Filtering based on process ranking

Mahsa Shakeri

May 06, 2019

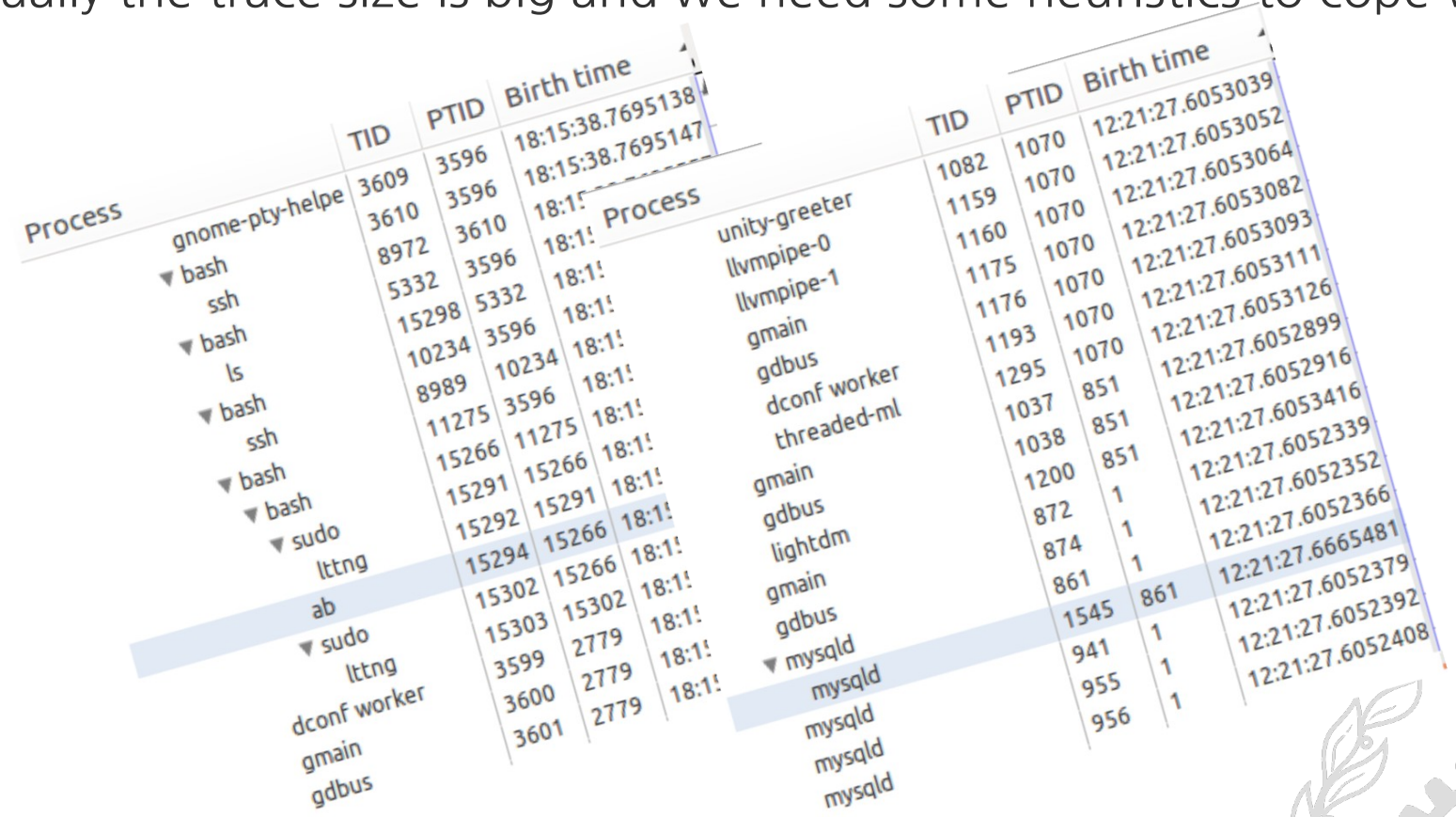
Polytechnique Montréal

DORSAL Lab

Introduction

Tracing is used for performance analysis and debugging.

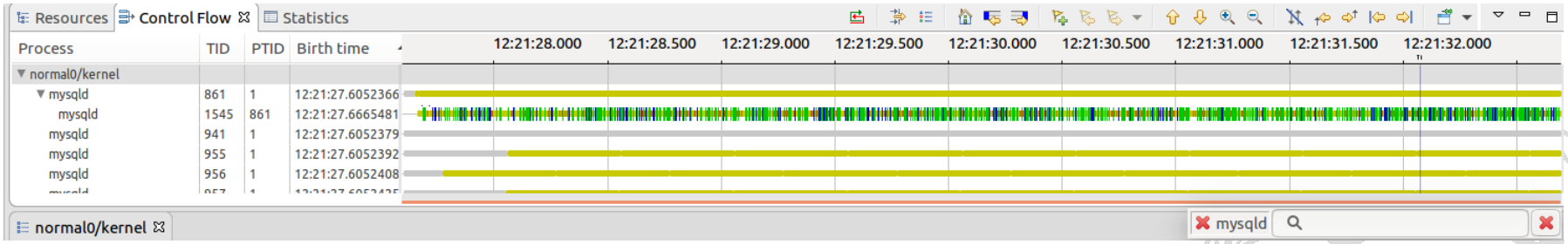
Usually the trace size is big and we need some heuristics to cope with it.



Current methods

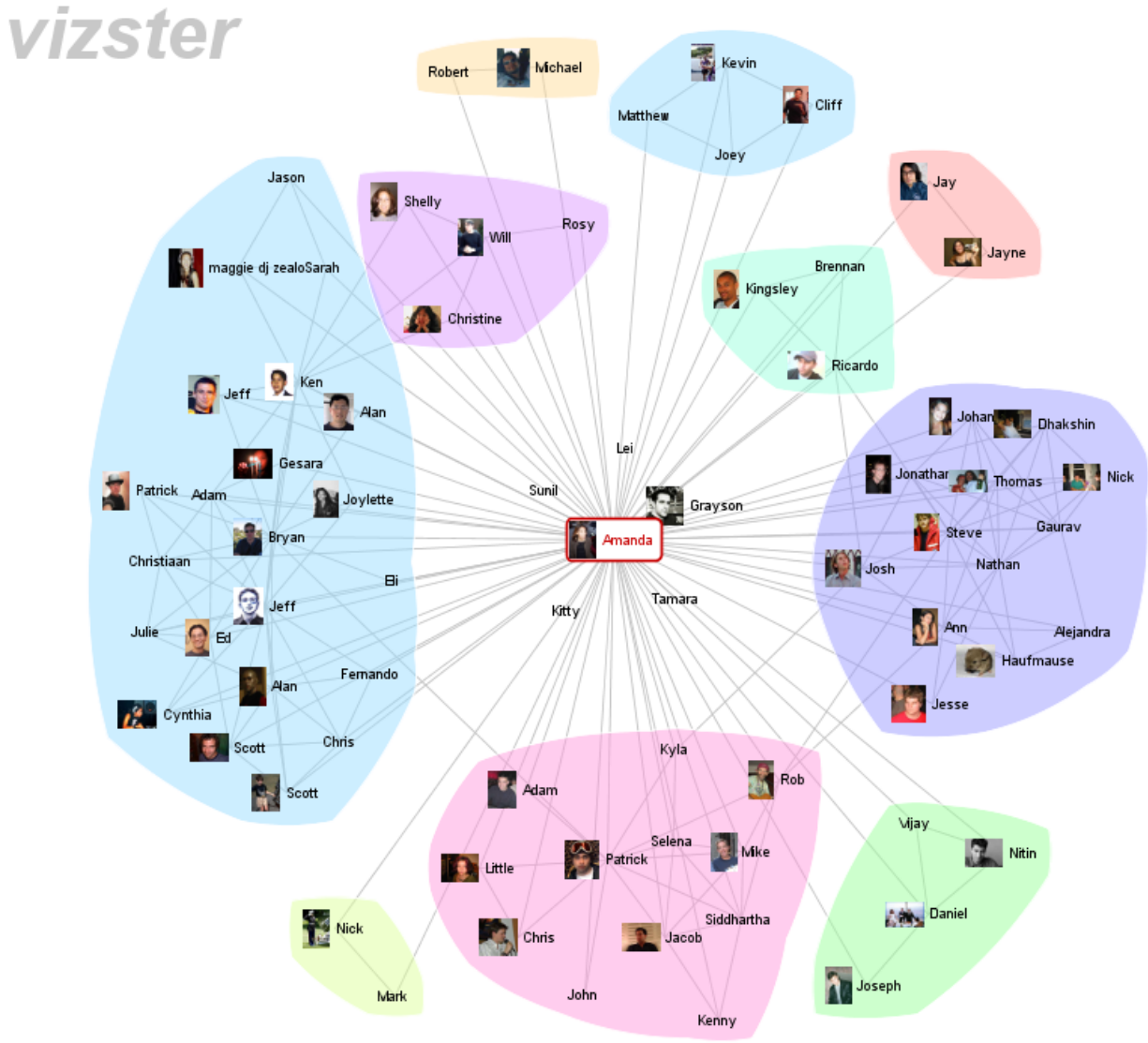
- Sampling or event filtering.
- Time view filtering.
- Global filtering

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=r
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

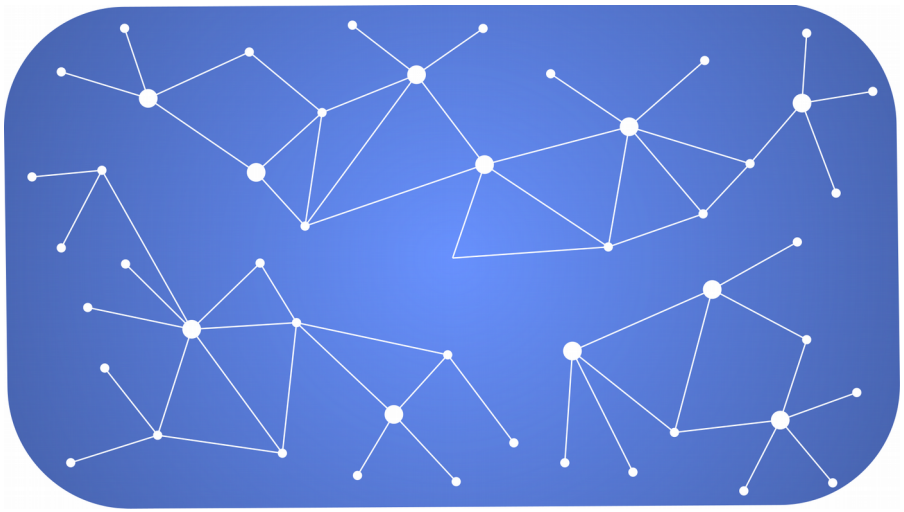


Picture from: http://vis.stanford.edu/jheer/projects/vizster/

Method

I.

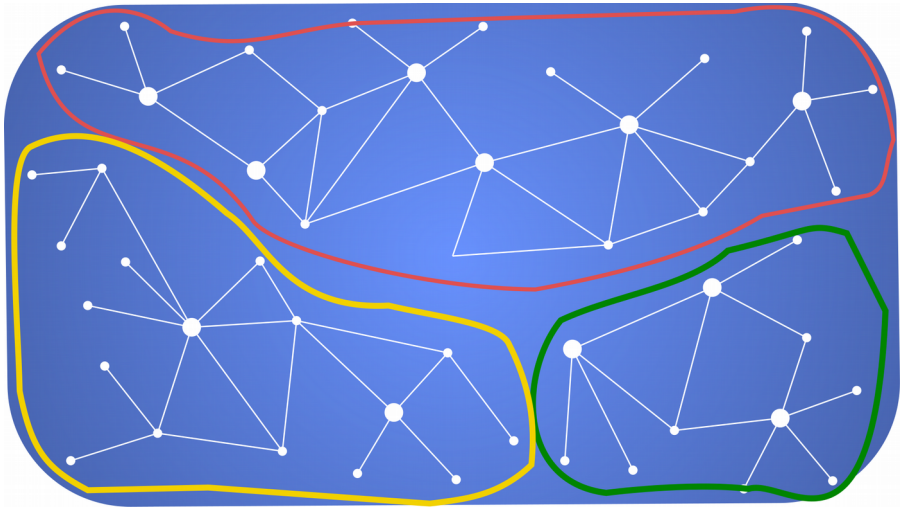
- Extract the processes interactions based on the execution graph.
- Building a graph G = (V,E)
- V includes all the processes running on the system
- E is a set of edges defined based on the interactions between the processes.



Method

II.

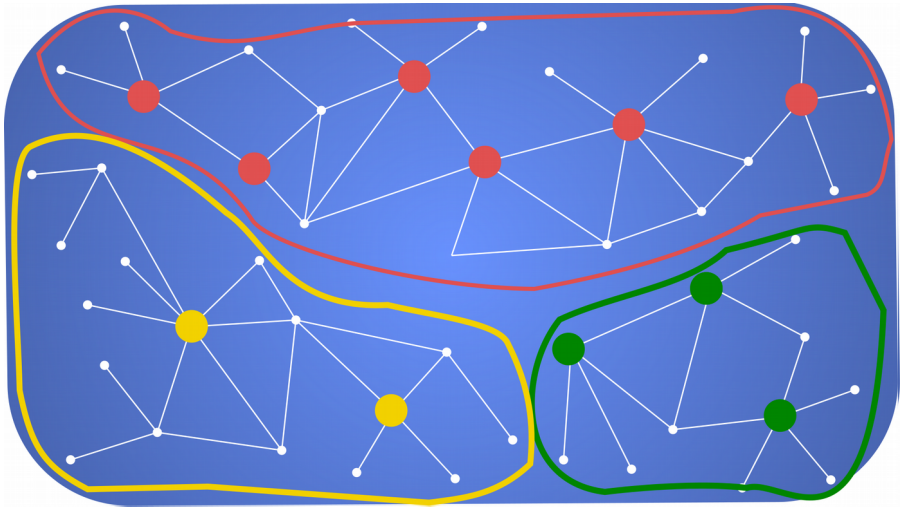
Compute connected components on the graph to extract all the subgraphs. (e.g., BFS)



Method

III.

- Apply a ranking algorithm on each connected component to assign a rank to each process
- Choose the top-K processes..

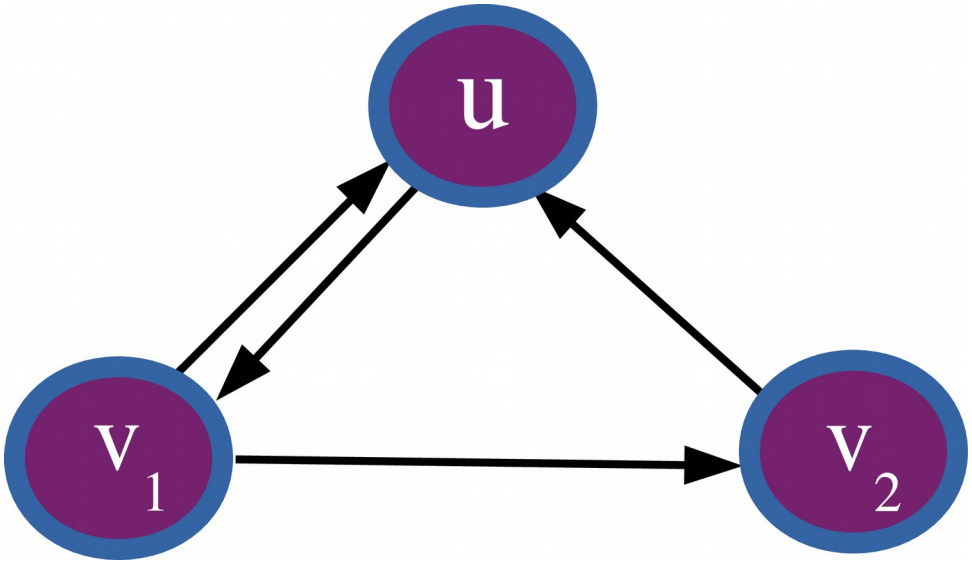


Method

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

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

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



u: a process

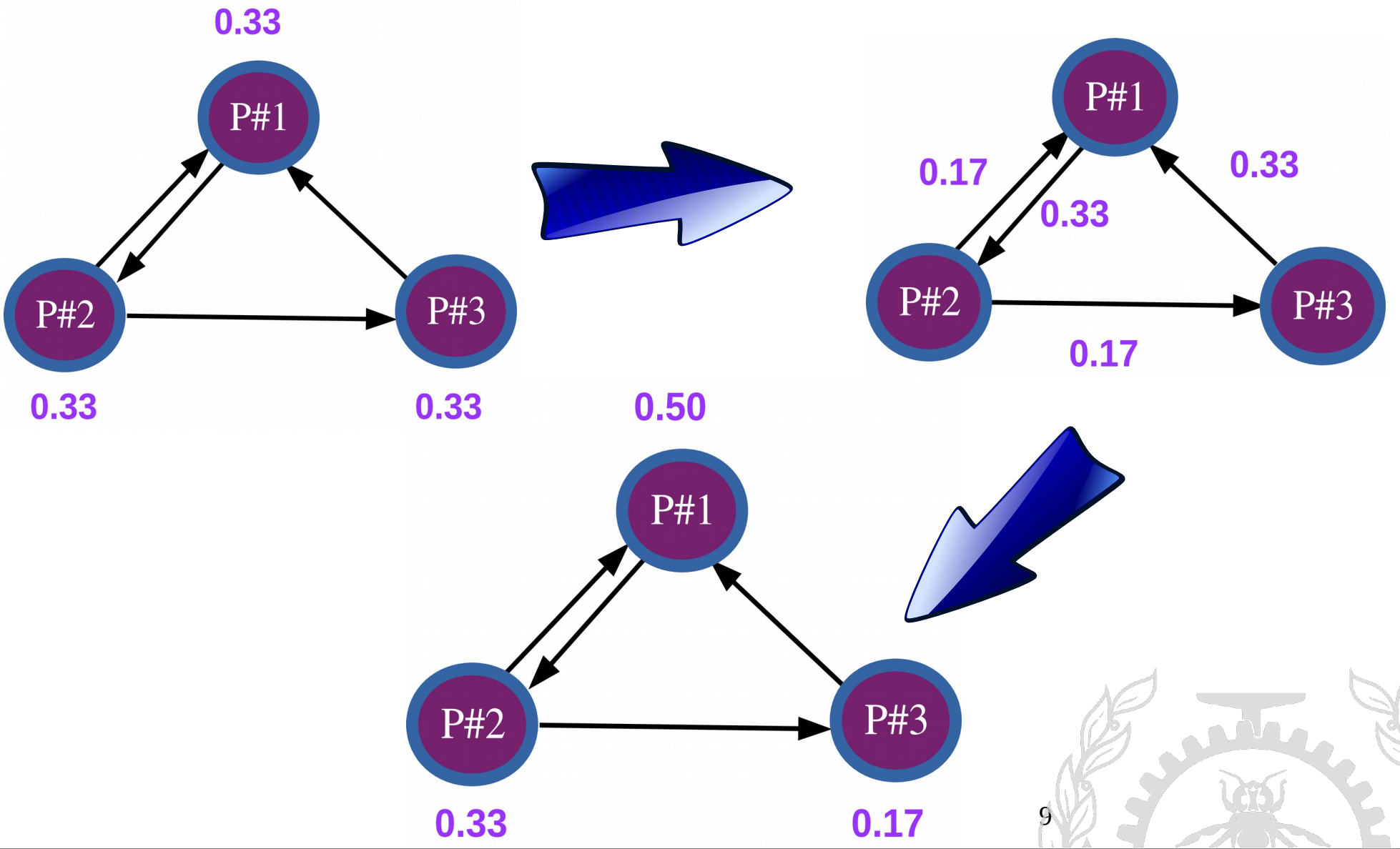
B_u: the set of u's in-edges

N_v: the number of out-edges of process v

c: the normalization factor

Method

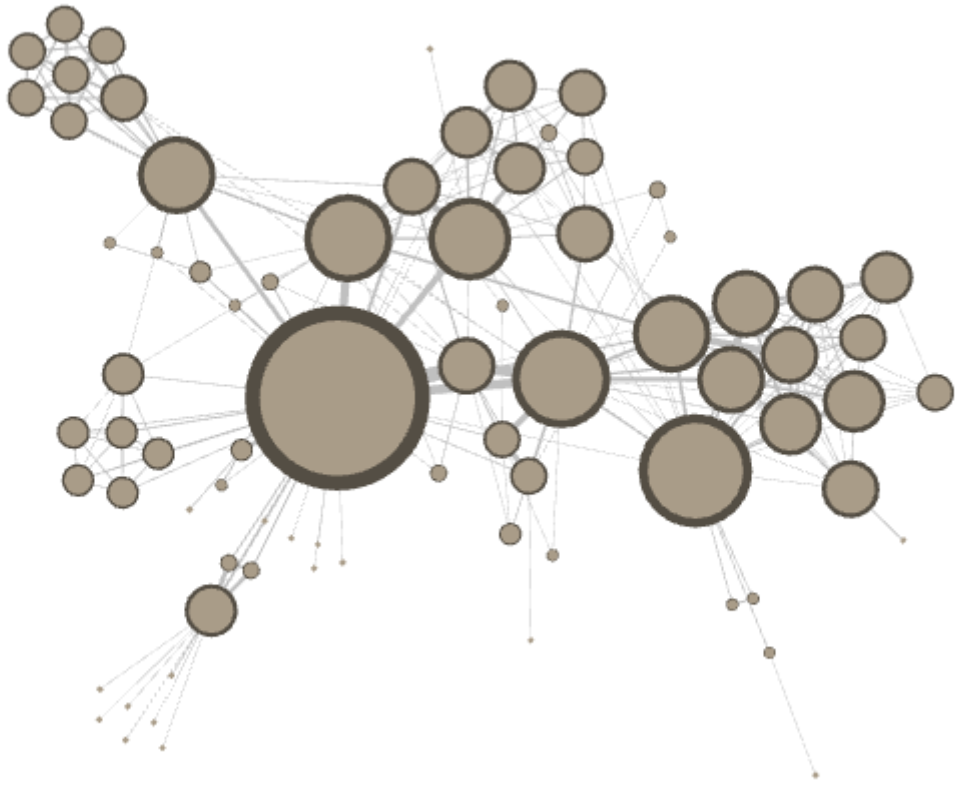
Modified PageRank algorithm:



Method

NB. Computing connected components on the graph

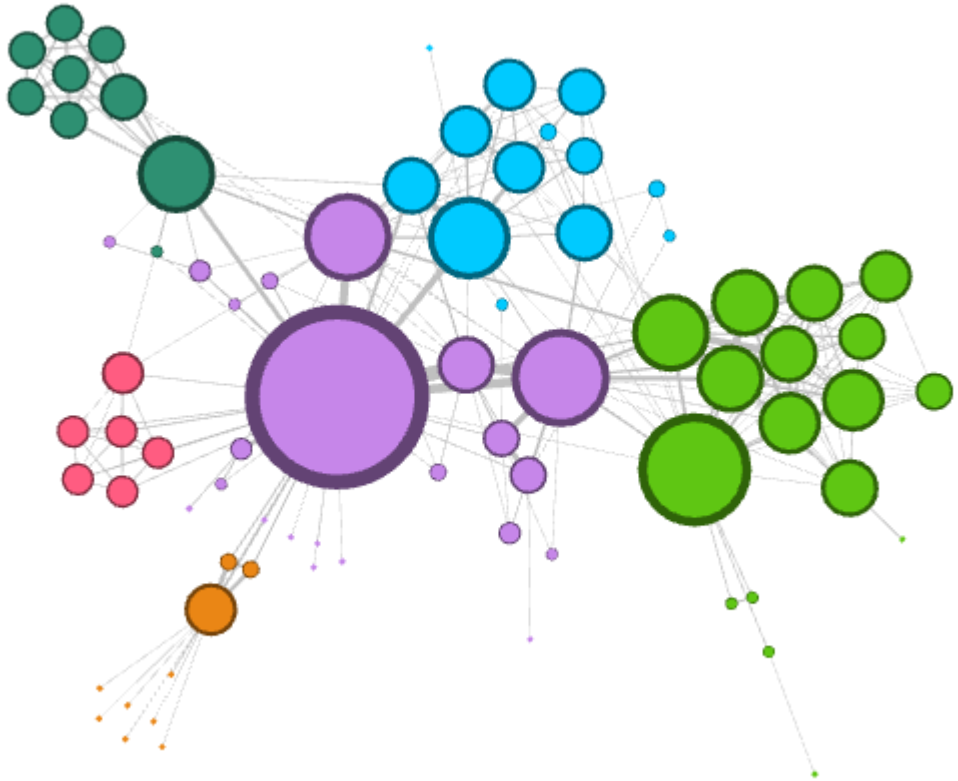
Other approaches:
e.g., Graph Modularity



Method

NB. Computing connected components on the graph

Other approaches:
e.g., Graph Modularity



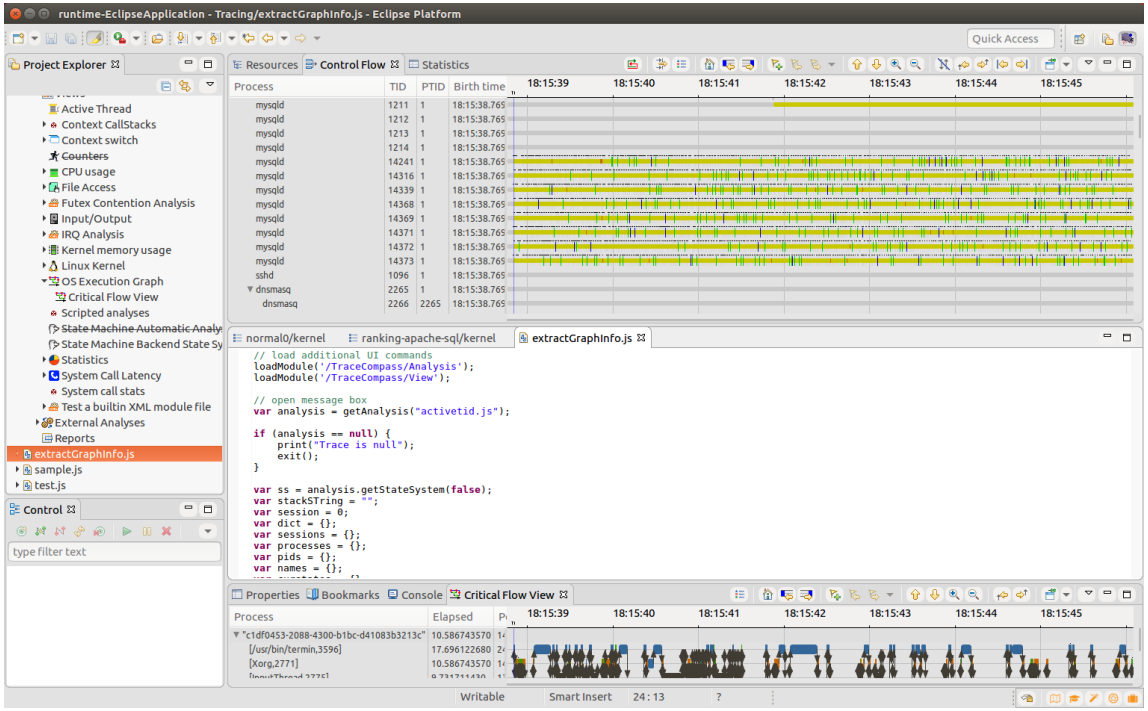
Use case I

Trace Compass:

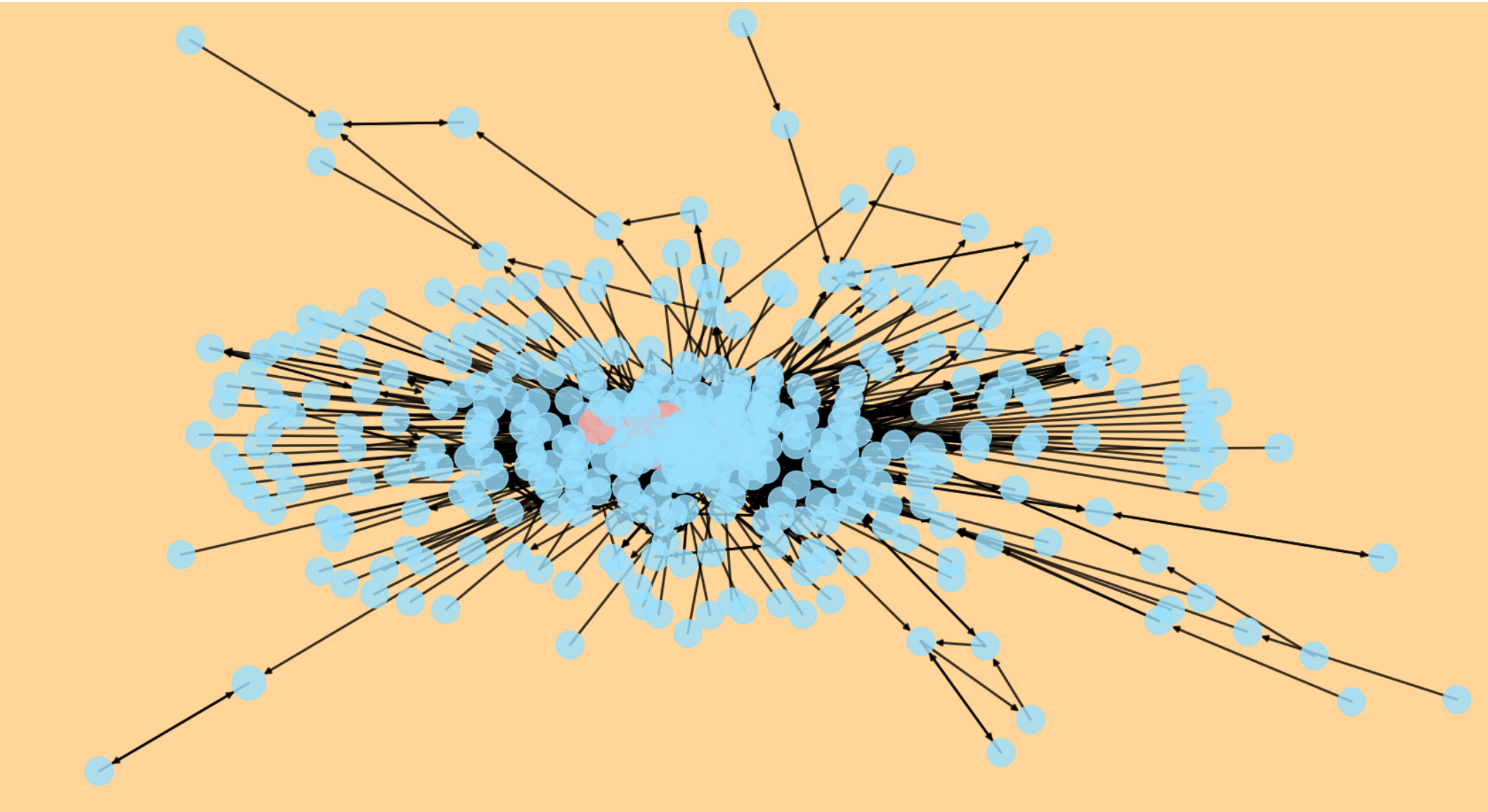
Extract process interactions using EASE scripting.

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

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

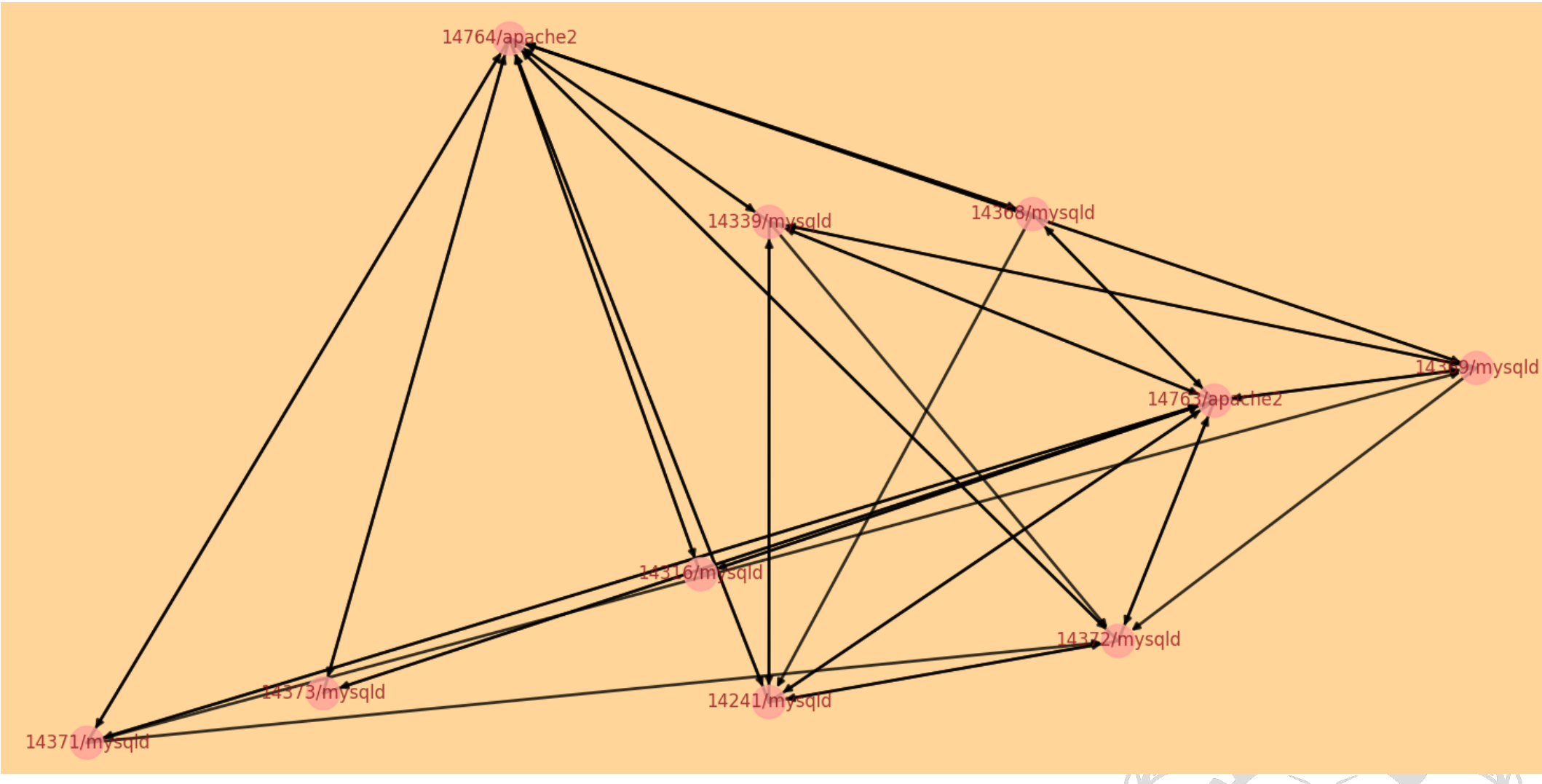


Use case I



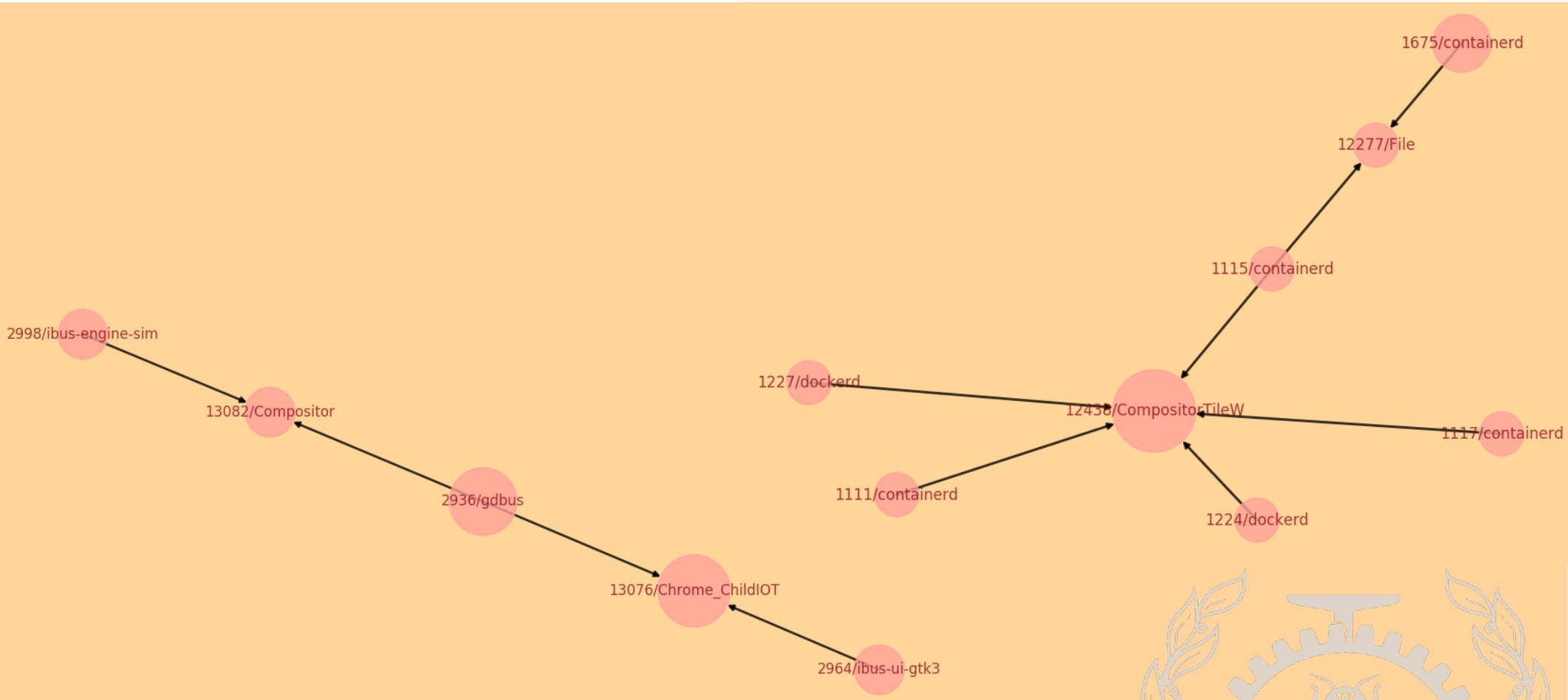
Use case I

Top-ranked processes in subgraph I



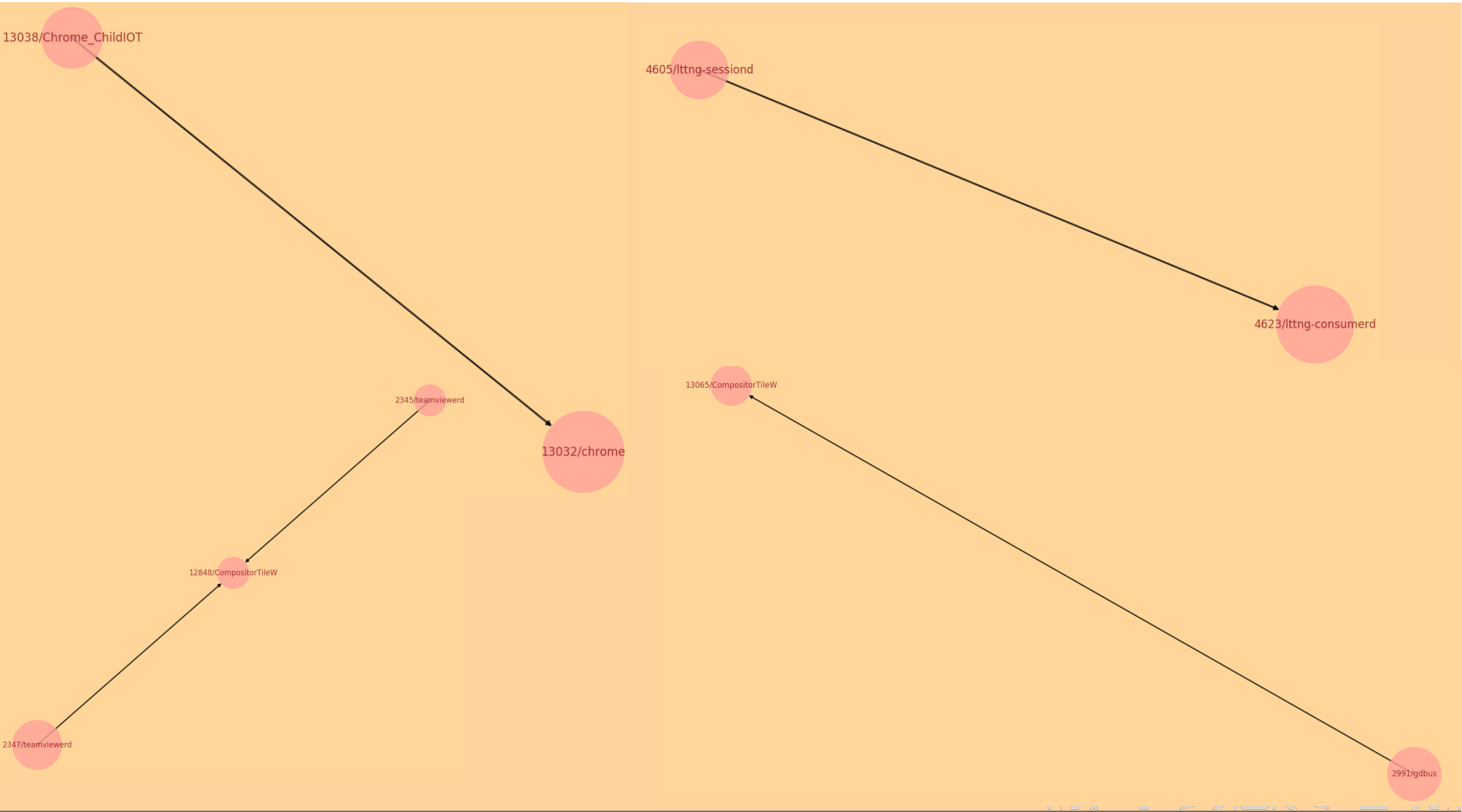
Use case I

Top-ranked processes in subgraph II and III



Use case I

Top-ranked processes in other subgraphs:

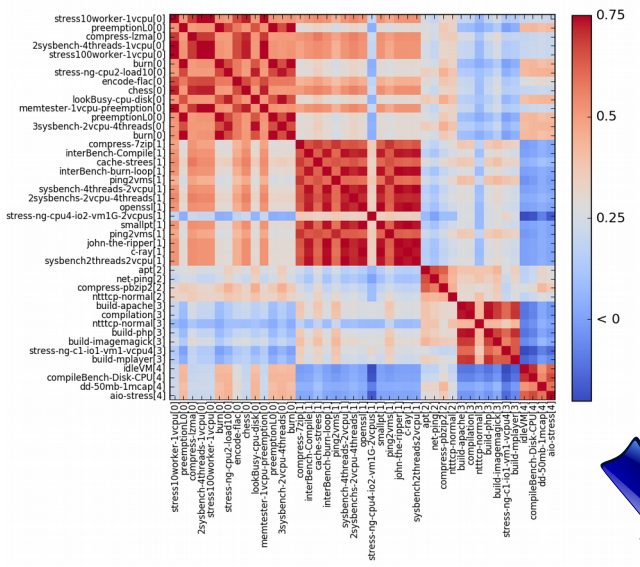


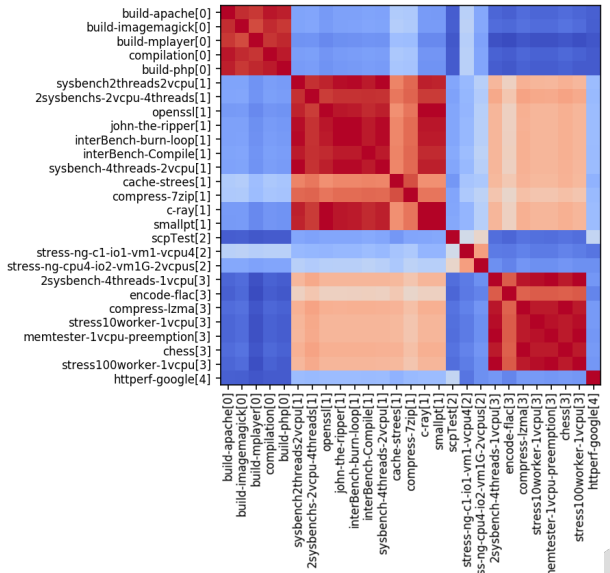
Use case II

VM clustering[1] :

Identifying the top important processes running on each VM.

Filtering out the irrelevant processes to provide a more distinct VM workload clustering.

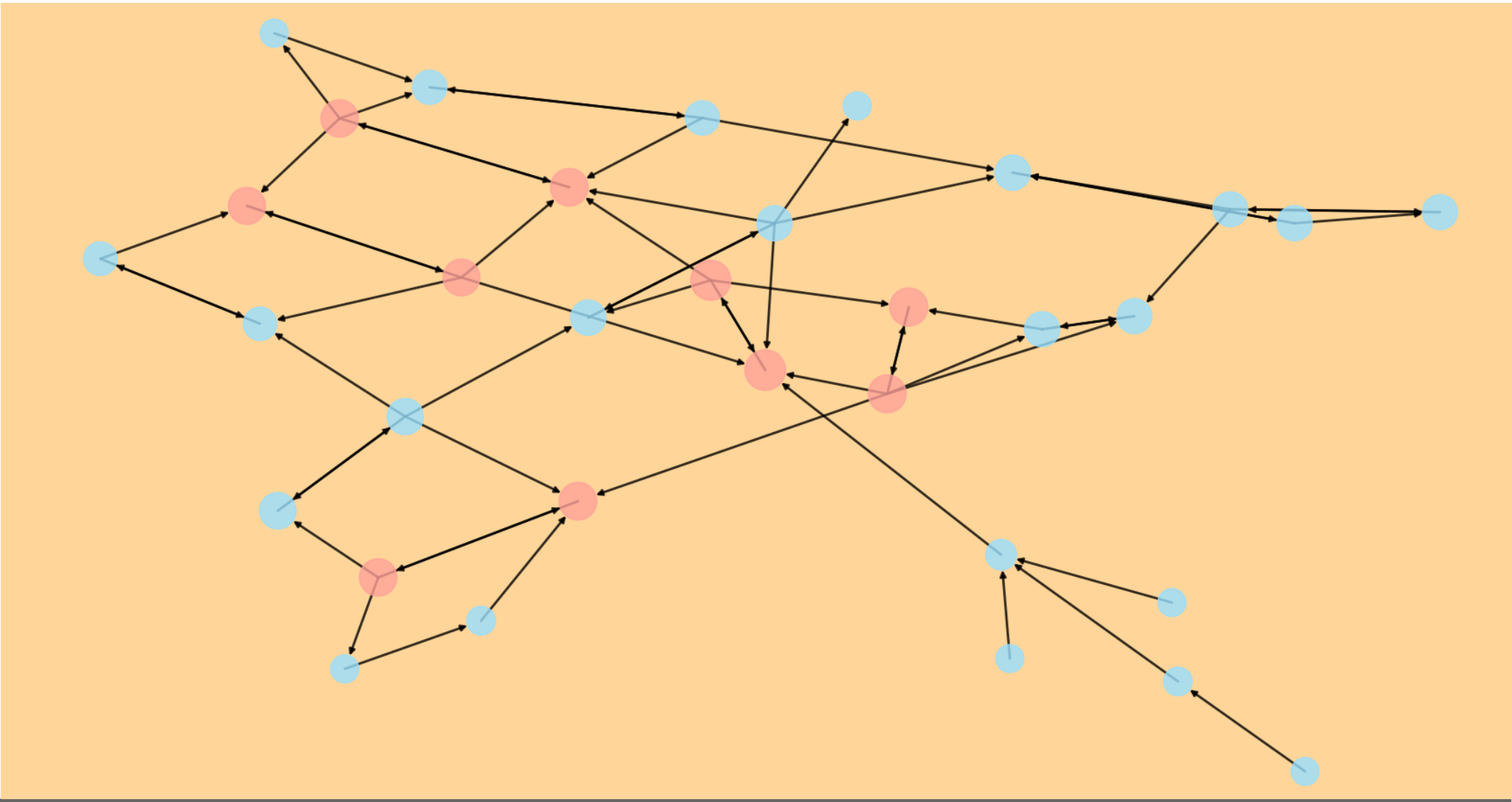




[1]Nemati H., Azhari V., et al., Host-based Virtual Machine Workload Characterization using Hypervisor Trace Mining, ACM Trans 2019

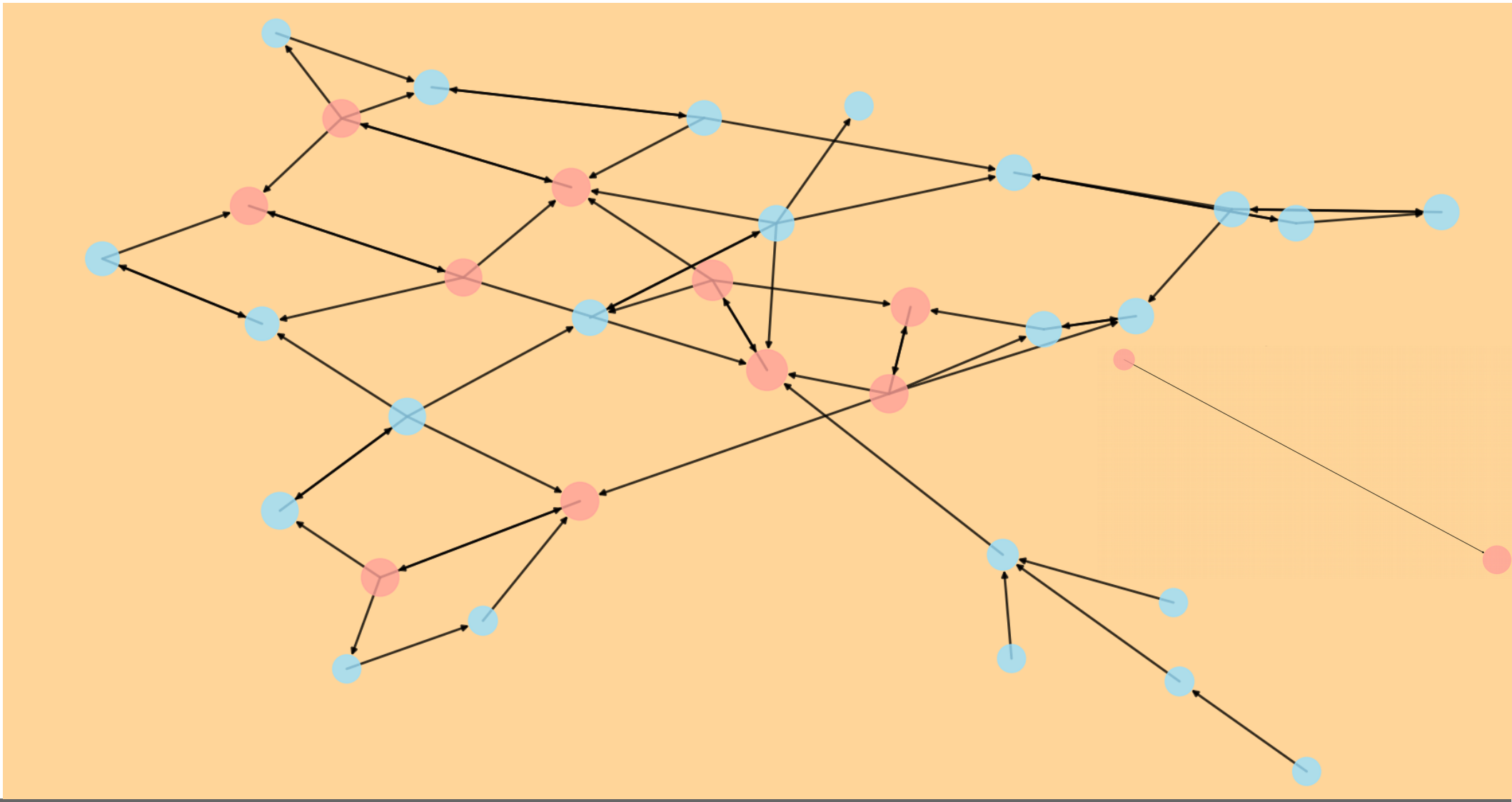
Use case II

Subgraph I:



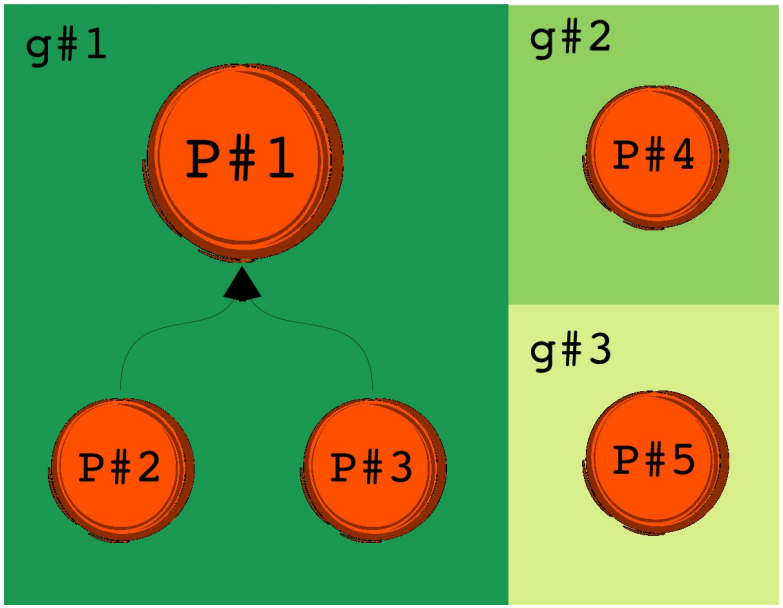
Use case II

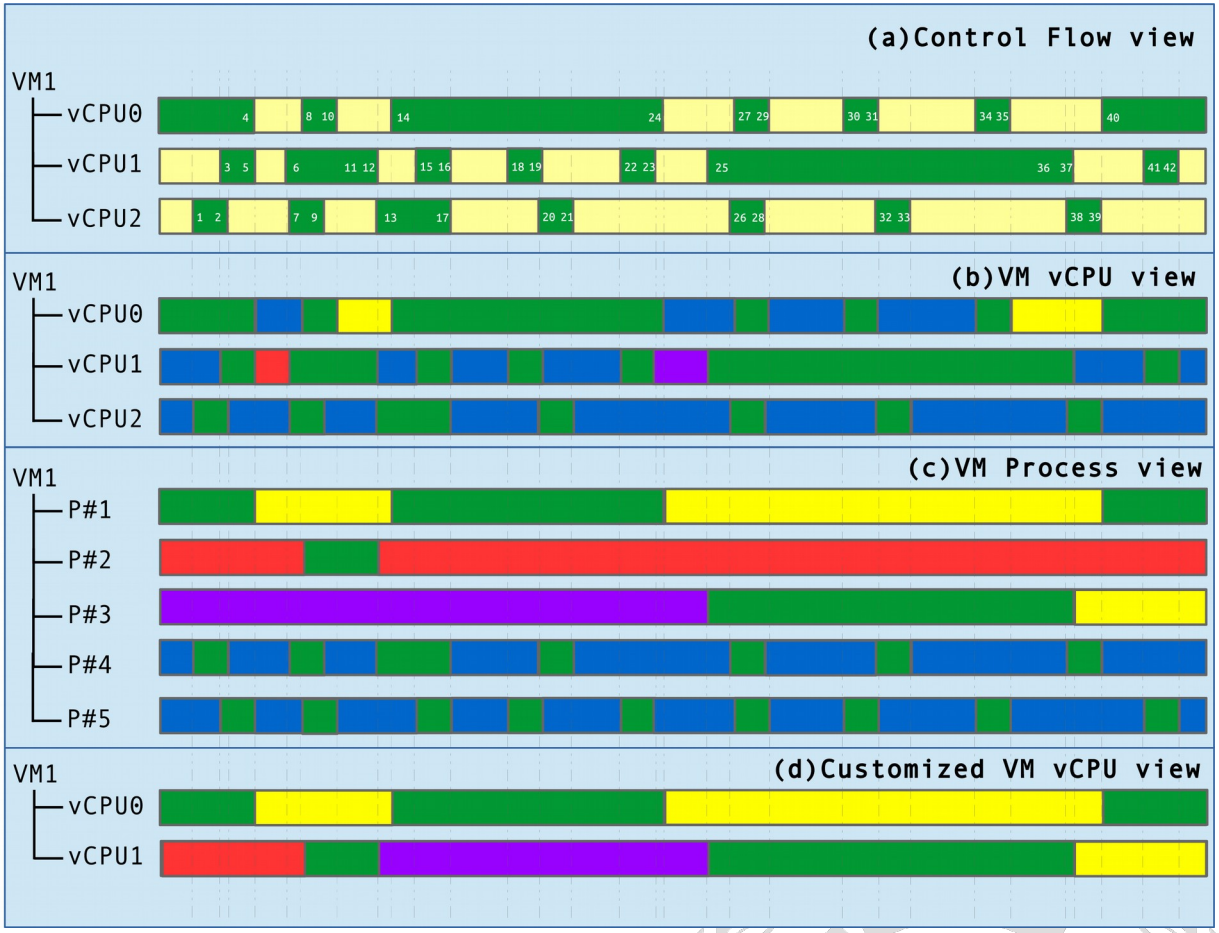
Subgraph I and II:



Use case II

- A **customized vCPU view** based on active processes
- **Example:**





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!

Appendix

Appendix

- VM

Algorithm 2: Process Ranking Algorithm (PRA)

```
1  if event == sched_ttwu then
2      j = getVMvCPU(wakee_tid);
3      k = getVMvCPU(waker_tid);
4      wakerCR3 = getLastCR3(vCPU_k);
5      updateWaker(vCPU_j, wakerCR3);
6  else if event == vm_inj_virq then
7      j = getVMvCPU(tid);
8      pCR3 = getVMvCPU(vCPU_j);
9      if vec == IPI then
10         wakerCR3 = queryWaker(vCPU_j);
11         LINK_HORIZONTAL(wakerCR3, pCR3);
12 for all process cr3_i ∈ CR3 do
13     R(cr3_i);
14 connected_subgraph = breadth-first-search(CR3);
15 customized_graph = denoise(connected_subgraph, R(CR3));
```