core Telecom Systems

Overview of Research Program

Michel Dagenais, Professor
Dept. of Computer and Software Eng.

POLYTECHNIQUE MONTRÉAL
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

Objectives

- Smart clients everywhere, always online cloud-based services accessed transparently.
- Complex heterogeneous many-core systems.
- Crucial need for better tools!

The objective of this project is to provide better tools that can be used in production, under real load, for the debugging, performance analysis and monitoring of distributed heterogeneous many-core systems. We need to extract and collect information with low overhead, minimize the perturbation on the systems being studied, and efficiently analyse online the gigabytes of data produced.

Projects Evolution

TDMC Feb. 2009-2012

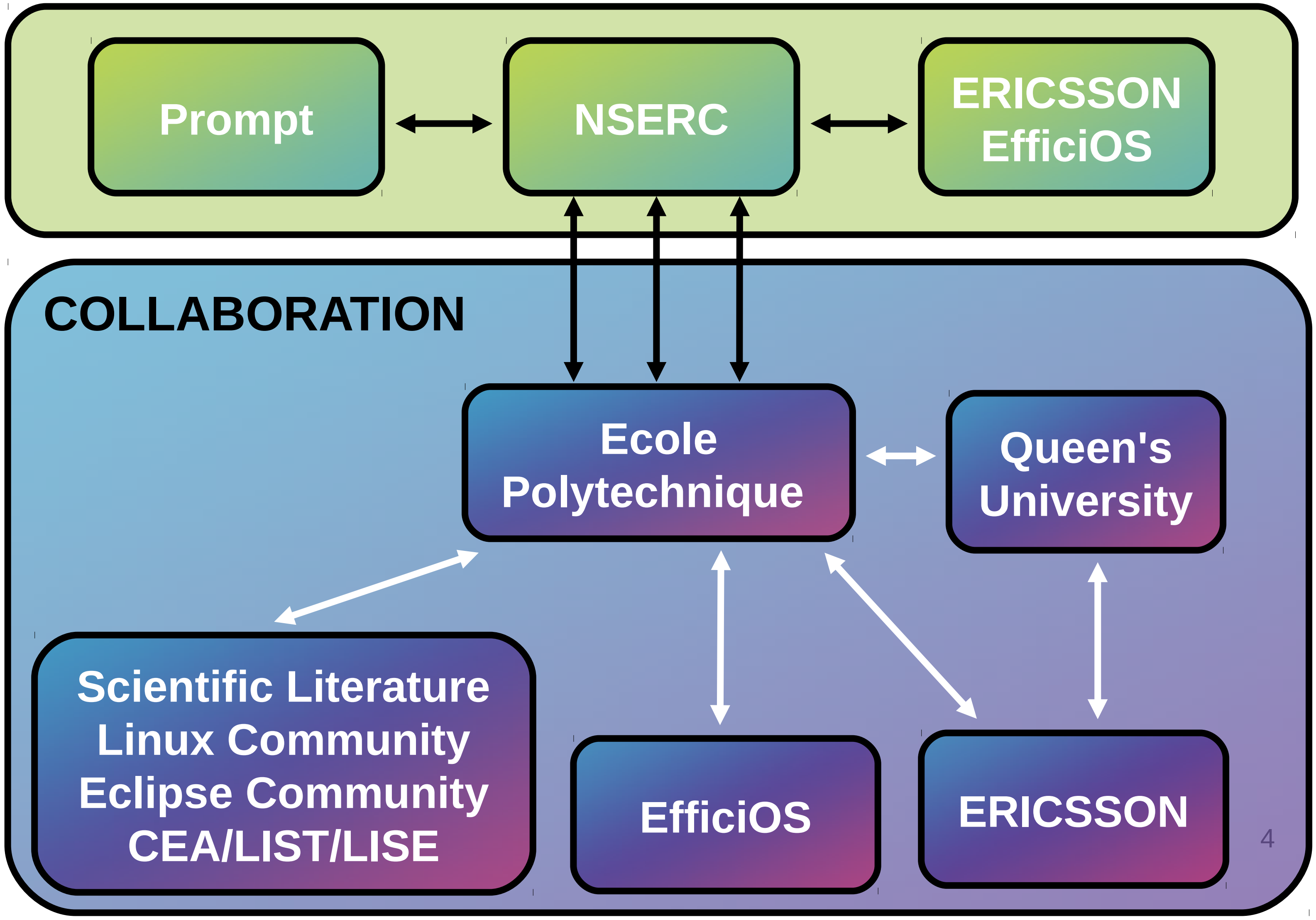
AHLS Feb. 2012-2015

CTPD Sep. 2012-2015

DMHMC Jan. 2015-2018

- TDMC: Ericsson, DRDC, NSERC; static tracepoints for kernel and user-space tracing, multi-session tracing, a posteriori traces synchronization, state system.
- AHLS: Ericsson, DRDC, NSERC; live tracing and synchronization, user-definable state system and views (e.g. Windows), store multiple state subtrees in state system.
- CTPD: Ericsson, EfficiOS, NSERC, Prompt; tracing on Tilera, ARM, Xeon Phi, and GPUs, dynamic tracepoints, VM tracing and view, RPC Debugging, critical path.
- HSDM: Ericsson, EfficiOS, NSERC, Prompt; models (e.g., UML, MARTE) tracing and analysis, Kalray, Adapteva and Intel PT tracing, Cloud and Software Defined Networks tracing, specialized analysis for OpenCL and system resources (I/O, memory, power), tolerating unknown states.

Structure of Project



Project funding = 4 x industrial contribution

Modus operandi

- Problems identification and prioritisation with the industrial partners.
- M.Sc., Ph.D. and PostDoc students focus on these difficult applied research problems with input from industry.
- Research associates help the graduate students to integrate their new proposed algorithms in the toolchain for validation and optimisation.
- The best algorithms are added to the LTTng and Trace Compass toolchain with support from industrial partner's R&D engineers.
- LTTng is redistributed by most Linux vendors: Wind River, MontaVista, Linaro, LTSI, Debian, Ubuntu, Suse, Fedora/Red Hat... Code contributions were received from more than 100 individuals and 25 companies including Ericsson, Google, IBM, Red Hat, Ubuntu, Mentor...

T1: Tracing, Debugging and Profiling Mechanisms and Architecture on Many-Core Systems

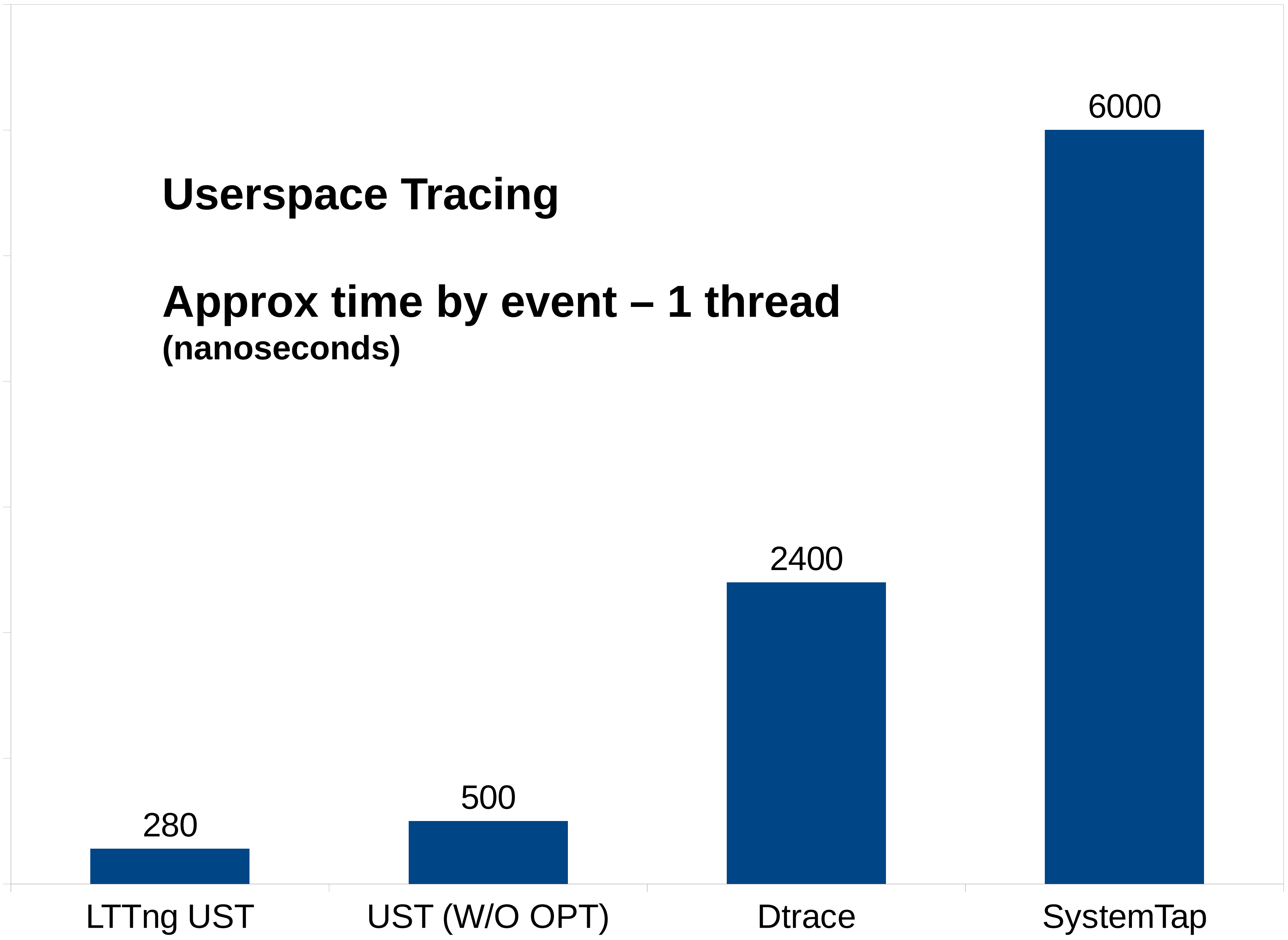
Objective: develop common efficient mechanisms for tracing and debugging heterogeneous many-core systems.

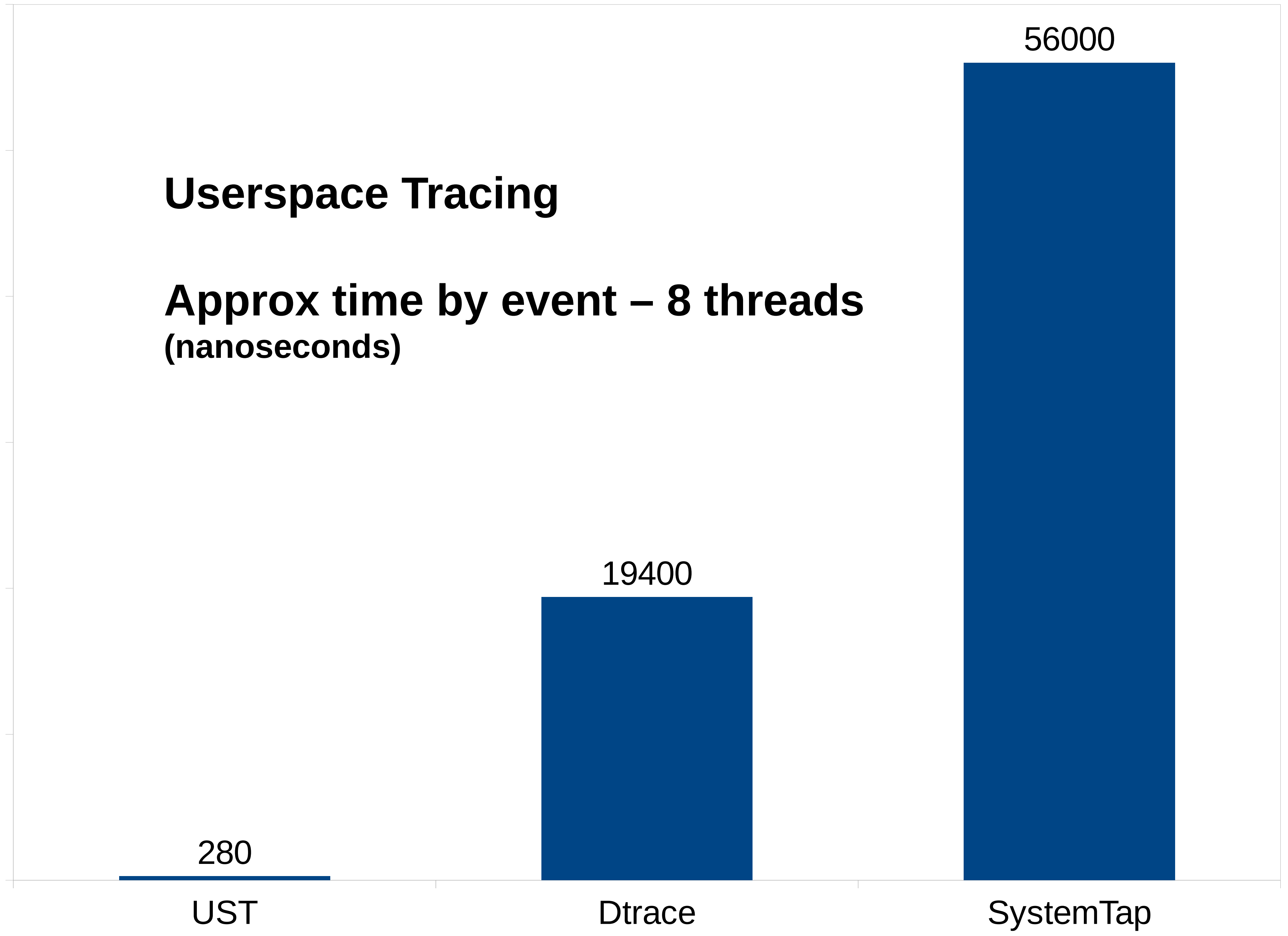
- T1M1: tracing new architectures (Adapteva Epiphany, Kalray MPPA).
- T1M2: tracing new architectures (ARM CoreSight, Intel PT, bare-metal processors).
- T1D1: efficient algorithms to control debugging, profiling and tracing on many-core processors and to retrieve data.
- T1D2: scalable tracing / debugging mechanisms for dynamic instrumentation, checking conditions, aggregating values…

Key: low disturbance, low overhead, no impact on system, dynamically adjust the overhead / level of details compromise

Userspace Tracing

Approx time by event – 1 thread
(nanoseconds)




Userspace Tracing
Approx time by event – 8 threads
(nanoseconds)
280
19400
56000
UST
Dtrace
SystemTap

Trace collection performance

- User-space tracing in user space, no system call.
- “Unlikely if” for tracepoint activation.
- Efficient binary format being optimised and standardized as the Multi-Core Association Common Trace Format. No formatting!
- Per CPU buffers with local lockless atomic operations.
- Read Copy Update (RCU) synchronisation for configuration information (tracepoint activation, multiple sessions...).
- Zero-copy trace recording on disk.
- Tracepoints may be inserted even in interrupt and NMI contexts.
- Efficient timestamping (rdtsc) even from virtual machines.
- Most efficient and flexible tracer available!

Latency-tracker

- Kernel module to track down latency problems at run-time.
- Simple API that can be called from anywhere in the kernel (tracepoints, kprobes, netfilter hooks, hardcoded in other module or the kernel tree source code).
- Keep track of entry/exit events and calls a callback (e.g., to dump stack or take a trace snapshot) if the delay between the two events is higher than a threshold, or no exit before timeout.

```
latency_tracker_event_in(tracker, key,
threshold, timeout, callback);
....
latency_tracker_event_out(tracker, key);
```

Sample usage

- Block layer latency
  - Delay between block request issue and complete
- Wake-up latency
  - Delay between sched_wakeup and sched_switch
- Network latency
- IRQ latency
- System call latency
  - Delay between the entry and exit of a system call
- Offcpu latency
  - How long a process has been scheduled out

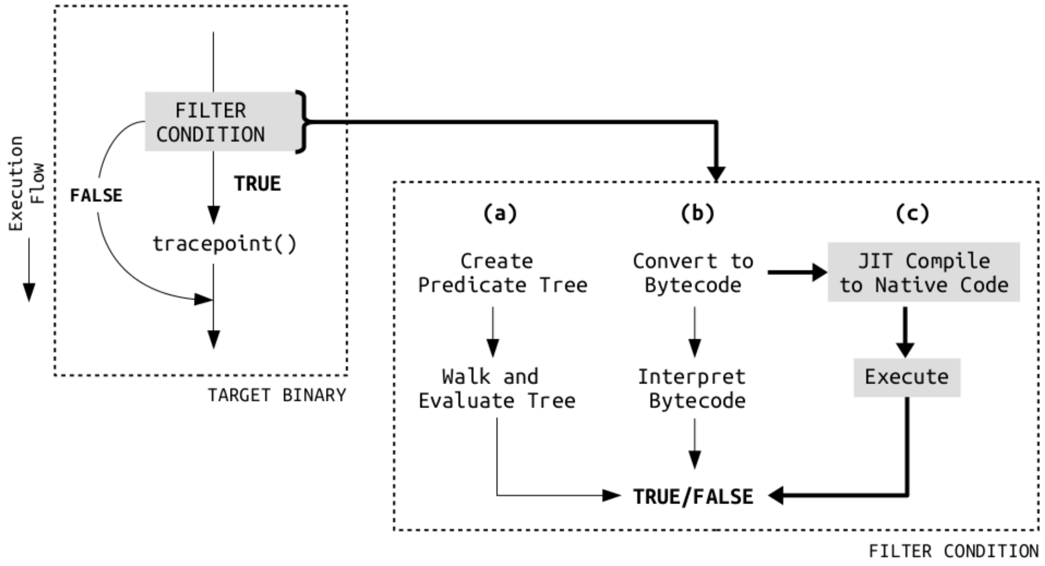
```
on sched_switch:
      event = latency_tracker_get_event(next_pid);
      if event && ((now – event->start) > threshold):
         dump_stack(next_pid);
```

Tracker performance

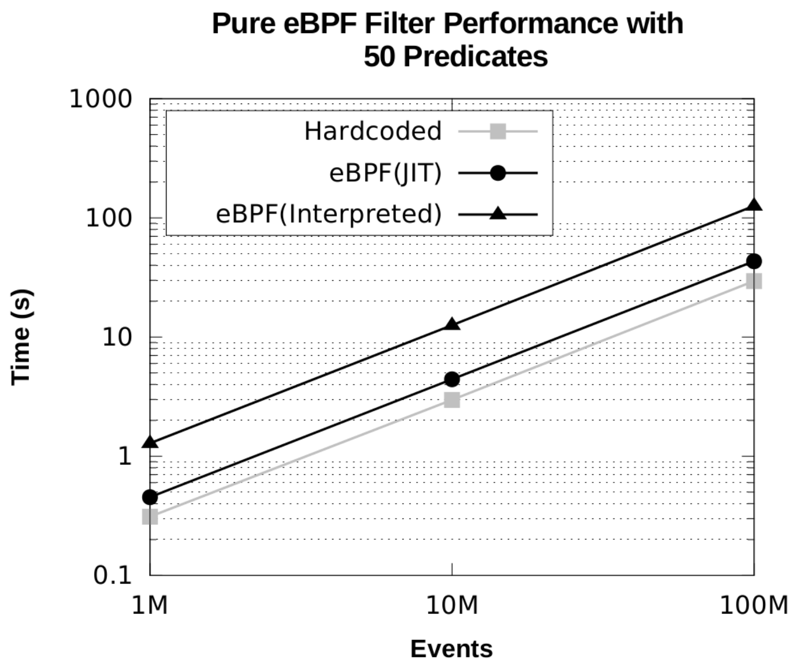
- Controlled memory allocation.
- Lockless free-list.
- Out-of-context reallocation of memory if needed/enabled.
- Using userspace-rcu hashtable (kernel version) for lockless insert and lookup.
- Custom call_rcu thread to avoid the variable side-effects of the builtin one.

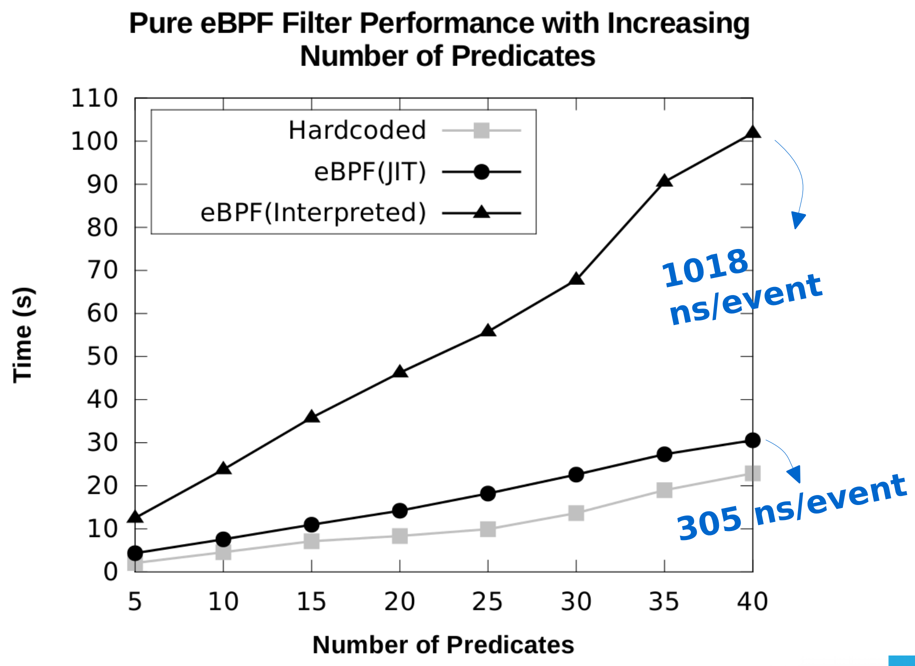
Dynamic instrumentation

- When we cannot recompile or restart program.
- When dynamic decisions can help optimise tracing.
- Comparison of different kernel and user-space solutions, using bytecode and JIT compilation.



Using eBPF bytecode or JIT for tracing





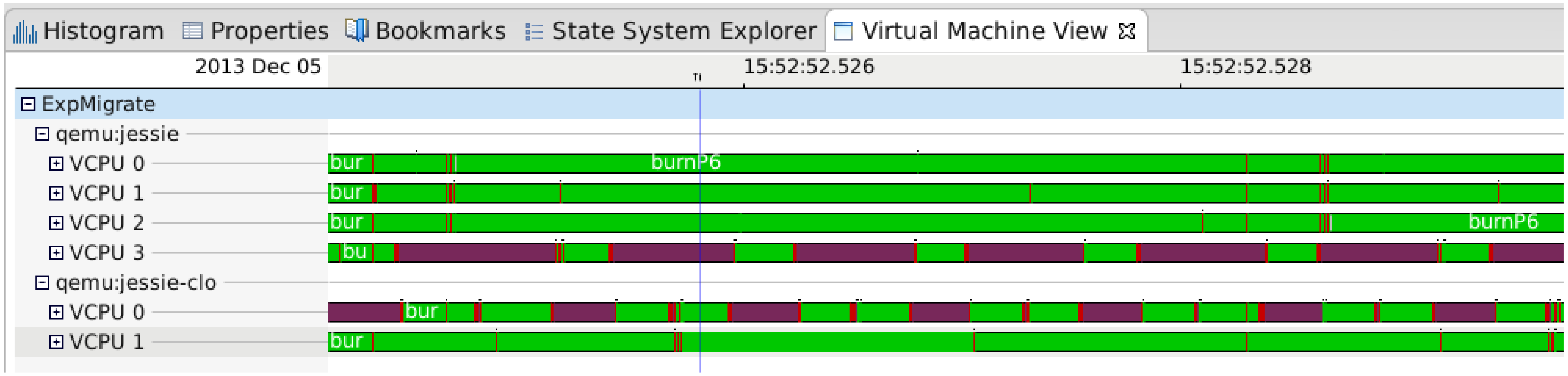
T2: Cloud Debugging and Monitoring

Objective: provide efficient tools to monitor and understand the performance of clouds and cloud-based applications.

- T2M1: data collection, analysis and monitoring for clouds (OpenStack).
- T2M2: data collection, analysis and monitoring for Software Defined Networks (OpenDaylight).
- T2D1: monitoring of migrating virtual machines in the cloud, with ability to navigate through information at multiple levels.

Key: combine information from multiple levels, maintain scalability through sampling and varying levels of details.

Real Performance of Virtual Machines



T3: Advanced Analysis

Objective: propose a number of new advanced analysis and views, in particular for specialized parallel processors.

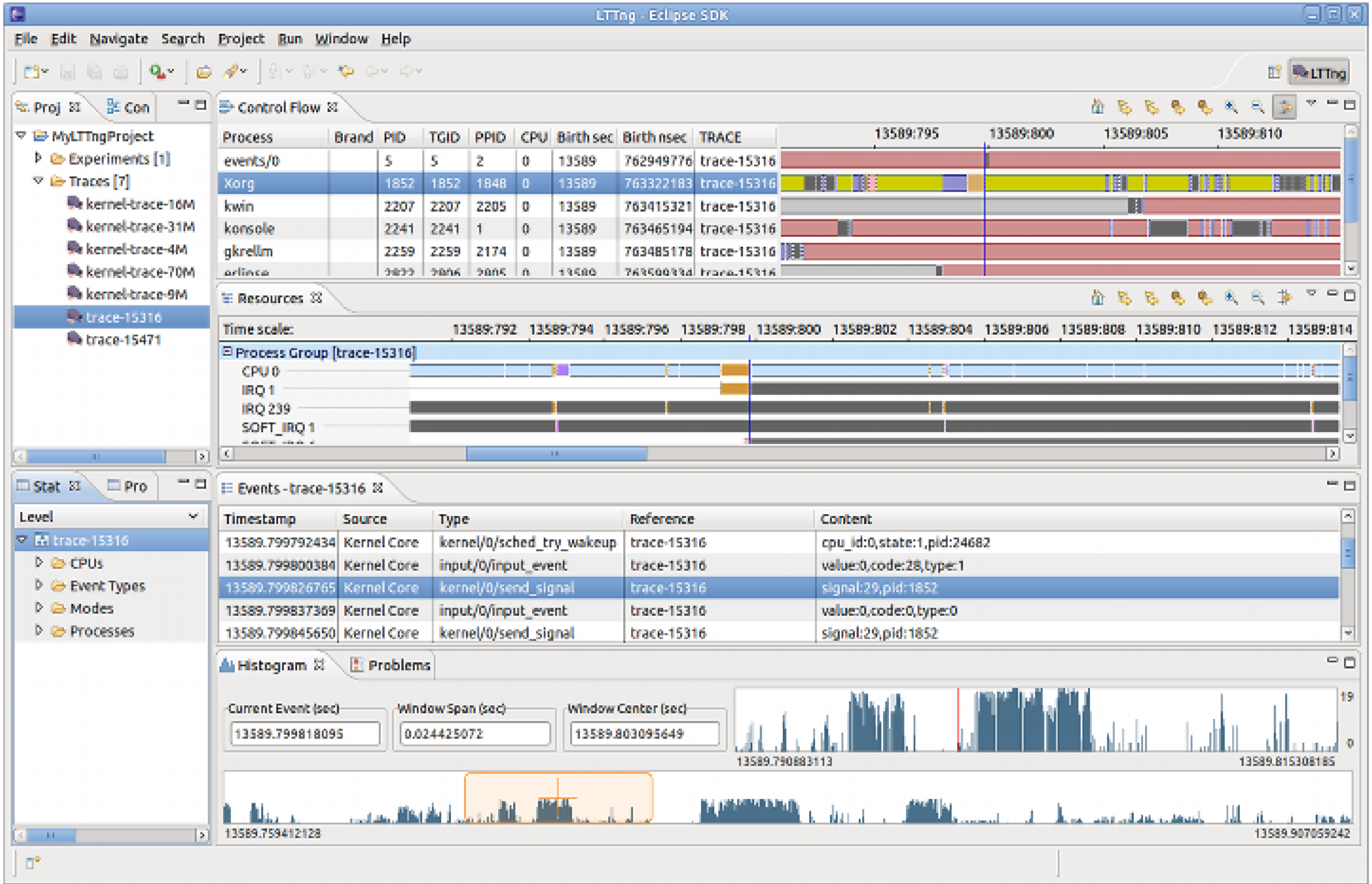
- T3M1: specialized programming models such as OpenCL, OpenMP and MPI.
- T3M2: specialized analysis and views for tasks such as virtual / physical memory usage, and power consumption.
- T3D1: sophisticated pattern matching on the state history and tracing events.
- T3D2: trace correlation for regression testing, highlighting the differences between similar runs of different versions, or in different contexts.

Key: tackle huge traces efficiently, help the user model the system and automate problem identification, provide contextualized analysis.

Trace Analysis

- Present events from different traces on a common reference time.
- Build a model of the traced system (process table, opened file descriptors...).
- Allow easy and efficient navigation and zooming in the trace.
- Present statistics and histograms for different intervals in the trace.
- Help identify the dependencies among events and trace the critical path.
- Traces may contain hundreds of GBs of data!

Eclipse Linux Tools



Process	Brand	PID	TGID	PPID	CPU	Birth sec	Birth nsec	TRACE
events/0		5	5	2	0	13589	762949776	trace-15316
Xorg		1852	1852	1848	0	13589	763322183	trace-15316
kwin		2207	2207	2205	0	13589	763415321	trace-15316
konsole		2241	2241	1	0	13589	763465194	trace-15316
gkrellm		2259	2259	2174	0	13589	763485178	trace-15316
eclipse		2822	2906	2905	0	13589	763599334	trace-15316

Timestamp	Source	Type	Reference	Content
13589.799792434	Kernel Core	kernel/0/sched_try_wakeup	trace-15316	cpu_id:0,state:1,pid:24682
13589.799800384	Kernel Core	input/0/input_event	trace-15316	value:0,code:28,type:1
13589.799826765	Kernel Core	kernel/0/send_signal	trace-15316	signal:29,pid:1852
13589.799837369	Kernel Core	input/0/input_event	trace-15316	value:0,code:0,type:0
13589.799845650	Kernel Core	kernel/0/send_signal	trace-15316	signal:29,pid:1852

Current Event (sec): 13589.799818095

Window Span (sec): 0.024425072

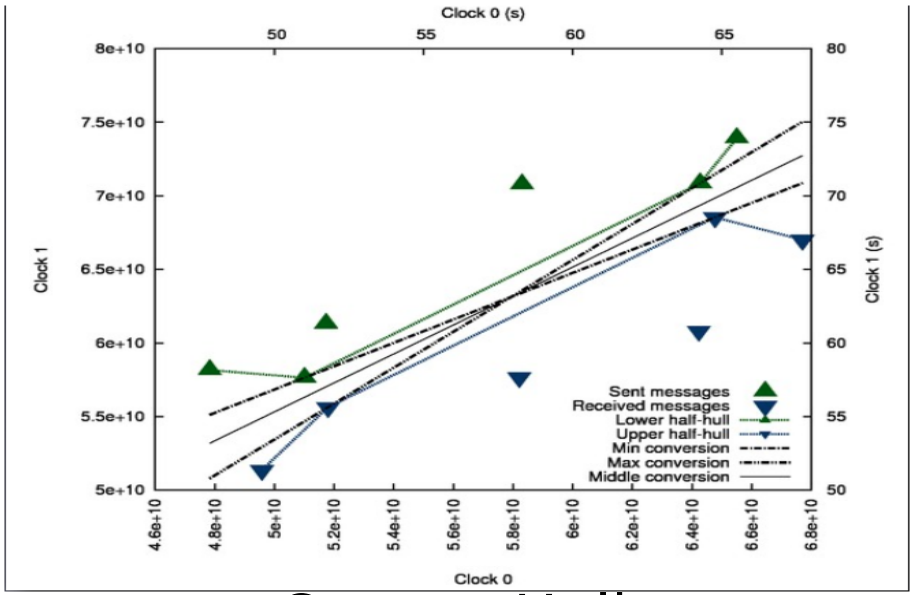
Window Center (sec): 13589.803095649

Distributed Traces Synchronization

- Each node, VM or heterogeneous processor may have its own independent clock, yet correlating events from different traces on a common time reference is needed.
- First order effect: different initial time and clock frequency.
- Second order effects: clock jitter, latency to read clock value, clock frequency drift with time.
- Identify corresponding send and receive events for packets to estimate clock differences.

Synchronisation

- New linear incremental algorithm to compute the clock offset and drift between two traces.
- Send and received events are matched by unique id (e.g., TCP sequence number for a given connection) and are incrementally added to build the upper and lower convex hull bounds.



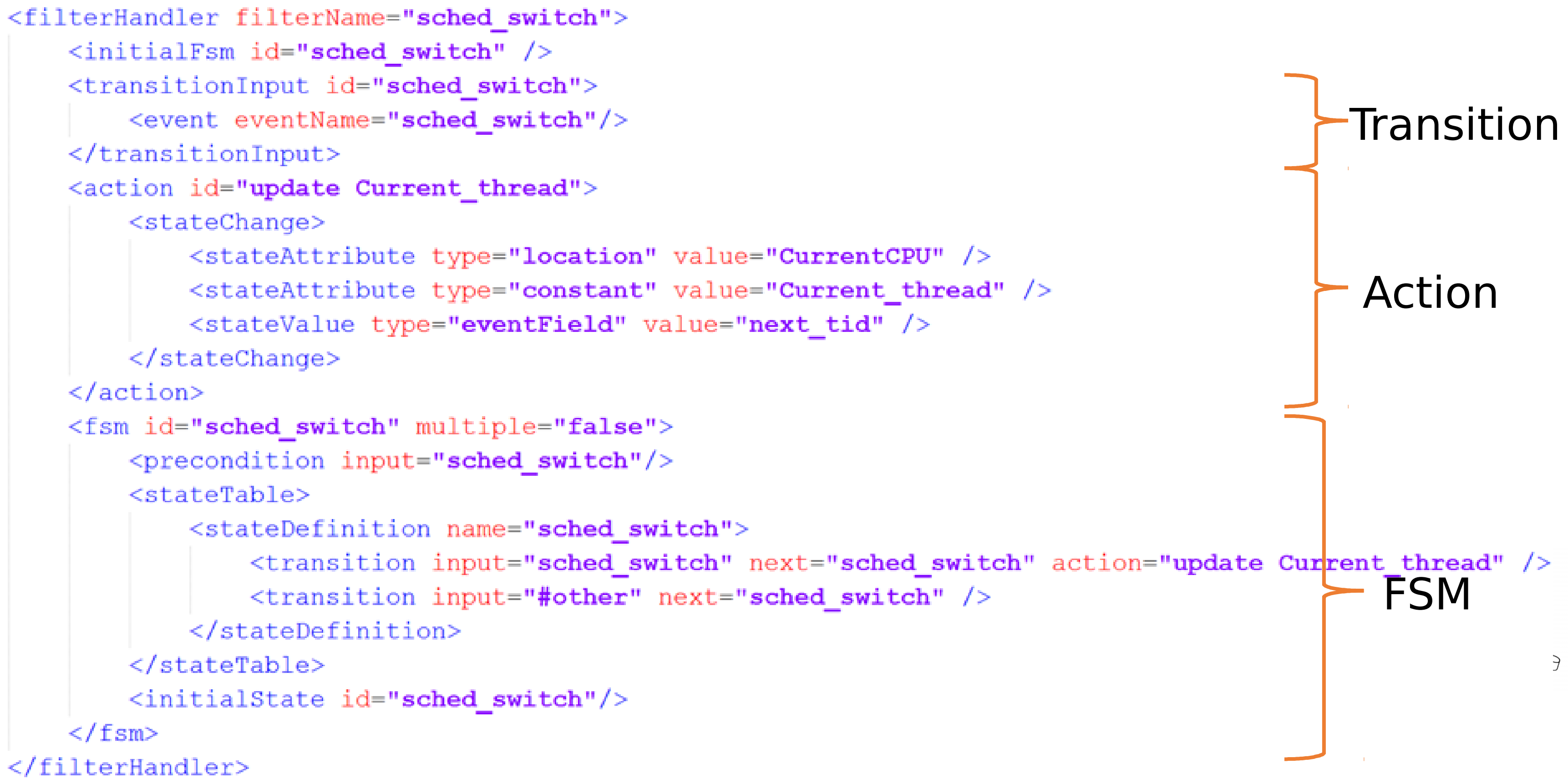
Convex Hull

Modeled State

- Current State Tree expressed as a generic attribute tree.
- For the Linux kernel, the Current State Tree contains: process table, file descriptors, sockets...
- Some events cause a state change (based on event type, event fields, current state).
- State builder expressed as custom java code or declarative XML expression.
- Efficient State History Tree database for quickly navigating through traces.
- The same declarative XML expressions can be used to define views, filters, abstract events and metrics.

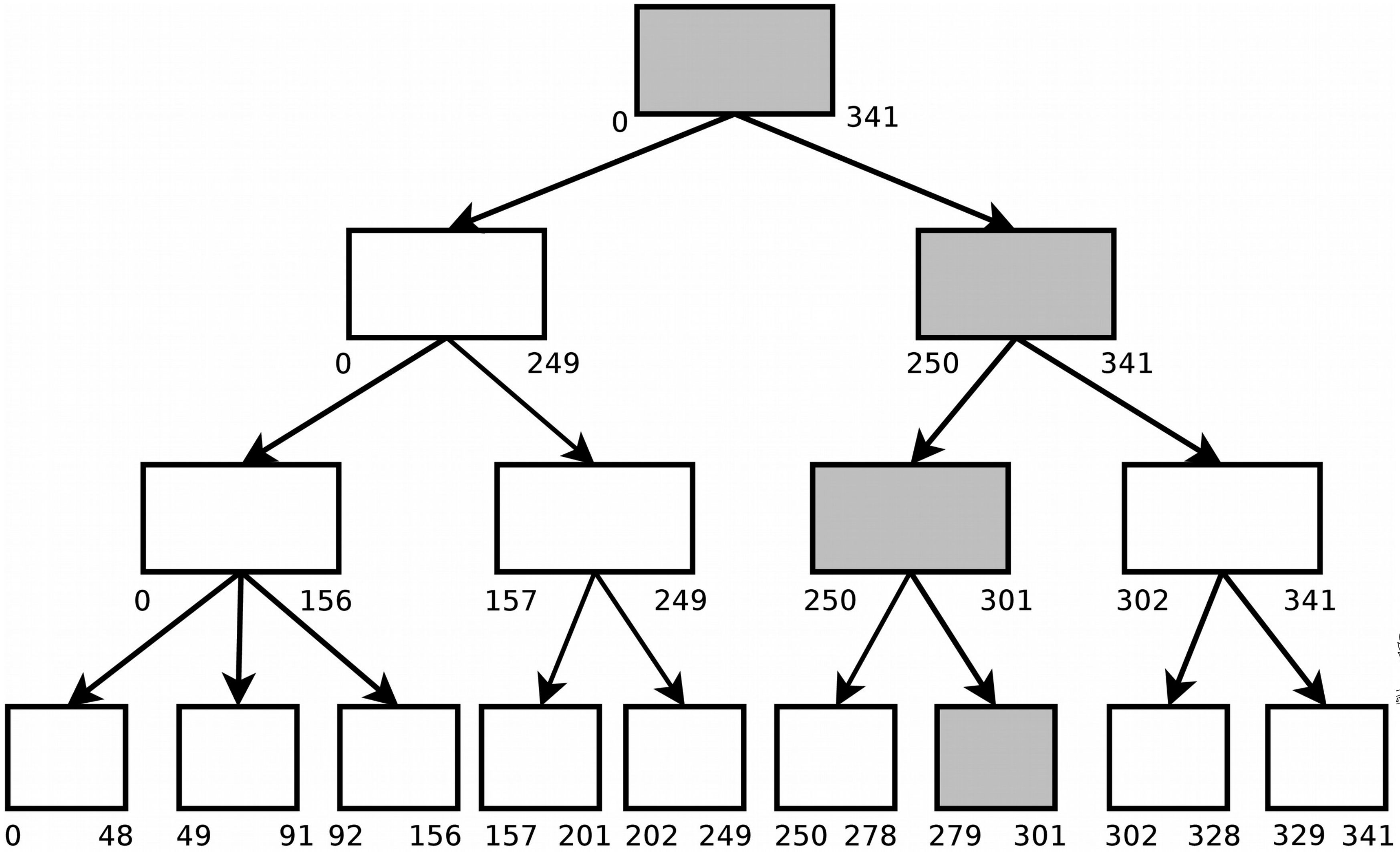
Declarative XML expression example

XML structure



```xml
<filterHandler filterName="sched_switch">
    <initialFsm id="sched_switch" />
    <transitionInput id="sched_switch">
        <event eventName="sched_switch"/>
    </transitionInput>
    <action id="update Current_thread">
        <stateChange>
            <stateAttribute type="location" value="CurrentCPU" />
            <stateAttribute type="constant" value="Current_thread" />
            <stateValue type="eventField" value="next_tid" />
        </stateChange>
    </action>
    <fsm id="sched_switch" multiple="false">
        <precondition input="sched_switch"/>
        <stateTable>
            <stateDefinition name="sched_switch">
                <transition input="sched_switch" next="sched_switch" action="update Current_thread" />
                <transition input="#other" next="sched_switch" />
            </stateDefinition>
        </stateTable>
        <initialState id="sched_switch"/>
    </fsm>
</filterHandler>
```

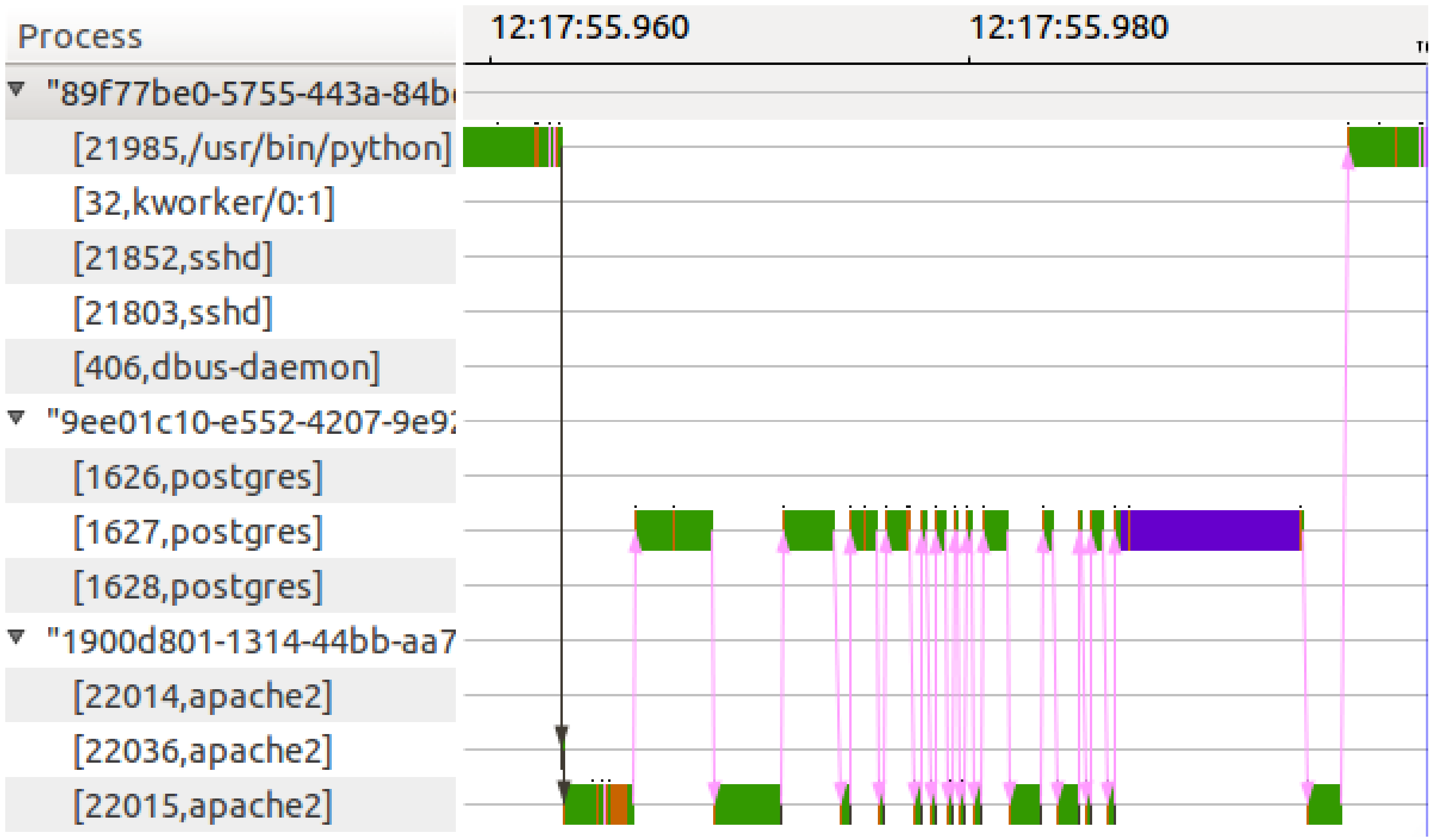
State History Tree



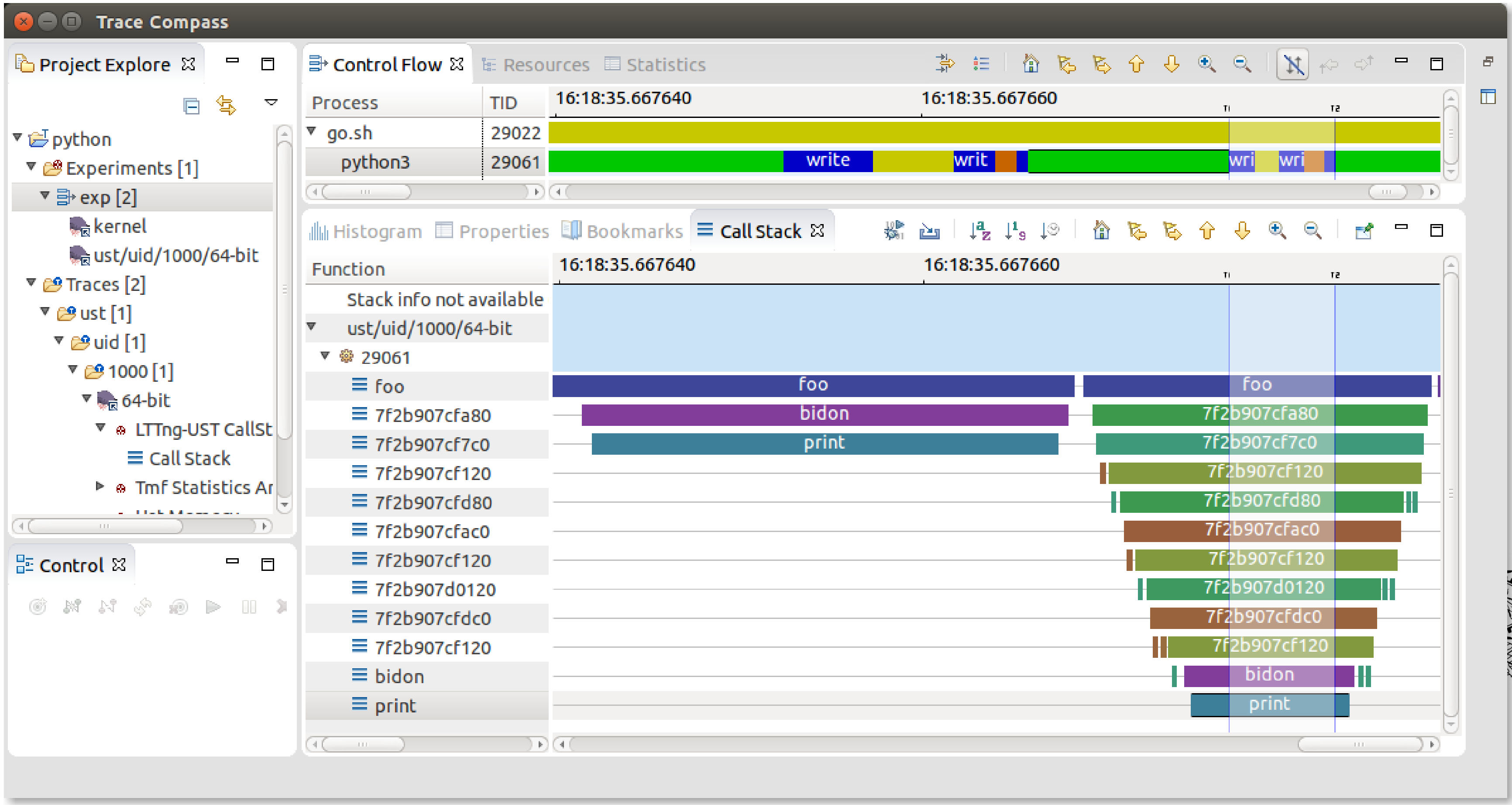
Critical Path Analysis

- For each process, the elapsed time is broken down by state: running, waiting for resource (file, pipe, socket, CPU) or sleeping on a timer.
- The critical path for reaching a point in a process is the chain of unblocking events (wakeup) and their origin (device ready, lock freed or packet received from another process).
- Find and display the critical path with a breakdown of elapsed time per process and by state (e.g. web request took 3s of Firefox, 2s of X11, 5s of Apache, 3s of MySQL).

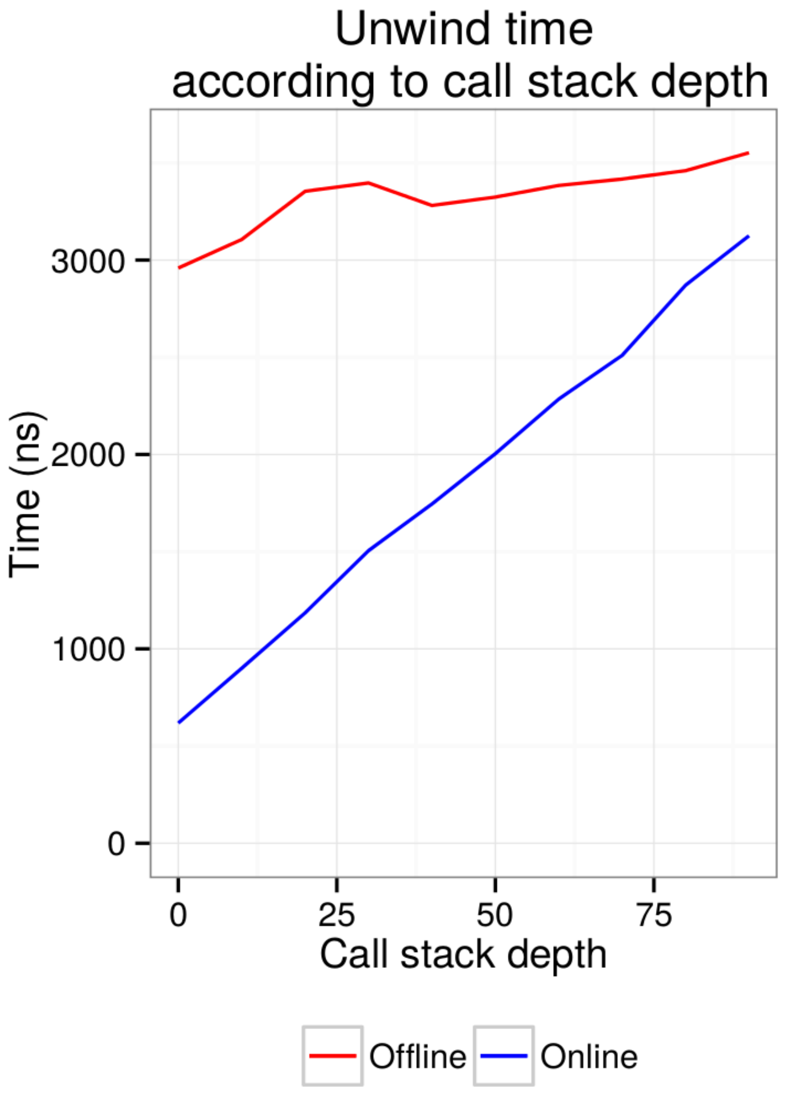
Critical Path Analysis view

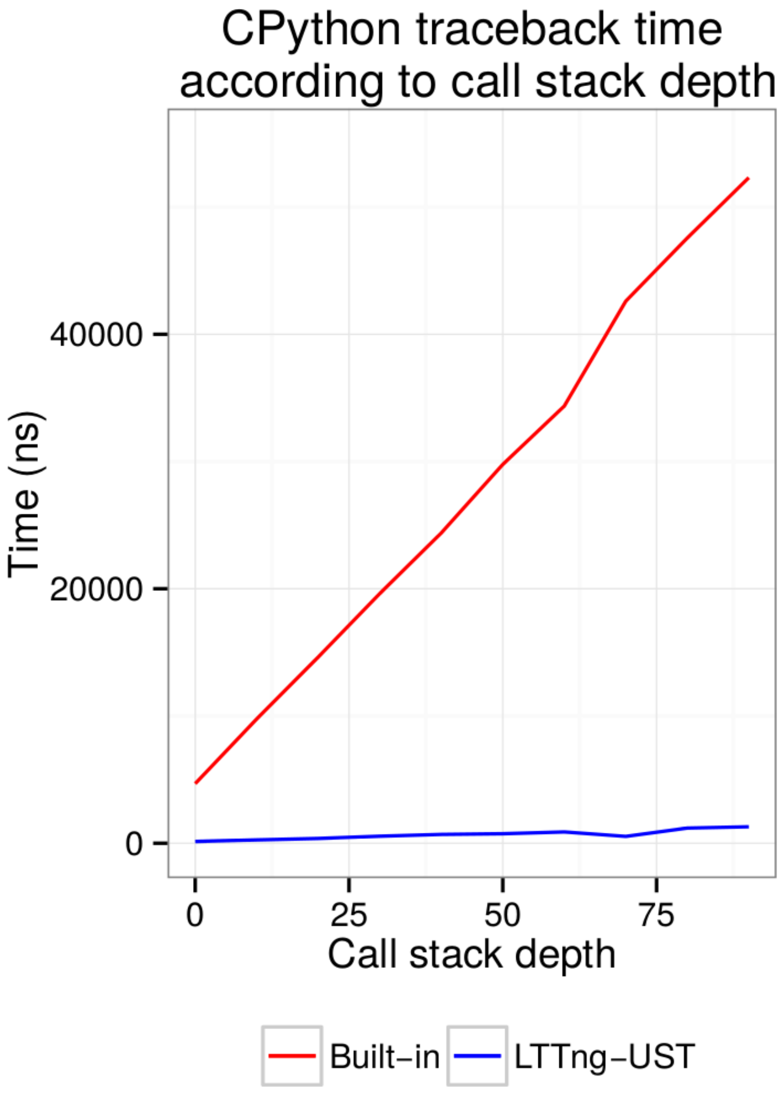


Profile a path segment by call stack



Call stack computation



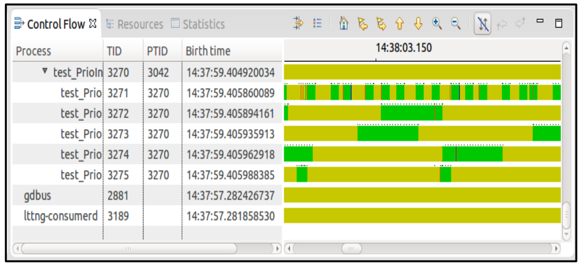


Real-time systems analysis

- Develop a model to define real-time task executions in a trace.
- Identify common problems in real-time systems and useful information for their analysis.
- Develop a method to analyze the trace segment corresponding to an execution to identify if the execution presents a problem.

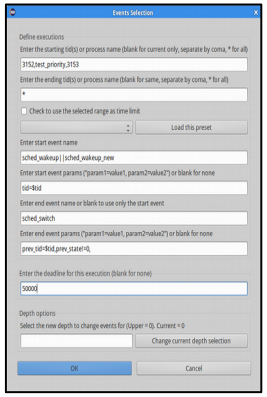
Overview

1) Control Flow View



Process	TID	PTID	Birth time
▼ test_PrioIn	3270	3042	14:37:59.404920034
test_Prio	3271	3270	14:37:59.405860089
test_Prio	3272	3270	14:37:59.405894161
test_Prio	3273	3270	14:37:59.405935913
test_Prio	3274	3270	14:37:59.405962918
test_Prio	3275	3270	14:37:59.405988385
gdbus	2881		14:37:57.282426737
lttng-consumerd	3189		14:37:57.281858530

2) Define executions



Define executions

Enter the starting tid(s) or process name (blank for current only, separate by coma, * for all)

3152,test_priority,3153

Enter the ending tid(s) or process name (blank for same, separate by coma, * for all)

*

☐ Check to use the selected range as time limit

Enter start event name

sched_wakeup||sched_wakeup_new

Enter start event params ("param1=value1, param2=value2") or blank for none

tid=$tid

Enter end event name or blank to use only the start event

sched_switch

Enter end event params ("param1=value1, param2=value2") or blank for none

prev_tid=$tid,prev_state!=0,

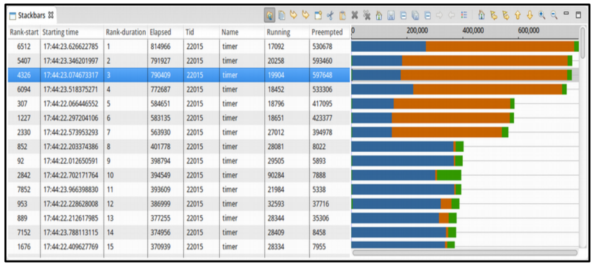
Enter the deadline for this execution (blank for none)

50000

Depth options

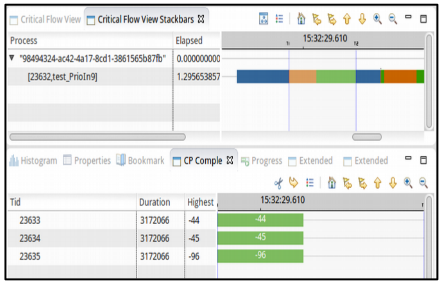
Select the new depth to change events for (Upper = 0). Current = 0

3) Stackbars View



Rank-start	Starting time	Rank-duration	Elapsed	Tid	Name	Running	Preempted
6512	17:44:23.626622785	1	814966	22015	timer	17092	530678
5407	17:44:23.346201997	2	791927	22015	timer	20258	593460
4326	17:44:23.074673317	3	790409	22015	timer	19904	597648
6094	17:44:23.518375271	4	772687	22015	timer	18452	533306
307	17:44:22.066446552	5	584651	22015	timer	18796	417095
1227	17:44:22.297204106	6	583135	22015	timer	18651	423377
2330	17:44:22.573953293	7	563930	22015	timer	27012	394978
852	17:44:22.203374386	8	401778	22015	timer	28081	8022
92	17:44:22.012650591	9	398794	22015	timer	29505	5893
2842	17:44:22.702171764	10	394549	22015	timer	90284	7888
7852	17:44:23.966398830	11	393609	22015	timer	21984	5338
953	17:44:22.228628008	12	386999	22015	timer	32593	37716
889	17:44:22.212617985	13	377255	22015	timer	28344	35306
7152	17:44:23.788113115	14	374956	22015	timer	28409	8458
1676	17:44:22.409627769	15	370939	22015	timer	28334	7955

4) Critical Flow View with
CP Complement view

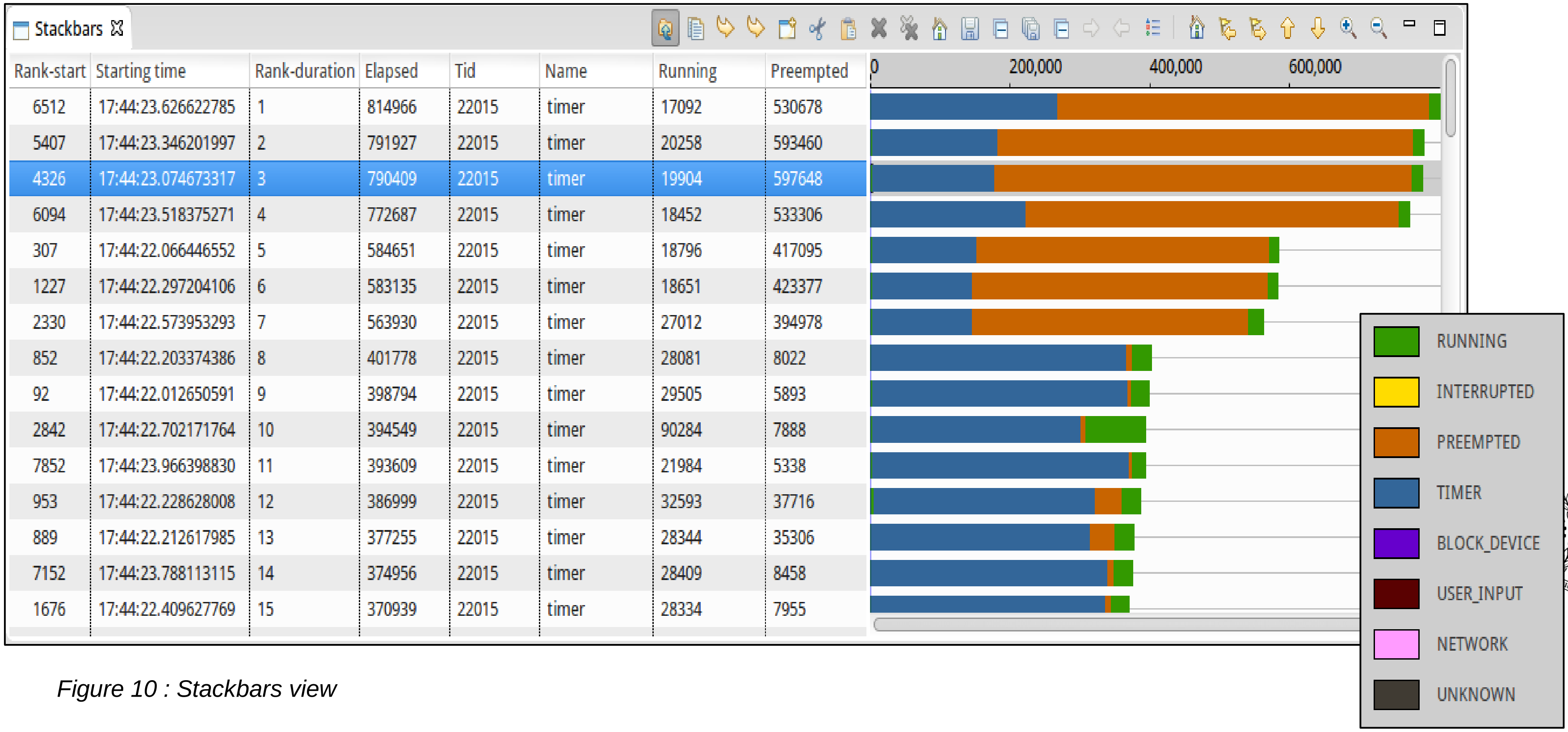


Process	Elapsed
▼ "98494324-ac42-4a17-8cd1-3861565b87fb"	0.000000000
[23632,test_PrioIn9]	1.295653857

Tid	Duration	Highest
23633	3172066	-44
23634	3172066	-45
23635	3172066	-96

5) Other views

Stackbar view: comparing task executions

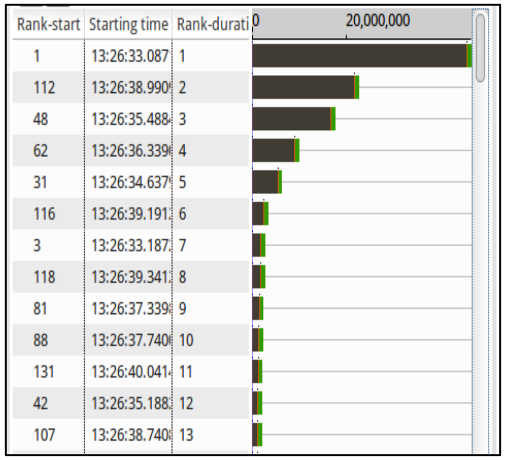


Rank-start	Starting time	Rank-duration	Elapsed	Tid	Name	Running	Preempted
6512	17:44:23.626622785	1	814966	22015	timer	17092	530678
5407	17:44:23.346201997	2	791927	22015	timer	20258	593460
4326	17:44:23.074673317	3	790409	22015	timer	19904	597648
6094	17:44:23.518375271	4	772687	22015	timer	18452	533306
307	17:44:22.066446552	5	584651	22015	timer	18796	417095
1227	17:44:22.297204106	6	583135	22015	timer	18651	423377
2330	17:44:22.573953293	7	563930	22015	timer	27012	394978
852	17:44:22.203374386	8	401778	22015	timer	28081	8022
92	17:44:22.012650591	9	398794	22015	timer	29505	5893
2842	17:44:22.702171764	10	394549	22015	timer	90284	7888
7852	17:44:23.966398830	11	393609	22015	timer	21984	5338
953	17:44:22.228628008	12	386999	22015	timer	32593	37716
889	17:44:22.212617985	13	377255	22015	timer	28344	35306
7152	17:44:23.788113115	14	374956	22015	timer	28409	8458
1676	17:44:22.409627769	15	370939	22015	timer	28334	7955

🟩	RUNNING
🟨	INTERRUPTED
🟧	PREEMPTED
🟦	TIMER
🟪	BLOCK_DEVICE
🟥	USER_INPUT
🟪	NETWORK
⬛	UNKNOWN

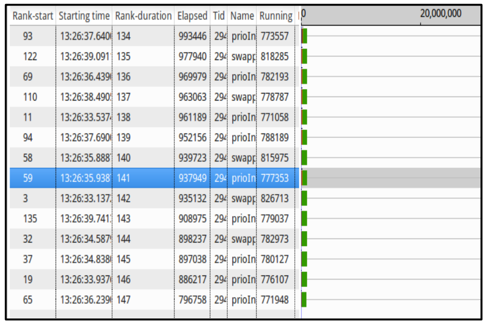
Figure 10 : Stackbars view

Distribution: duration and along the time axis



Rank-start	Starting time	Rank-durati
1	13:26:33.087	1
112	13:26:38.990	2
48	13:26:35.488	3
62	13:26:36.339	4
31	13:26:34.637	5
116	13:26:39.191	6
3	13:26:33.187	7
118	13:26:39.341	8
81	13:26:37.339	9
88	13:26:37.740	10
131	13:26:40.041	11
42	13:26:35.188	12
107	13:26:38.740	13

Figure 18 : Problematic executions



Rank-start	Starting time	Rank-duration	Elapsed	Tid	Name	Running
93	13:26:37.640	134	993446	294	prioIn	773557
122	13:26:39.091	135	977940	294	swapp	818285
69	13:26:36.439	136	969979	294	prioIn	782193
110	13:26:38.490	137	963063	294	swapp	778787
11	13:26:33.537	138	961189	294	prioIn	771058
94	13:26:37.690	139	952156	294	prioIn	788189
58	13:26:35.888	140	939723	294	swapp	815975
59	13:26:35.938	141	937949	294	prioIn	777353
3	13:26:33.137	142	935132	294	swapp	826713
135	13:26:39.741	143	908975	294	prioIn	779037
32	13:26:34.587	144	898237	294	swapp	782973
37	13:26:34.838	145	897038	294	prioIn	780127
19	13:26:33.937	146	886217	294	prioIn	776107
65	13:26:36.239	147	796758	294	prioIn	771948

Figure 19 : Normal executions

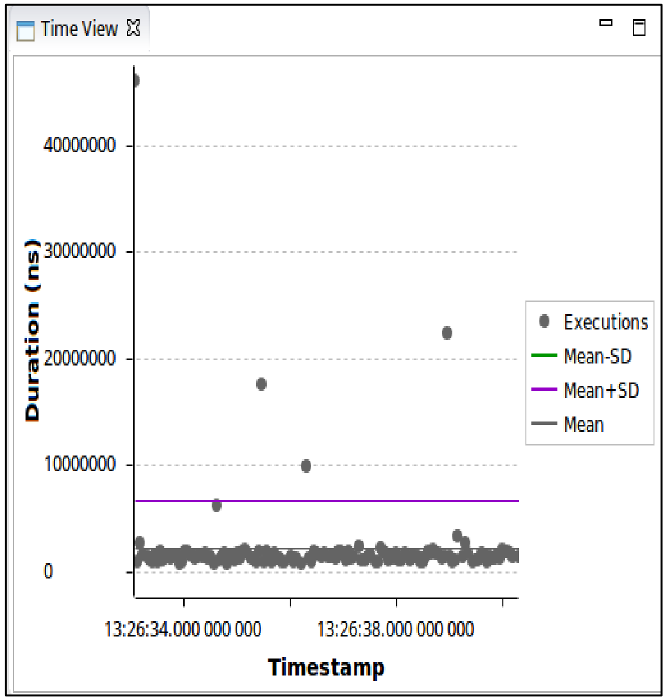
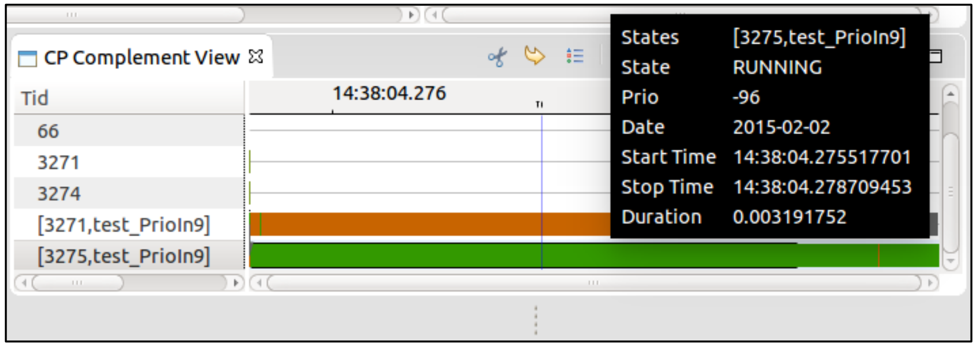


Figure 20 : Time View

Critical path view for a task execution

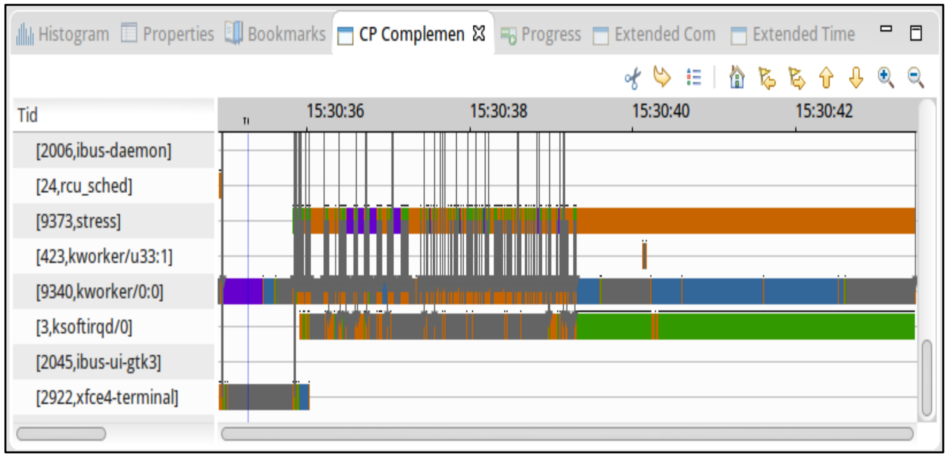


CP Complement View

Tid
66
3271
3274
[3271,test_PrioIn9]
[3275,test_PrioIn9]

14:38:04.276

States	[3275,test_PrioIn9]
State	RUNNING
Prio	-96
Date	2015-02-02
Start Time	14:38:04.275517701
Stop Time	14:38:04.278709453
Duration	0.003191752



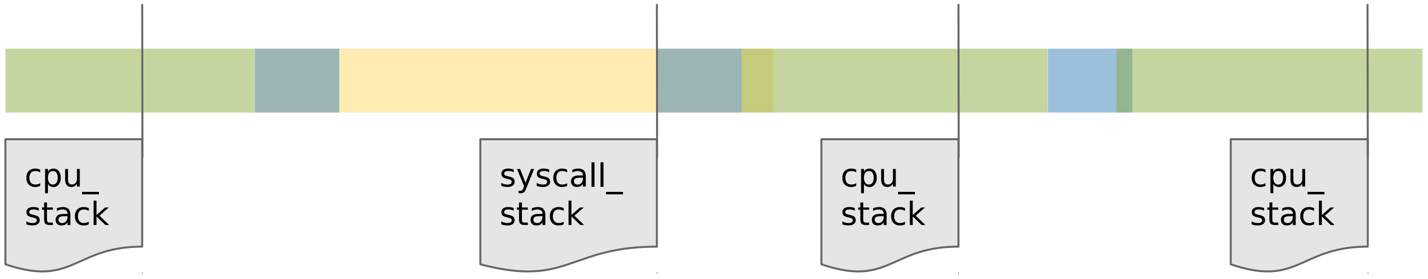
Histogram | Properties | Bookmarks | CP Complemen | Progress | Extended Com | Extended Time

Tid
[2006,ibus-daemon]
[24,rcu_sched]
[9373,stress]
[423,kworker/u33:1]
[9340,kworker/0:0]
[3,ksoftirqd/0]
[2045,ibus-ui-gtk3]
[2922,xfce4-terminal]

15:30:36 15:30:38 15:30:40 15:30:42

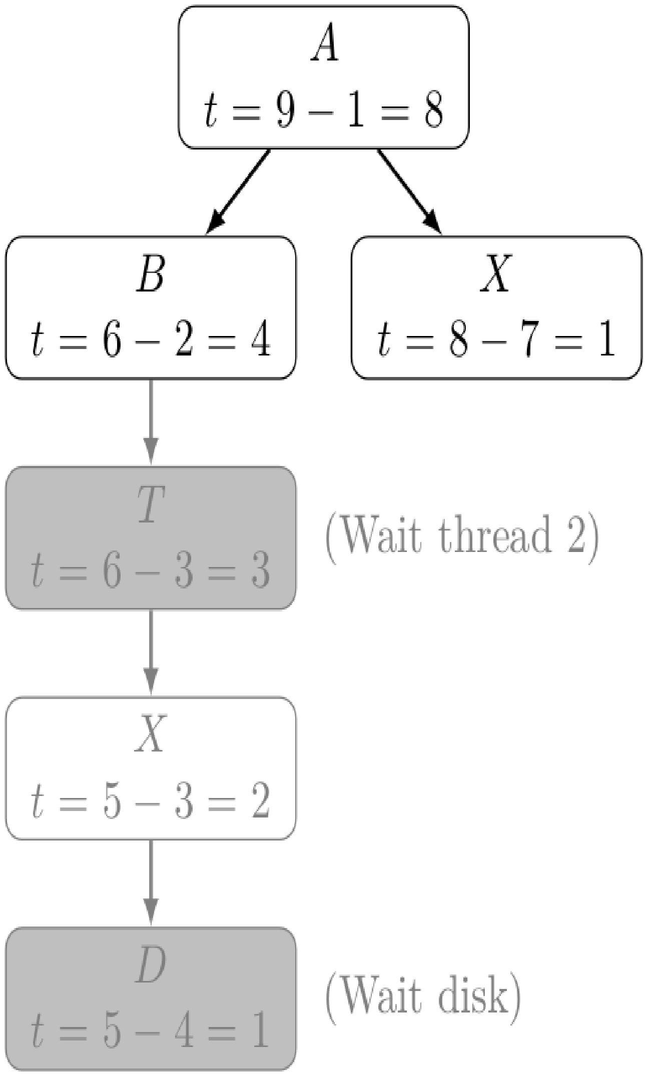
TraceCompare: diagnose performance variations

- Good versus bad run: program, library or OS version, hardware, network load?
- Collect execution traces and compute different metrics for different tasks (e.g., time for complete process, search request, or frame redraw).
- Add call stack samples collected periodically, or upon a very long system call.



Call context tree with nested calls and waits

Time	Thread 1
1	Call A
2	Call B
3	
4	
5	
6	Return B
7	Call X
8	Return X
9	Return A

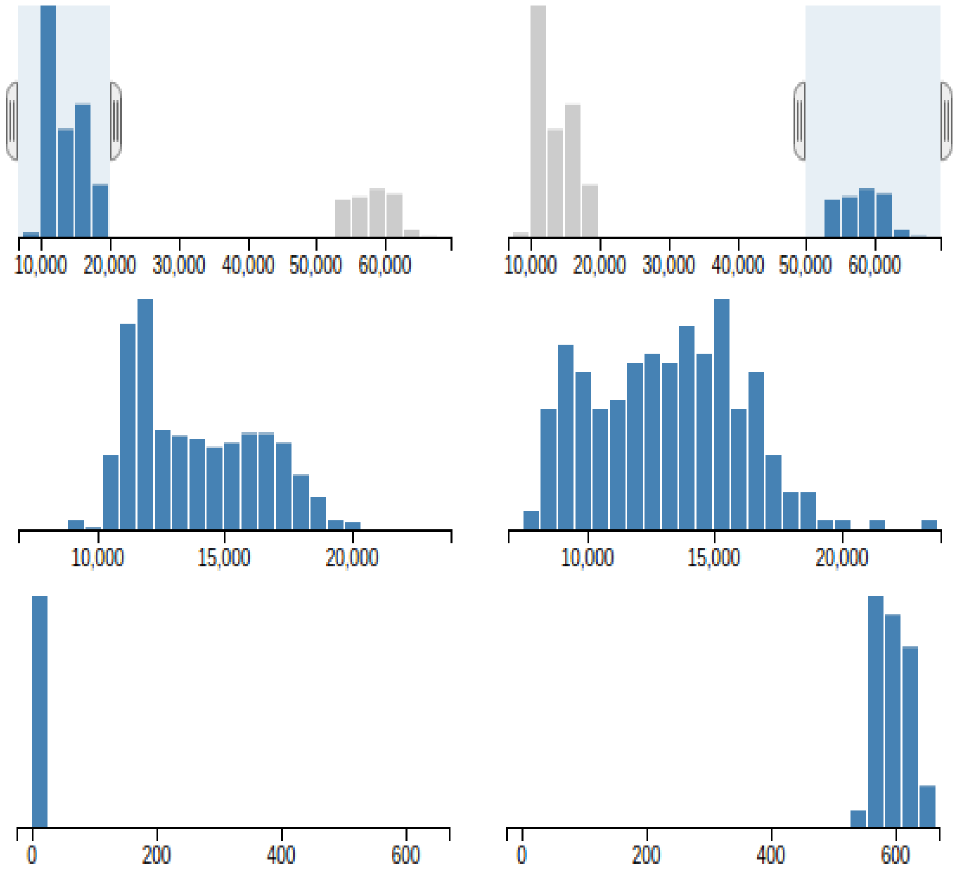


Select groups of executions based on metrics

Total Time

Running Time

Bytes read from disk

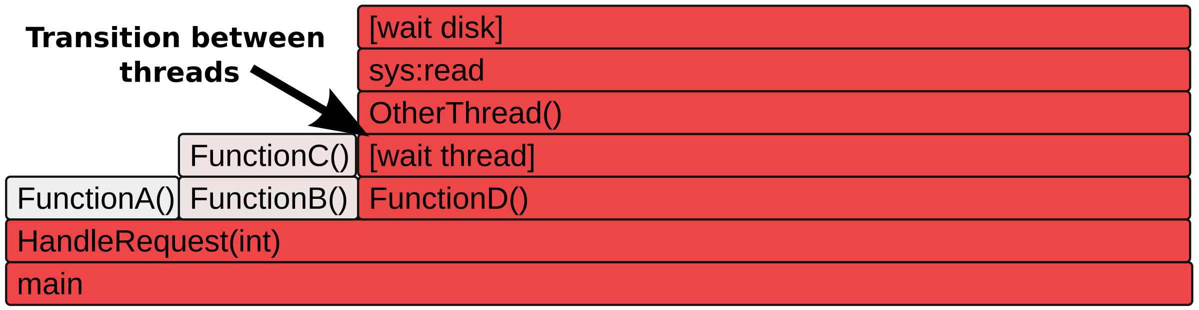


Group A

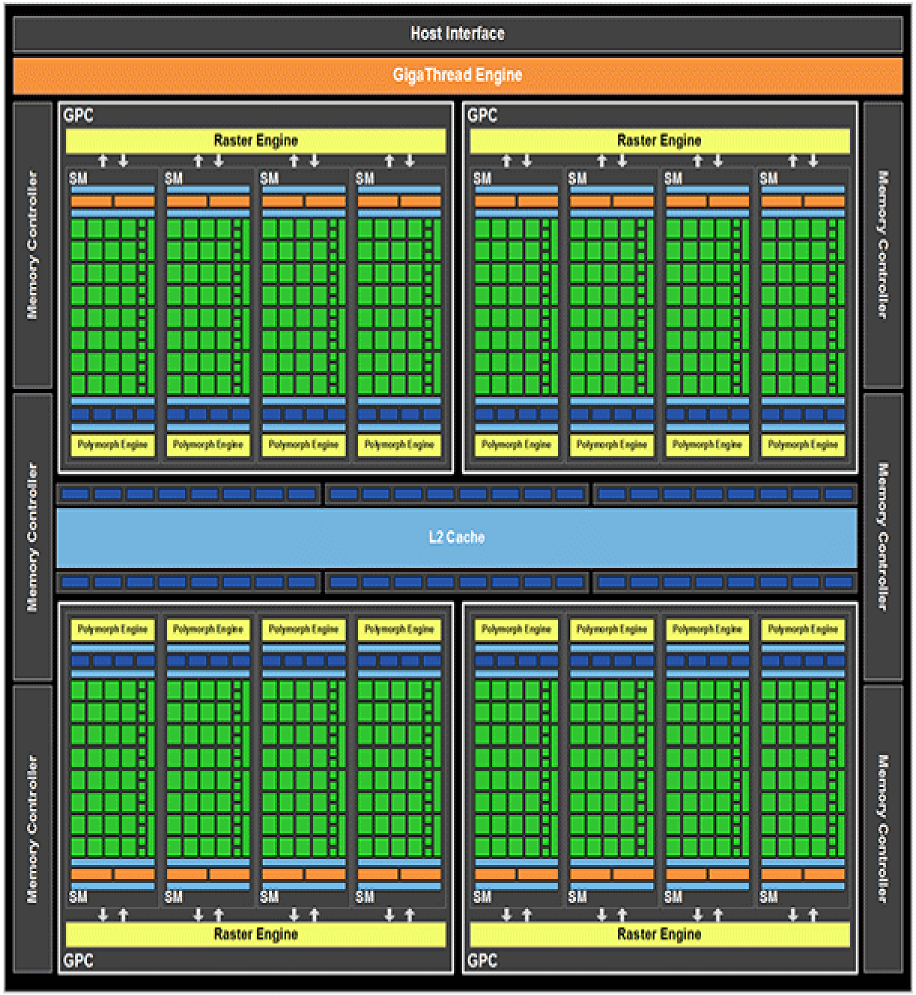
Group B

Differential flame graph of the two groups

- View of call stack over time for the tasks (assuming relatively stable call tree).
- Average duration of each call in group A versus group B, highlight where the differences are significant.

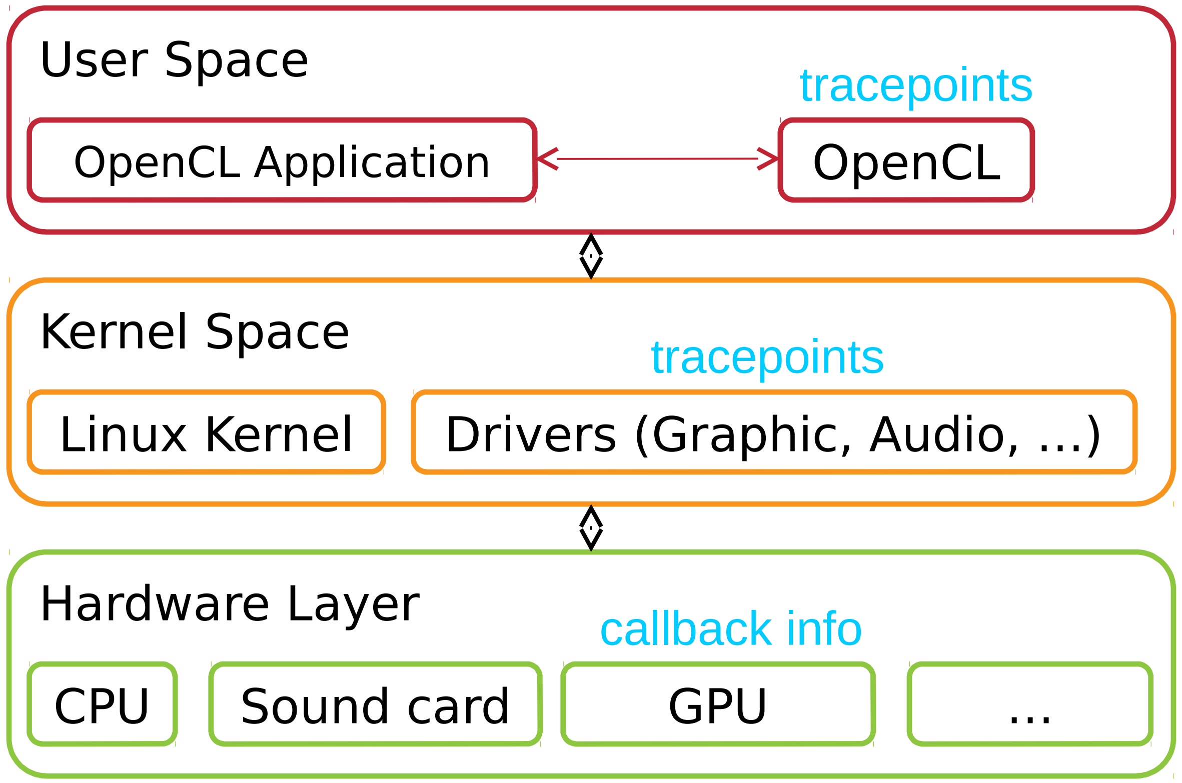


Tracing the GPU as an asynchronous resource



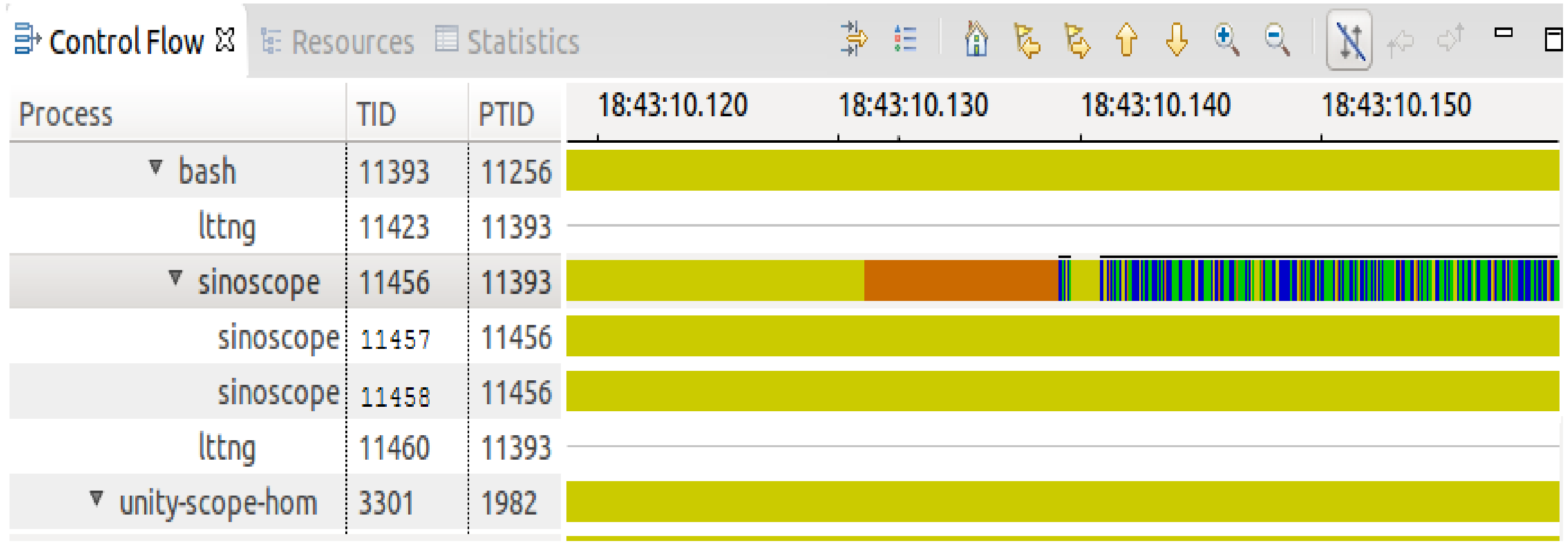
Source: nvidia.com

Combining info from OpenCL, driver and HW



CPU trace

Trace Compass:



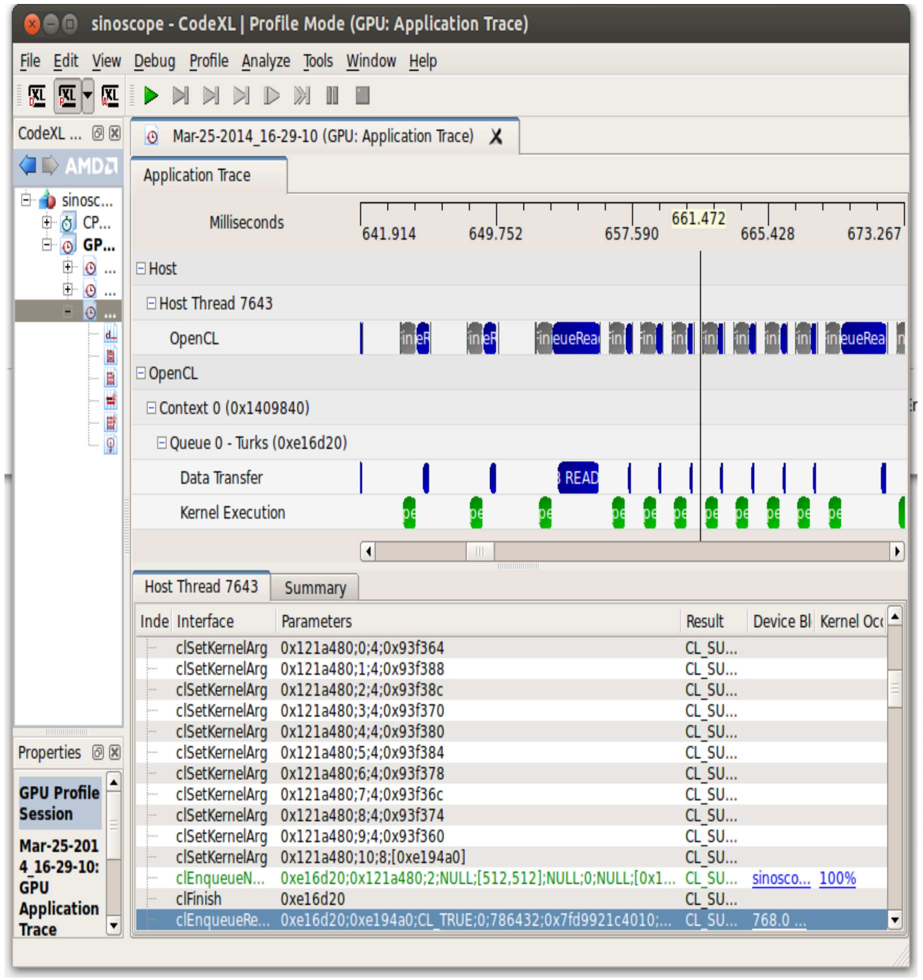
Process	TID	PTID
▼ bash	11393	11256
lttng	11423	11393
▼ sinoscope	11456	11393
sinoscope	11457	11456
sinoscope	11458	11456
lttng	11460	11393
▼ unity-scope-hom	3301	1982

GPU trace

CodeXL
(AMD’s GPU tracing tool)

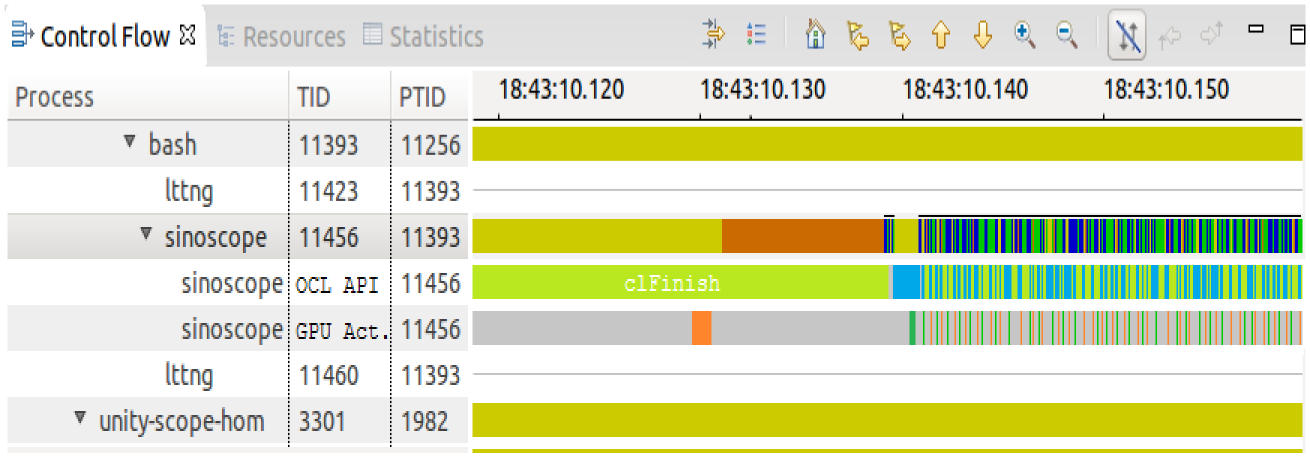
Limitations:

- OpenCL application has to be launched by CodeXL
- Recording trace performance for large traces
- AMD GPU only


sinoscope - CodeXL | Profile Mode (GPU: Application Trace)
File Edit View Debug Profile Analyze Tools Window Help
CodeXL ...
Mar-25-2014_16-29-10 (GPU: Application Trace)
Application Trace
Milliseconds
641.914
649.752
657.590
661.472
665.428
673.267
Host
Host Thread 7643
OpenCL
OpenCL
Context 0 (0x1409840)
Queue 0 - Turks (0xe16d20)
Data Transfer
READ
Kernel Execution
Properties
GPU Profile Session
Mar-25-2014_16-29-10: GPU Application Trace
Host Thread 7643
Summary

Inde	Interface	Parameters	Result	Device Bl	Kernel Occ
	clSetKernelArg	0x121a480;0;4;0x93f364	CL_SU...		
	clSetKernelArg	0x121a480;1;4;0x93f388	CL_SU...		
	clSetKernelArg	0x121a480;2;4;0x93f38c	CL_SU...		
	clSetKernelArg	0x121a480;3;4;0x93f370	CL_SU...		
	clSetKernelArg	0x121a480;4;4;0x93f380	CL_SU...		
	clSetKernelArg	0x121a480;5;4;0x93f384	CL_SU...		
	clSetKernelArg	0x121a480;6;4;0x93f378	CL_SU...		
	clSetKernelArg	0x121a480;7;4;0x93f36c	CL_SU...		
	clSetKernelArg	0x121a480;8;4;0x93f374	CL_SU...		
	clSetKernelArg	0x121a480;9;4;0x93f360	CL_SU...		
	clSetKernelArg	0x121a480;10;8;[0xe194a0]	CL_SU...		
	clEnqueueN...	0xe16d20;0x121a480;2;NULL;[512,512];NULL;0;NULL;[0x1...	CL_SU...	sinosco...	100%
	clFinish	0xe16d20	CL_SU...		
	clEnqueueRe...	0xe16d20;0xe194a0;CL_TRUE;0;786432;0x7fd9921c4010;...	CL_SU...	768.0 ...	

Combined trace: GPU contention



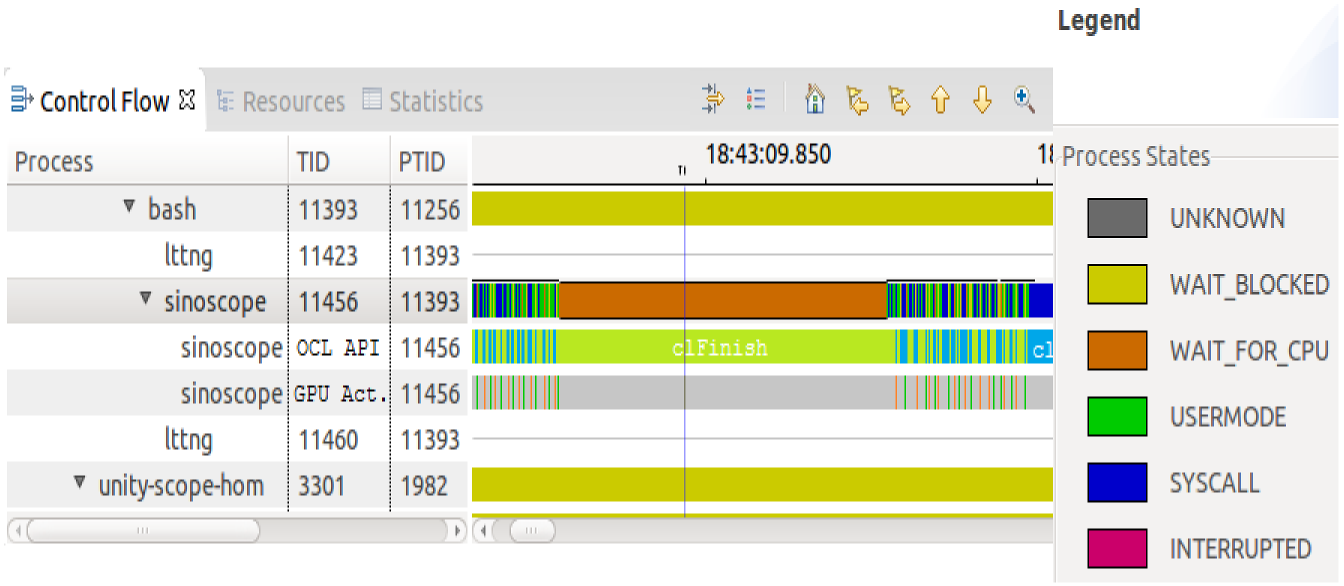
Process	TID	PTID
▼ bash	11393	11256
lttng	11423	11393
▼ sinoscope	11456	11393
sinoscope	OCL API	11456
sinoscope	GPU Act.	11456
lttng	11460	11393
▼ unity-scope-hom	3301	1982

Legend

Process States

⬛	UNKNOWN
🟨	WAIT_BLOCKED
🟧	WAIT_FOR_CPU
🟩	USERMODE
🟦	SYSCALL
🟥	INTERRUPTED

Combined trace: CPU preemption



Control Flow | Resources | Statistics

Process	TID	PTID
▼ bash	11393	11256
lttng	11423	11393
▼ sinoscope	11456	11393
sinoscope	OCL API	11456
sinoscope	GPU Act.	11456
lttng	11460	11393
▼ unity-scope-hom	3301	1982

Legend

Process States

⬛	UNKNOWN
🟨	WAIT_BLOCKED
🟧	WAIT_FOR_CPU
🟩	USERMODE
🟦	SYSCALL
🟥	INTERRUPTED

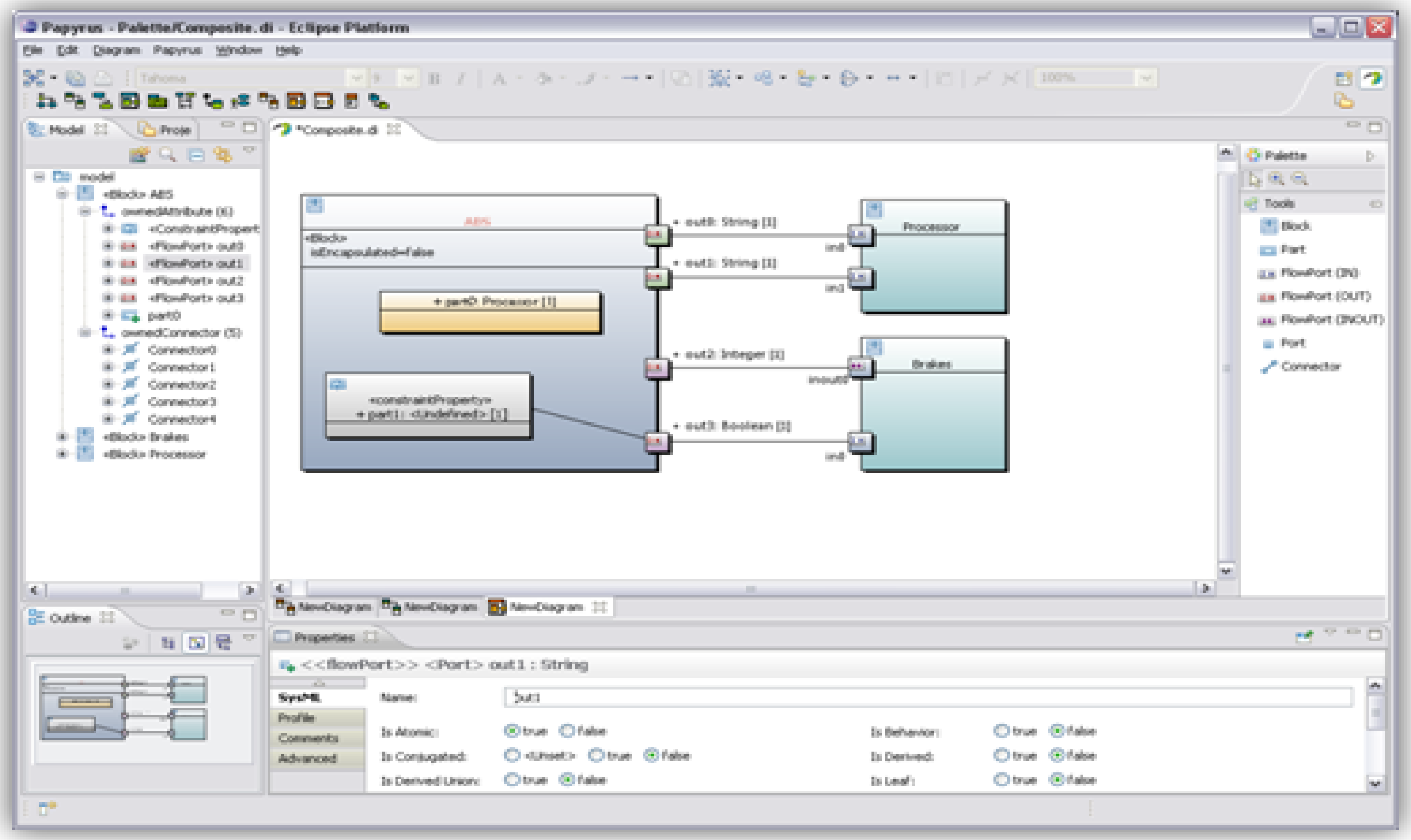
T4: Model-Driven Engineering support

Objective: fully support model-driven engineering with tracing, debugging and testing tools.

Personnel: Prof. Juergen Dingel, Francis Bordeleau Ph.D., Sébastien Gérard Ph.D., Bernd Hufmann B.Eng., RA1.

- T5M1, T5M2: extend the model-level to store as attribute values, current states, transitions taken and messages sent.
- T5D1: new algorithms and techniques for the automatic correction and refinement of environment assumptions based on runtime information.
- T5M3, T5M4: extend the model-level to specify test cases, execute the tests and annotate the results.
- T5D2: new automatic test generation algorithms for correcting, refining and completing the model-level information.

Model-Driven Engineering support



<<flowPort>> <Port> out1 : String

Name: out1

Is Atomic: [✓ true] false
Is Conjugated: <Unset> true [✓ false]
Is Derived Union: true [✓ false]
Is Behavior: true [✓ false]
Is Derived: true [✓ false]
Is Leaf: true [✓ false]

Conclusion

- Efficient tracing is an essential tool for high performance heterogeneous distributed systems.
- Free software as infrastructure: natural collaboration, rich user community, easy customization, no surprise (e.g. product or supplier end of life, high cost per seat).
- Viewpoints of multiple clients: benefit from the experience, use cases and best practices of several technology leaders.
- Technology transfer: research and prototyping in Polytechnique, collaborative development of mainline implementations.
- LTTng delivers low disturbance, scalable tracing.