Enhanced execution trace abstraction approach using social network analysis methods

Ji Wang, **Naser Ezzati-Jivan**

Oct 11, 2020

Symposium on Software Performance (SSP) 2020

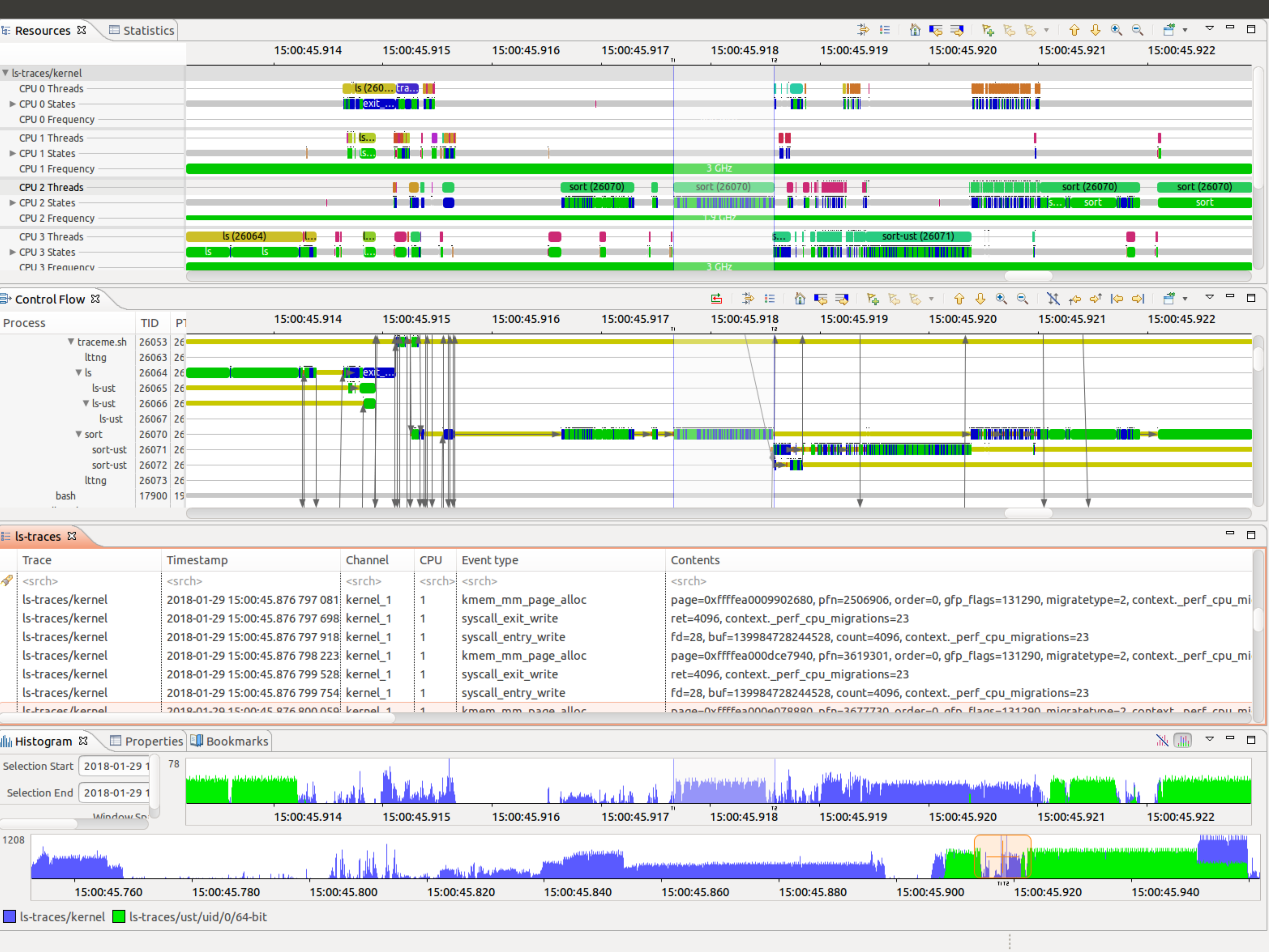
Introduction

Execution tracing is used for performance analysis and debugging.

Usually the trace size is huge, and we need some heuristics to cope with it.

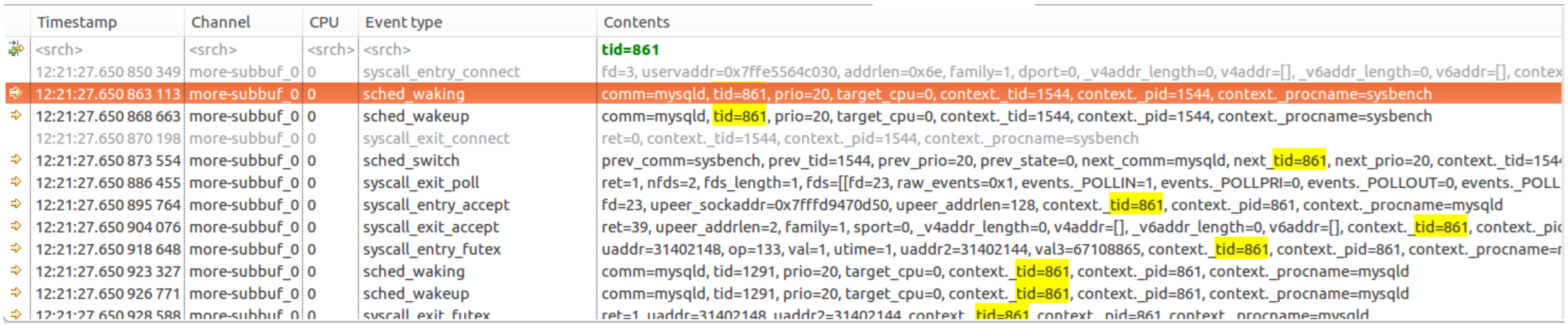
Process	TID	PTID	Birth time
gnome-pty-helpe	3609	3596	18:15:38.7695138
▼ bash	3610	3596	18:15:38.7695147
ssh	8972	3610	18:15:38.7695557
▼ bash	5332	3596	18:15:38.7695248
ls	15298	5332	18:15:39.9718121
▼ bash	10234	3596	18:15:38.7695260
ssh	8989	10234	18:15:38.7695567
▼ bash	11275	3596	18:15:38.7695713
▼ bash	15266	11275	18:15:38.7729050
▼ sudo	15291	15266	18:15:38.7729062
lttng	15292	15291	18:15:38.7729070
ab	15294	15266	18:15:38.8254856
▼ sudo	15302	15266	18:15:46.0692929
lttng	15303	15302	18:15:46.0735909
dconf worker	3599	2779	18:15:38.7695104
gmain	3600	2779	18:15:38.7695113
gdbus	3601	2779	18:15:38.7695120

Process	TID	PTID	Birth time
unity-greeter	1082	1070	12:21:27.6053039
llvmpipe-0	1159	1070	12:21:27.6053052
llvmpipe-1	1160	1070	12:21:27.6053064
gmain	1175	1070	12:21:27.6053082
gdbus	1176	1070	12:21:27.6053093
dconf worker	1193	1070	12:21:27.6053111
threaded-ml	1295	1070	12:21:27.6053126
gmain	1037	851	12:21:27.6052899
gdbus	1038	851	12:21:27.6052916
lightdm	1200	851	12:21:27.6053416
gmain	872	1	12:21:27.6052339
gdbus	874	1	12:21:27.6052352
▼ mysqld	861	1	12:21:27.6052366
mysqld	1545	861	12:21:27.6665481
mysqld	941	1	12:21:27.6052379
mysqld	955	1	12:21:27.6052392
mysqld	956	1	12:21:27.6052408

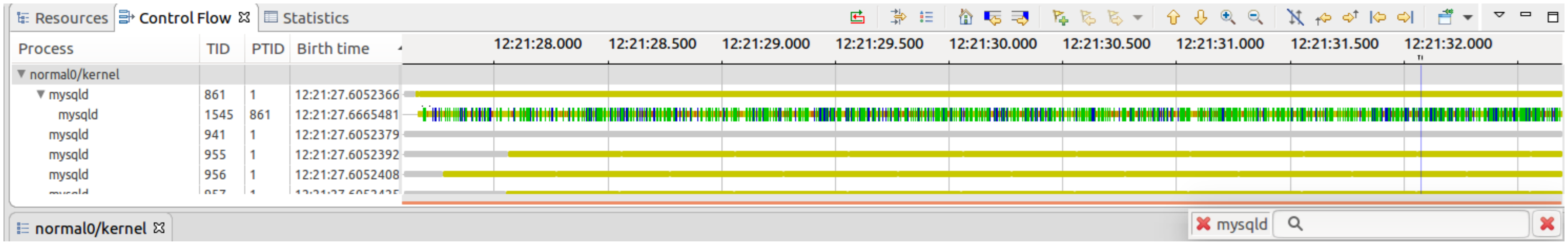

Resources
Statistics
15:00:45.914
15:00:45.915
15:00:45.916
15:00:45.917
15:00:45.918
15:00:45.919
15:00:45.920
15:00:45.921
15:00:45.922
ls-traces/kernel
CPU 0 Threads
CPU 0 States
CPU 0 Frequency
CPU 1 Threads
CPU 1 States
CPU 1 Frequency
CPU 2 Threads
CPU 2 States
CPU 2 Frequency
CPU 3 Threads
CPU 3 States
CPU 3 Frequency
ls (26064)
sort (26070)
sort-ust (26071)
3 GHz
Control Flow
Process
TID
traceme.sh 26053
lttng 26063
ls 26064
ls-ust 26065
ls-ust 26066
ls-ust 26067
sort 26070
sort-ust 26071
sort-ust 26072
lttng 26073
bash 17900
ls-traces
Trace
Timestamp
Channel
CPU
Event type
Contents
ls-traces/kernel 2018-01-29 15:00:45.876 797 081 kernel_1 1 kmem_mm_page_alloc page=0xffffea0009902680, pfn=2506906, order=0, gfp_flags=131290, migratetype=2, context._perf_cpu_mi
ls-traces/kernel 2018-01-29 15:00:45.876 797 698 kernel_1 1 syscall_exit_write ret=4096, context._perf_cpu_migrations=23
ls-traces/kernel 2018-01-29 15:00:45.876 797 918 kernel_1 1 syscall_entry_write fd=28, buf=139984728244528, count=4096, context._perf_cpu_migrations=23
ls-traces/kernel 2018-01-29 15:00:45.876 798 223 kernel_1 1 kmem_mm_page_alloc page=0xffffea000dce7940, pfn=3619301, order=0, gfp_flags=131290, migratetype=2, context._perf_cpu_mi
ls-traces/kernel 2018-01-29 15:00:45.876 799 528 kernel_1 1 syscall_exit_write ret=4096, context._perf_cpu_migrations=23
ls-traces/kernel 2018-01-29 15:00:45.876 799 754 kernel_1 1 syscall_entry_write fd=28, buf=139984728244528, count=4096, context._perf_cpu_migrations=23
Histogram
Properties
Bookmarks
Selection Start
Selection End
78
1208
15:00:45.760
15:00:45.780
15:00:45.800
15:00:45.820
15:00:45.840
15:00:45.860
15:00:45.880
15:00:45.900
15:00:45.920
15:00:45.940
ls-traces/kernel
ls-traces/ust/uid/0/64-bit

Current methods

- ✓Sampling or event filtering.
- ✓Time view filtering.
- ✓Global filtering

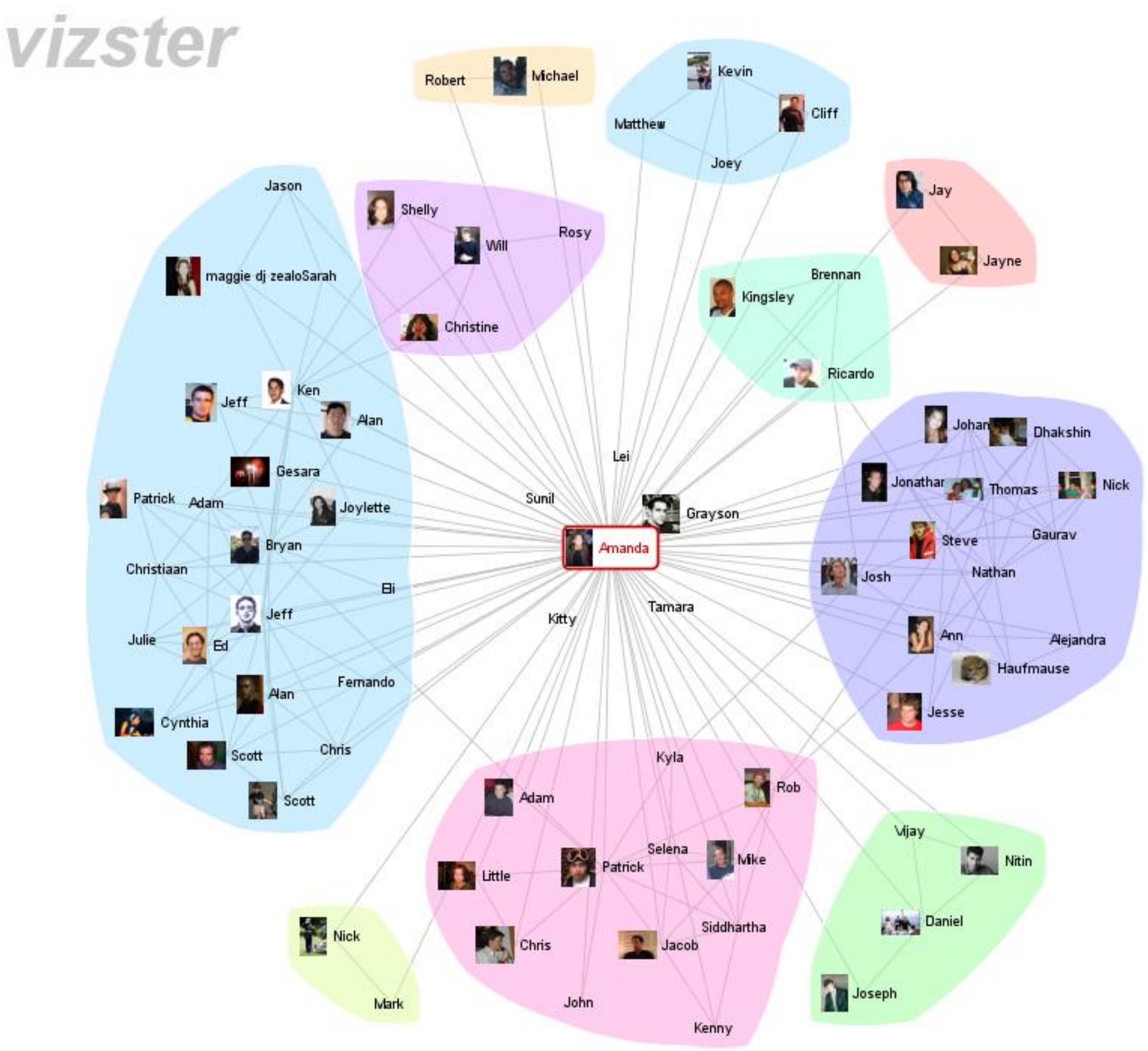


	Timestamp	Channel	CPU	Event type	Contents
	<srch>	<srch>	<srch>	<srch>	tid=861
	12:21:27.650 850 349	more-subbuf_0	0	syscall_entry_connect	fd=3, uservaddr=0x7ffe5564c030, addrlen=0x6e, family=1, dport=0, _v4addr_length=0, v4addr=[], _v6addr_length=0, v6addr=[], contex
⇨	12:21:27.650 863 113	more-subbuf_0	0	sched_waking	comm=mysqld, tid=861, prio=20, target_cpu=0, context._tid=1544, context._pid=1544, context._procname=sysbench
⇨	12:21:27.650 868 663	more-subbuf_0	0	sched_wakeup	comm=mysqld, tid=861, prio=20, target_cpu=0, context._tid=1544, context._pid=1544, context._procname=sysbench
	12:21:27.650 870 198	more-subbuf_0	0	syscall_exit_connect	ret=0, context._tid=1544, context._pid=1544, context._procname=sysbench
⇨	12:21:27.650 873 554	more-subbuf_0	0	sched_switch	prev_comm=sysbench, prev_tid=1544, prev_prio=20, prev_state=0, next_comm=mysqld, next_tid=861, next_prio=20, context._tid=154
⇨	12:21:27.650 886 455	more-subbuf_0	0	syscall_exit_poll	ret=1, nfds=2, fds_length=1, fds=[[fd=23, raw_events=0x1, events._POLLIN=1, events._POLLPRI=0, events._POLLOUT=0, events._POLL
⇨	12:21:27.650 895 764	more-subbuf_0	0	syscall_entry_accept	fd=23, upeer_sockaddr=0x7fffd9470d50, upeer_addrlen=128, context._tid=861, context._pid=861, context._procname=mysqld
⇨	12:21:27.650 904 076	more-subbuf_0	0	syscall_exit_accept	ret=39, upeer_addrlen=2, family=1, sport=0, _v4addr_length=0, v4addr=[], _v6addr_length=0, v6addr=[], context._tid=861, context._pi
⇨	12:21:27.650 918 648	more-subbuf_0	0	syscall_entry_futex	uaddr=31402148, op=133, val=1, utime=1, uaddr2=31402144, val3=67108865, context._tid=861, context._pid=861, context._procname=
⇨	12:21:27.650 923 327	more-subbuf_0	0	sched_waking	comm=mysqld, tid=1291, prio=20, target_cpu=0, context._tid=861, context._pid=861, context._procname=mysqld
⇨	12:21:27.650 926 771	more-subbuf_0	0	sched_wakeup	comm=mysqld, tid=1291, prio=20, target_cpu=0, context._tid=861, context._pid=861, context._procname=mysqld
⇨	12:21:27.650 928 588	more-subbuf_0	0	syscall_exit_futex	ret=1, uaddr=31402148, uaddr2=31402144, context._tid=861, context._pid=861, context._procname=mysqld



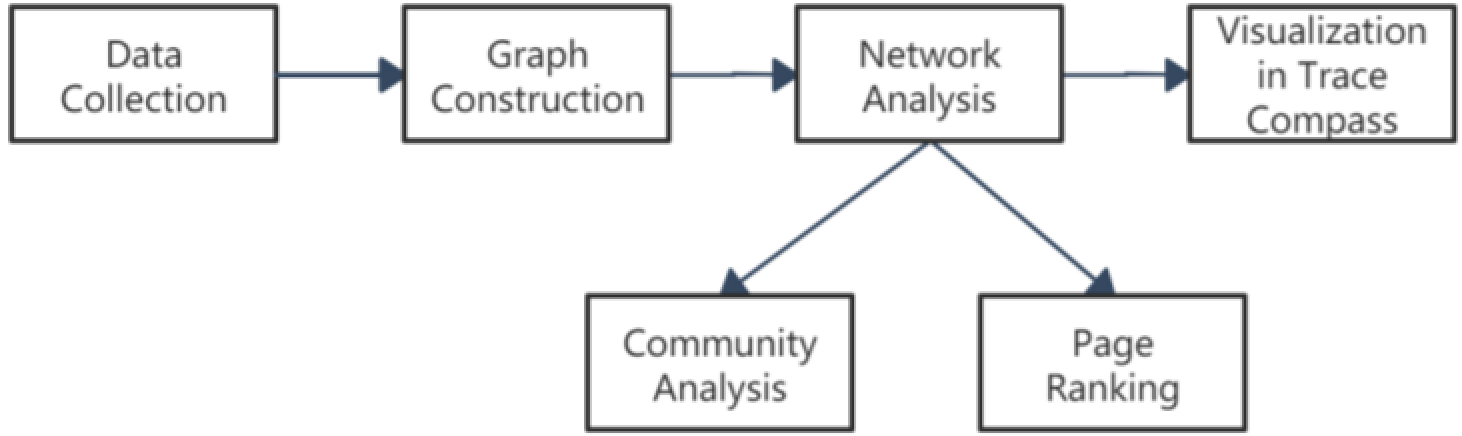
Idea

Graph theory



Picture from: http://vis.stanford.edu/jheer/projects/vizster/

Method



I.

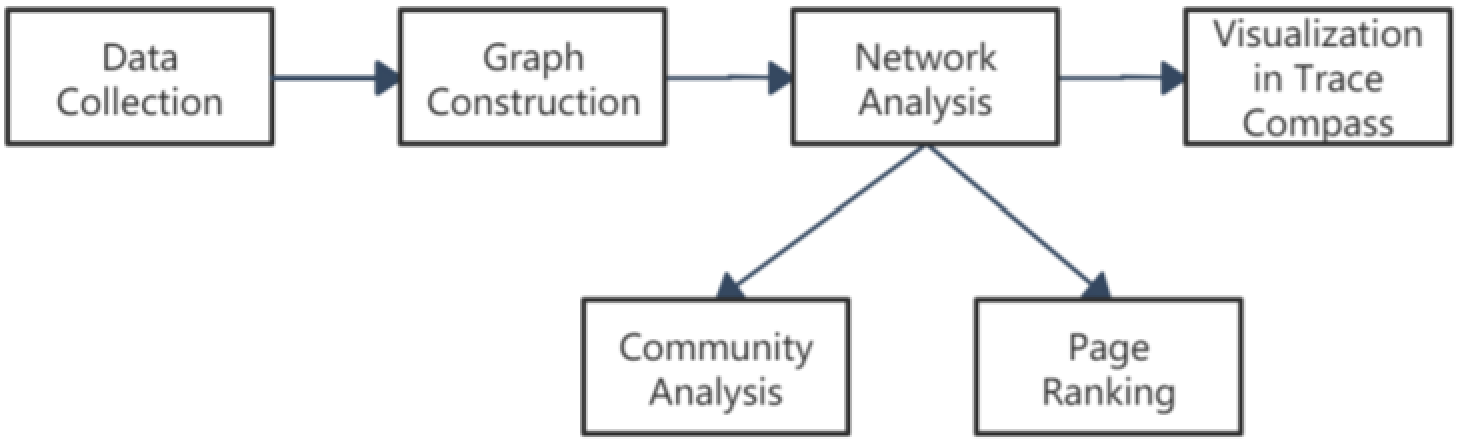
- Extract the processes interactions based on the execution graph.
- Building a graph G = (V,E)
- V includes all the processes running on the system
- E is a set of edges defined based on the interactions between the processes.

or kernel level lock methods.

Algorithm 1 Execution Graph Construction algorithm, for events collected from lock library .

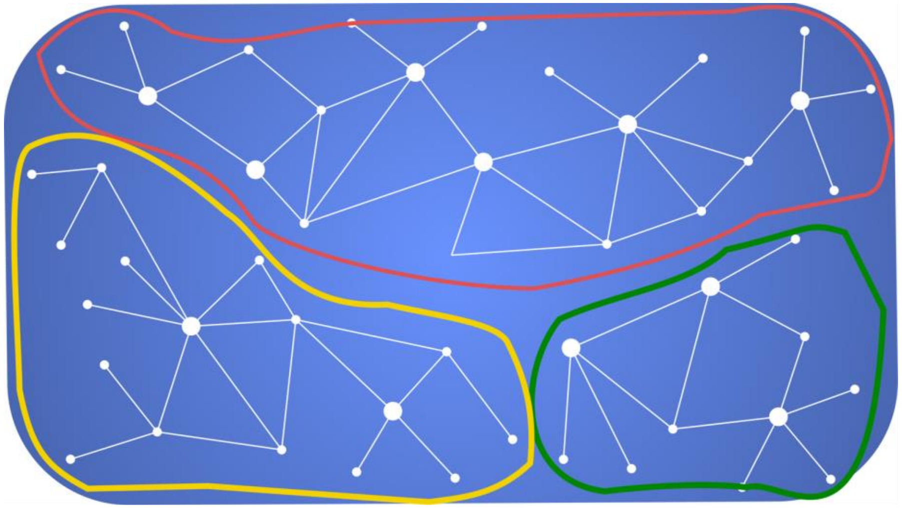
```
Input Input: Trace T, Threads = {t1, t2, ..., tn}
,
    Set1 = {*_lock_req},
    Set2 = {*_lock_acq},
    Set3 = {*_unlock}
    Output Output: execution graph G
THREADS ← initial set of threads active in the multi-thread application Declarations
LOCKHOLDER ← ∅
for all thread t ∈ THREADS do//Initialization
    Create initial vertex of thread t with timestamp t.begin
end for
for all event e ∈ T do//Mainprocedure
    if e ∈ Set1 then
        new_h_edge(e.tid, e.ts, blocked)
        new_v_edge(e.tid, LOCKHOLDER)
    end if
    if e ∈ Set2 then
        new_v_edge(LOCKHOLDER, e.tid)
        new_h_edge(e.tid, e.ts, running)
        LOCKHOLDER ← e.tid
    end if
    if e ∈ Set3 then
        LOCKHOLDER ← e.tid
    end if
```

Method



II.

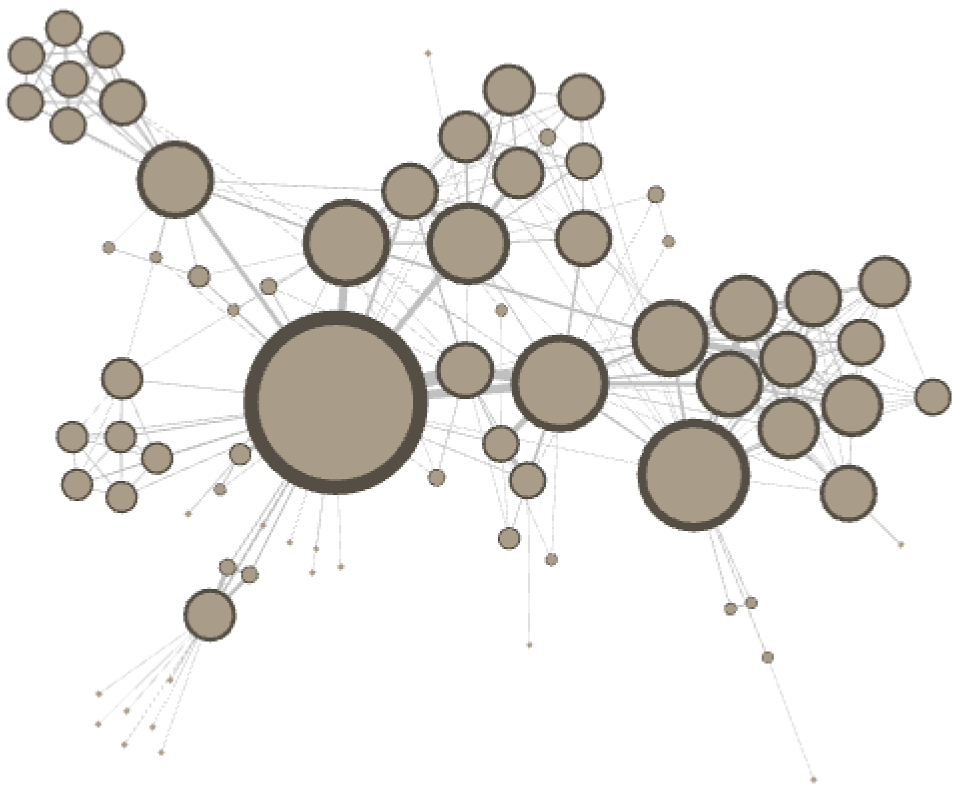
Compute connected components on the graph to extract all the subgraphs. (e.g., BFS)



Method

NB. Computing connected components on the graph

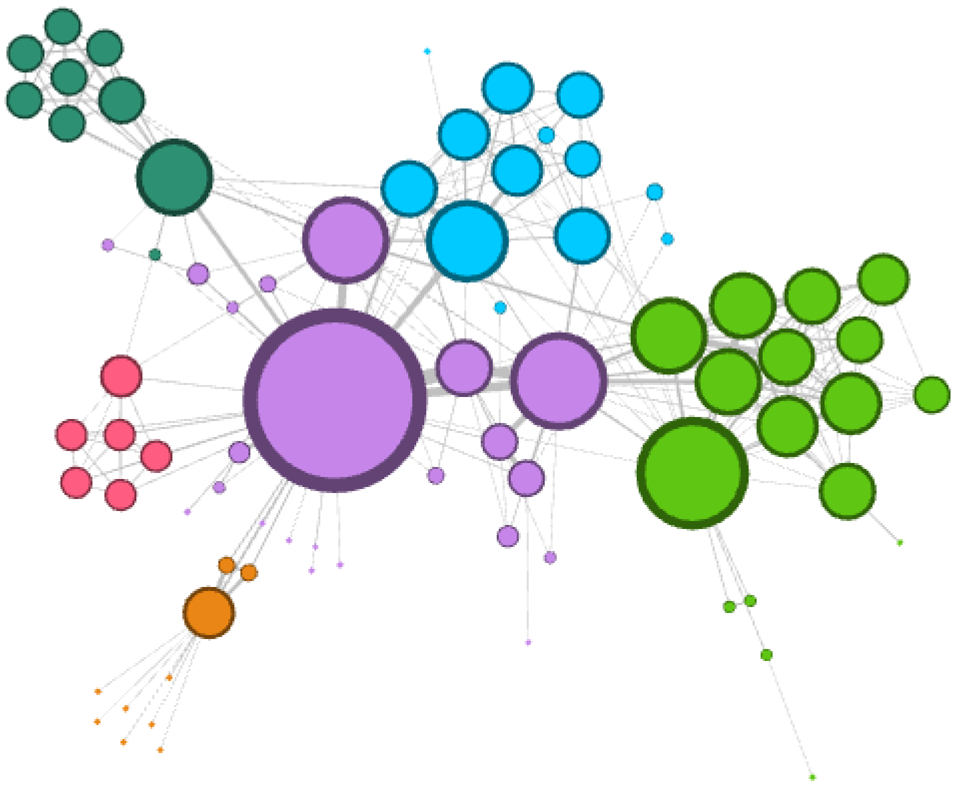
Other approaches:
e.g., Graph Modularity



Method

NB. Computing connected components on the graph

Other approaches:
e.g., Graph Modularity

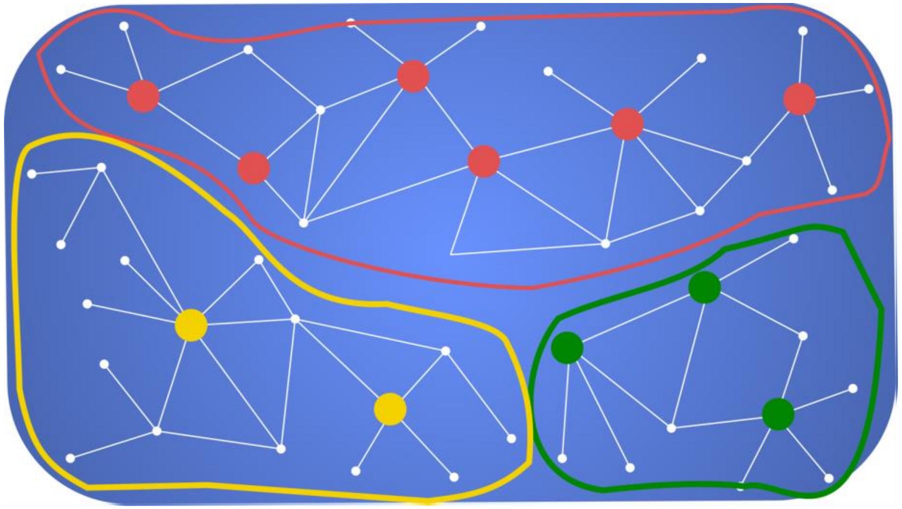


Method

III.

✓Apply a ranking algorithm on each connected component to assign a rank to each process

✓Choose the top-K processes..



Method

Modified PageRank algorithm: Includes only **one** iteration

Node rank initialization: equal weights, or even CPU/IO usage

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v}$$

Algorithm 2: Process Ranking Algorithm (PRA)

```
if event == sched_ttwu then
    j = getVMvCPU(wakee_tid);
    k = getVMvCPU(waker_tid);
    wakerCR3 = getLastCR3(vCPU_k);
    updateWaker(vCPU_j, wakerCR3);
else if event == vm_inj_virq then
    j = getVMvCPU(tid);
    pCR3 = getVMvCPU(vCPU_j);
    if vec == IPI then
        wakerCR3 = queryWaker(vCPU_j);
        LINK_HORIZONTAL(wakerCR3, pCR3);
for all process cr3_i ∈ CR3 do
    R(cr3_i);
connected_subgraph = breadth-first-search(CR3);
customized_graph = denoise(connected_subgraph, R(CR3));
```

u: a process

B_u: the set of u's in-edges

N_v: the number of out-edges of process v

c: the normalization factor

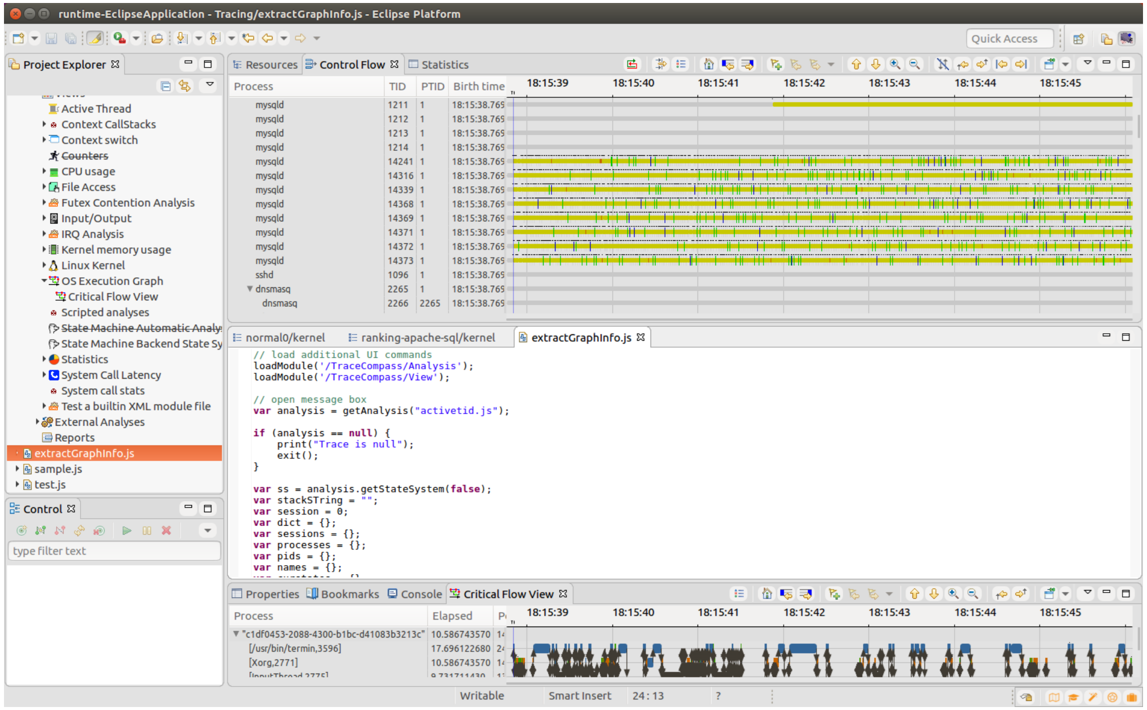
Use case I

Trace Compass:

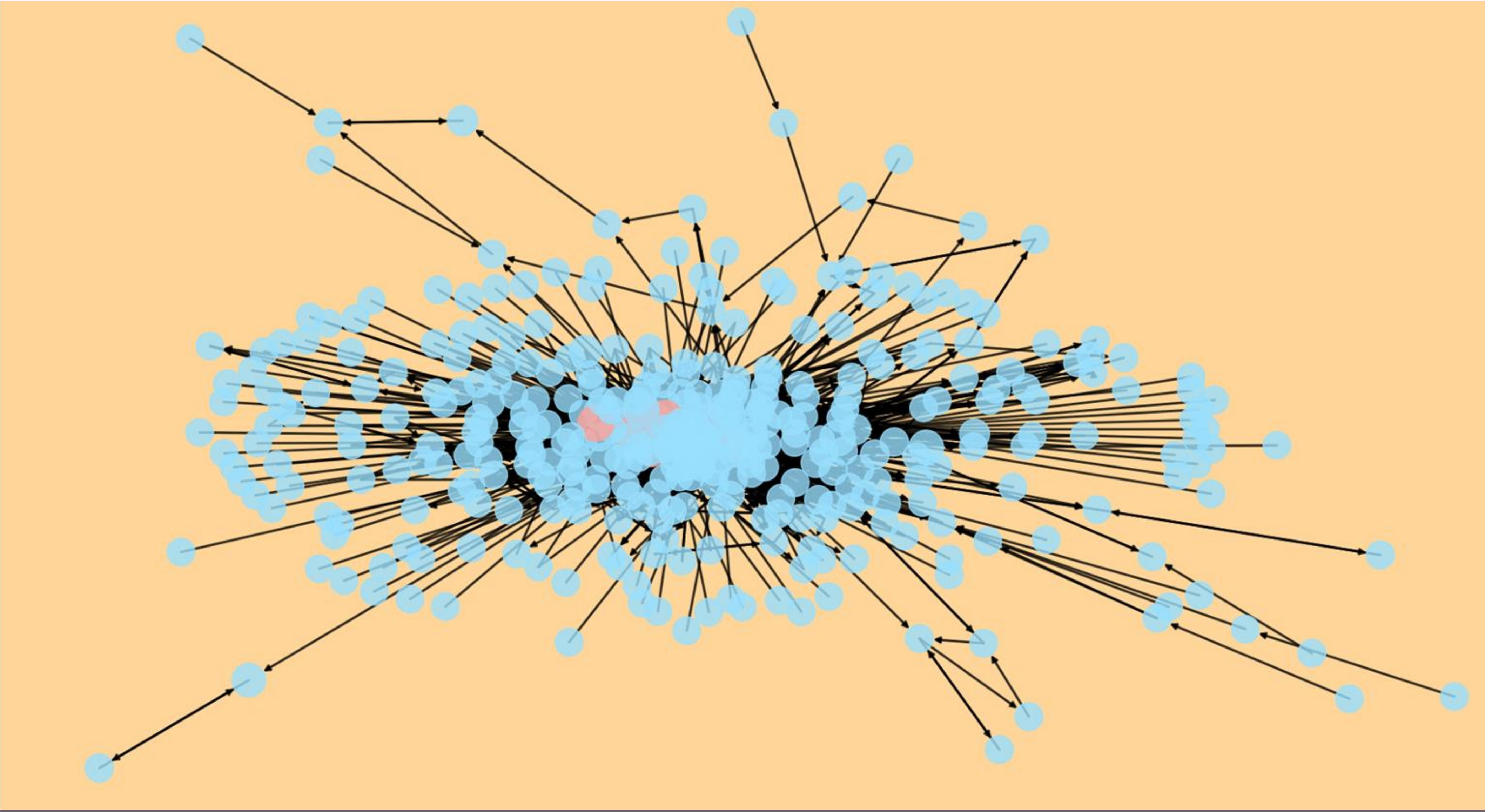
Extract process interactions using EASE scripting.

Providing a view showing top-ranked processes in each subgraph.

Filtering top-ranked processes in **event** view.

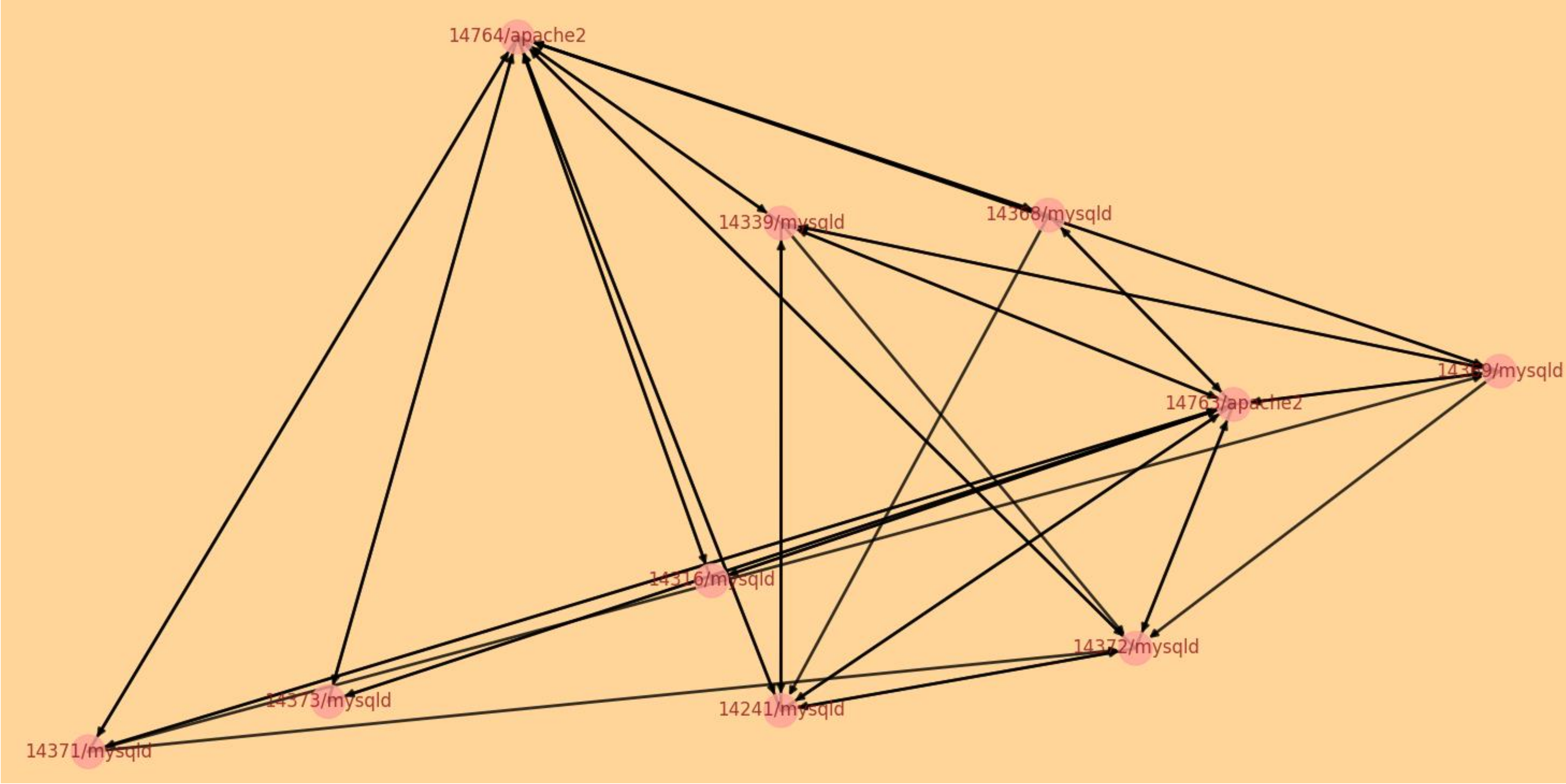


Use case I



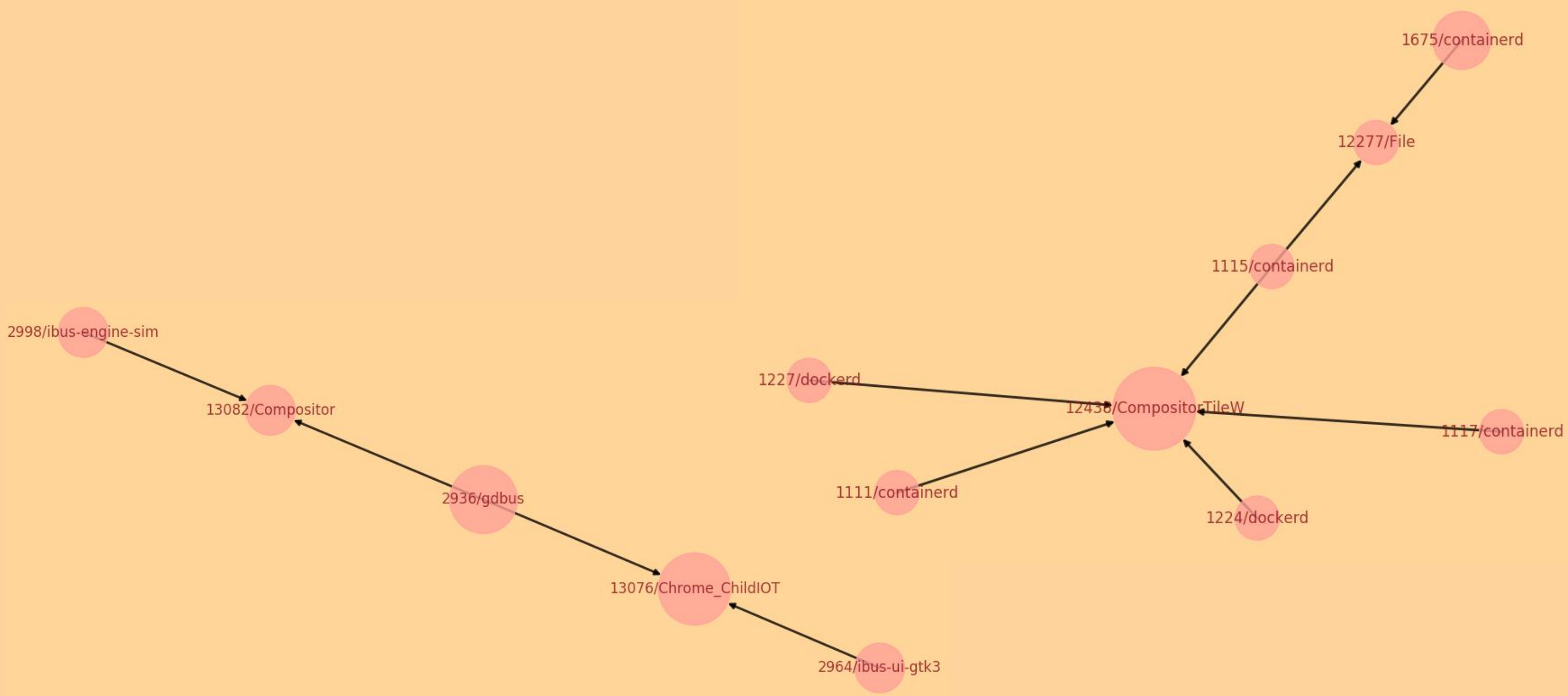
Use case I

Top-ranked processes in subgraph I



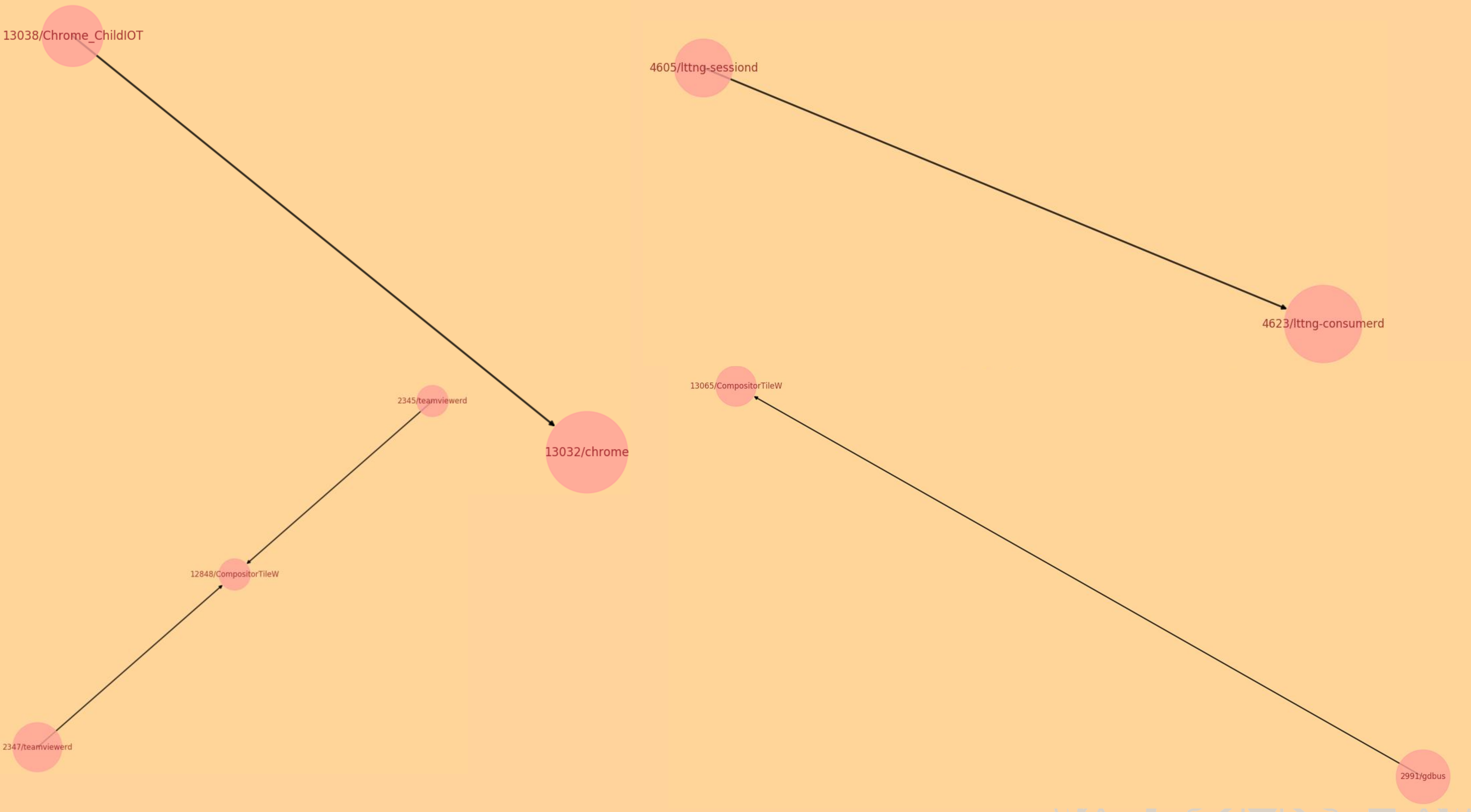
Use case I

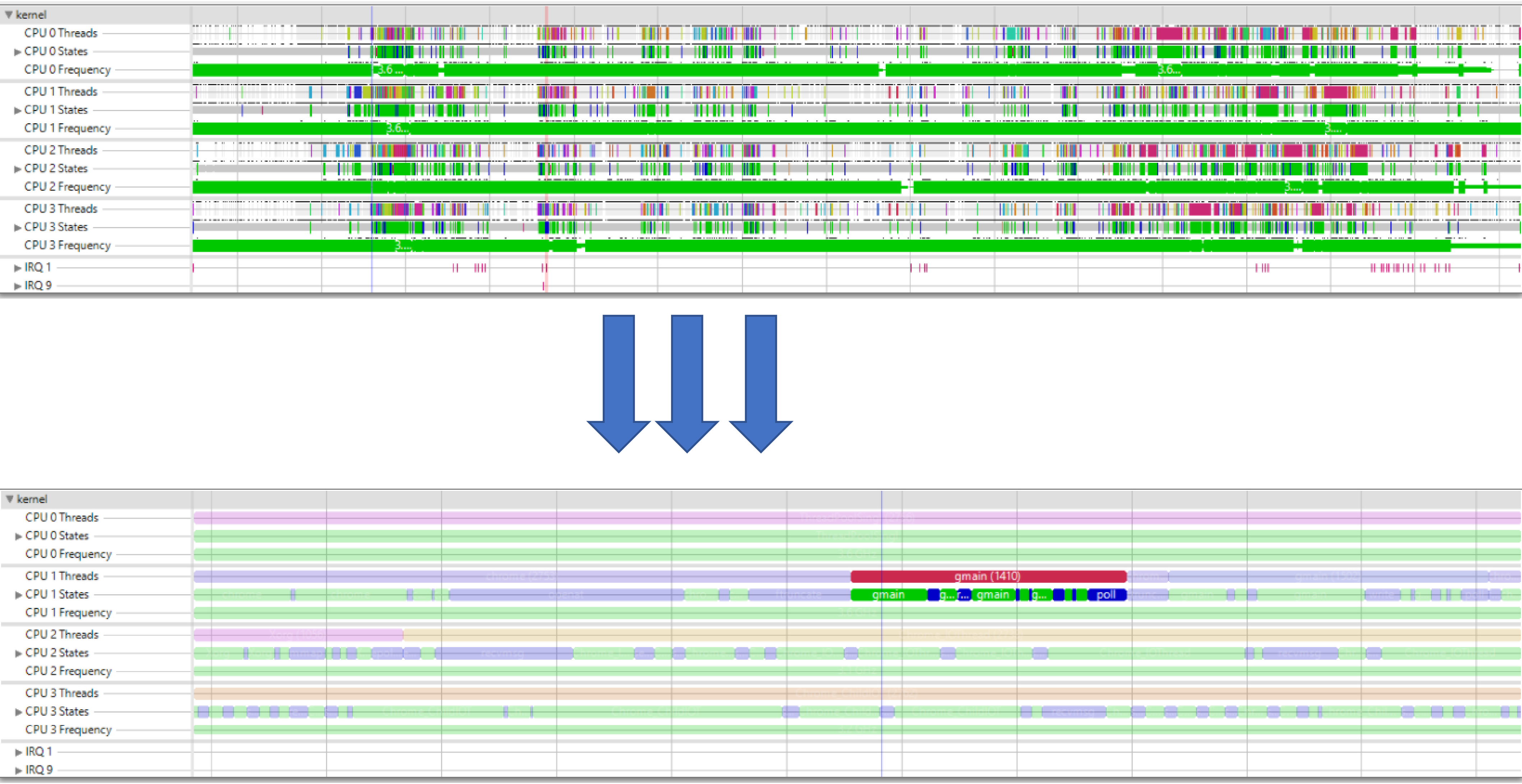
Top-ranked processes in subgraph II and III



Use case I

Top-ranked processes in other subgraphs:




kernel
CPU 0 Threads
CPU 0 States
CPU 0 Frequency
CPU 1 Threads
CPU 1 States
CPU 1 Frequency
CPU 2 Threads
CPU 2 States
CPU 2 Frequency
CPU 3 Threads
CPU 3 States
CPU 3 Frequency
IRQ 1
IRQ 9
kernel
CPU 0 Threads
CPU 0 States
CPU 0 Frequency
CPU 1 Threads
gmain (1410)
gmain
gmain
poll
CPU 1 States
CPU 1 Frequency
CPU 2 Threads
CPU 2 States
CPU 2 Frequency
CPU 3 Threads
CPU 3 States
CPU 3 Frequency
IRQ 1
IRQ 9

Conclusions & Future work

- Filtering based on process ranking
- Top interacting processes in one system
- Interaction between VM's processes in a cloud environment
- Time-window feature extraction for machine learning applications
- Enhance graph structure analysis, e.g., graph modularity

Questions

Thanks for your attention!

nezzati@brocku.ca