Multi Level Trace Events Linking, Storage and Display

Naser Ezzati Jivan
Michel Dagenais
Department of Computer and Software Engineering

December 9, 2011
École Polytechnique, Montreal

Motivations

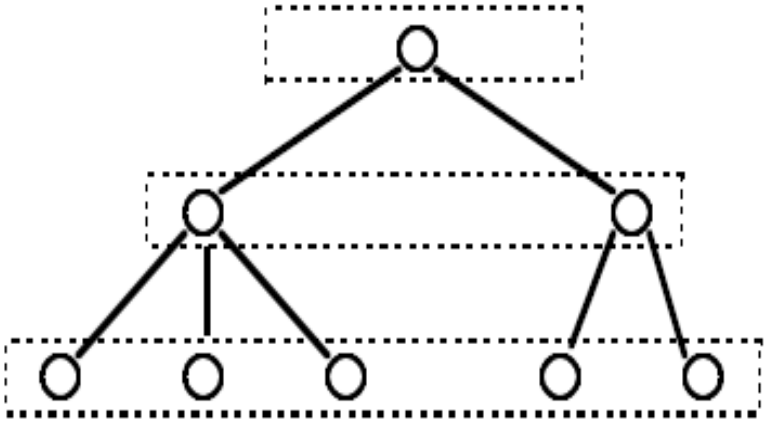
- Tracing systems provides information at several levels: Operating system, virtual machine, user space.
- There are several modules that generate higher level analytical events from the raw trace events.
  - Trace Abstraction , Trace directed modeling, Automated fault identification

Motivations

- For better understanding the system, users need to access easily and efficiently to all of these multi level information.
- Users must be able to navigate from raw events to the new information generated by the analysis modules.

Different levels of detail

- There are many levels in the hierarchy.
- Most analysts start with an overview of the system before refining their view to be more detailed.
- The highest level is the most abstract level of the hierarchy, and can be an overview of the system
- the lowest level is the most detailed level (raw events)



How to create the hierarchy?

1- Pattern matching to create compound events

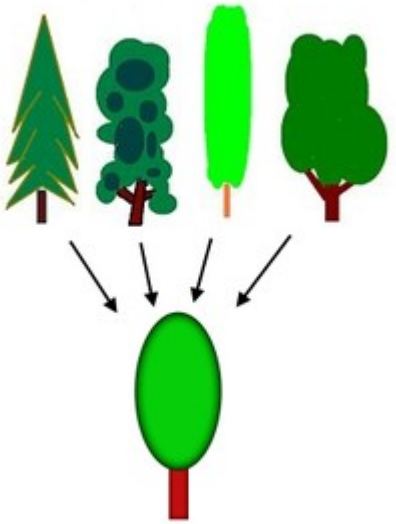
- Constructed by aggregating set of low level events to single compound event using pattern matching techniques.
- Example: highlighted area can be aggregated to a “file write” event.

```
kernel.sched_schedule: 245.173141831 (project/kernel_1, 1874, 613, rs:main Q:Reg, , 1, 0x0, MAYBE_USER_MODE { prev_pid = 0, next_pid = 1874, prev_state = 0 }
mm.page_free: 245.173143761 (project/mm_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, MAYBE_USER_MODE { pfn = 1617390, order = 0 }
kernel.syscall_exit: 245.173146346 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, MAYBE_USER_MODE { ret = 0 }
kernel.syscall_entry: 245.173148681 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { ip = 0x7f4a267bf955, syscall_id = 202 [syscall 202] }
kernel.syscall_exit: 245.173149576 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, MAYBE_USER_MODE { ret = 0 }
kernel.syscall_entry: 245.173165061 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { ip = 0x7f4a267bfc5d, syscall_id = 1 [syscall 1] }
fs.write: 245.173179378 (project/fs_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { count = 66, fd = 4 }
kernel.syscall_exit: 245.173180026 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, MAYBE_USER_MODE { ret = 66 }
kernel.syscall_entry: 245.173183408 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { ip = 0x7f4a267bfc5d, syscall_id = 1 [syscall 1] }
fs.write: 245.173188953 (project/fs_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { count = 66, fd = 1 }
kernel.syscall_exit: 245.173189260 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, MAYBE_USER_MODE { ret = 66 }
kernel.syscall_entry: 245.173207223 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { ip = 0x7f4a267bfc5d, syscall_id = 1 [syscall 1] }
fs.write: 245.173210895 (project/fs_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { count = 73, fd = 4 }
kernel.syscall_exit: 245.173211185 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, MAYBE_USER_MODE { ret = 73 }
kernel.syscall_entry: 245.173213390 (project/kernel_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { ip = 0x7f4a267bfc5d, syscall_id = 1 [syscall 1] }
fs.write: 245.173216020 (project/fs_1), 1874, 613, rs:main Q:Reg, , 1, 0x0, SYSCALL { count = 73, fd = 1 }
```

How to create the hierarchy?

2- Generalization

- relates a class to a superclass

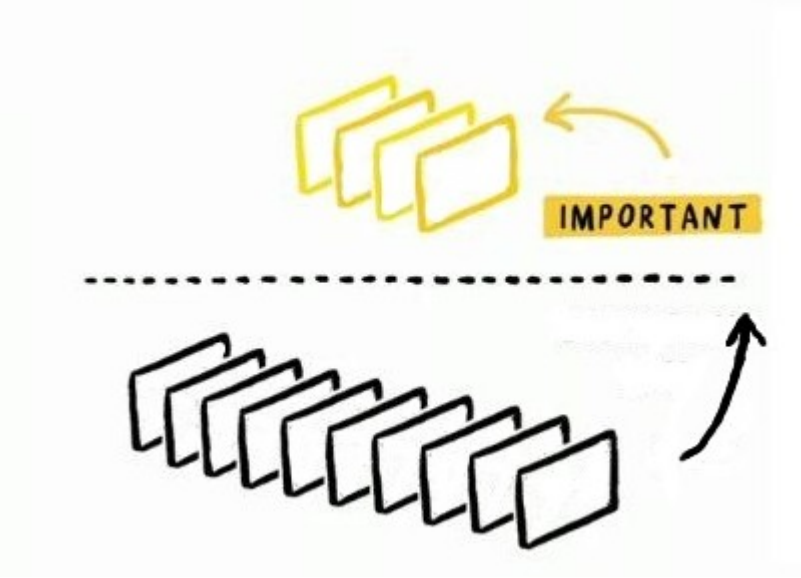


Raw Events	Generalized events	More Generalized
Socket read	Socket read	Read
File read File readv File pread64	File read	Read
Socket send Socket write	Socket send	Write
File write File writev File pwrite64	File write	Write

How to create the hierarchy?

3- Filtering

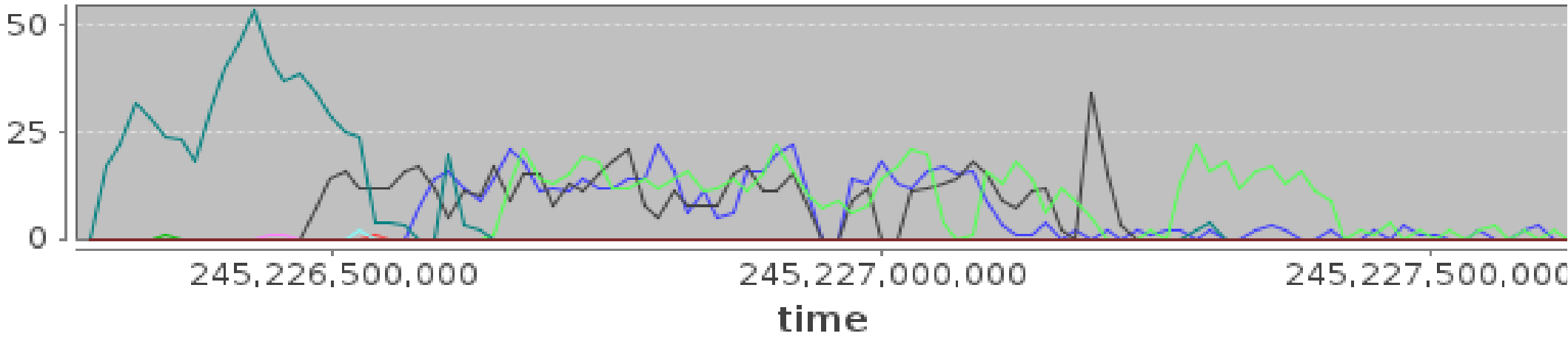
- Filter some base events and generates a subset of events on a higher level.
- Filtering is based on predefined priority and weights.



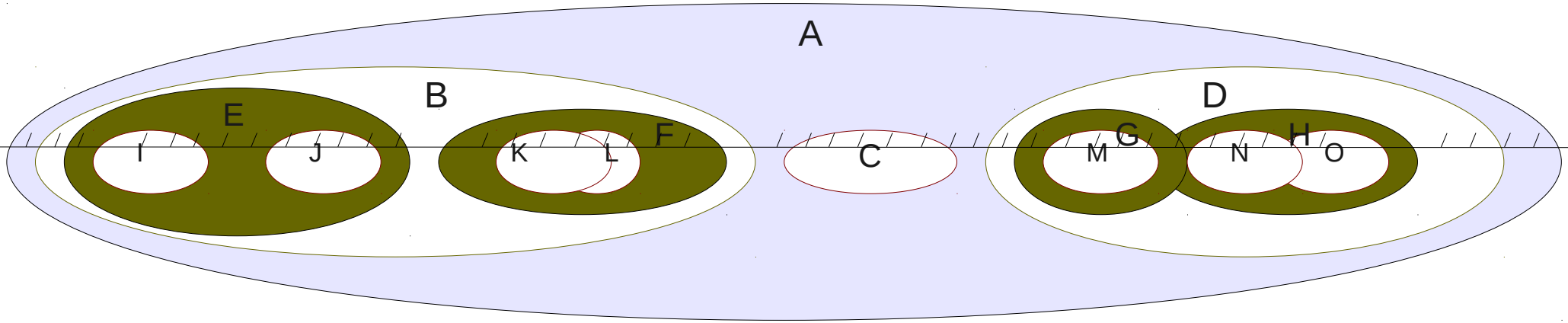
How to create the hierarchy?

4- Make statistics as an overview of the system

- Statistics of important system metrics
  - CPU usage
  - Number of connections
  - Number of open files
  - I/O throughput
  - Number of faults
  - ...



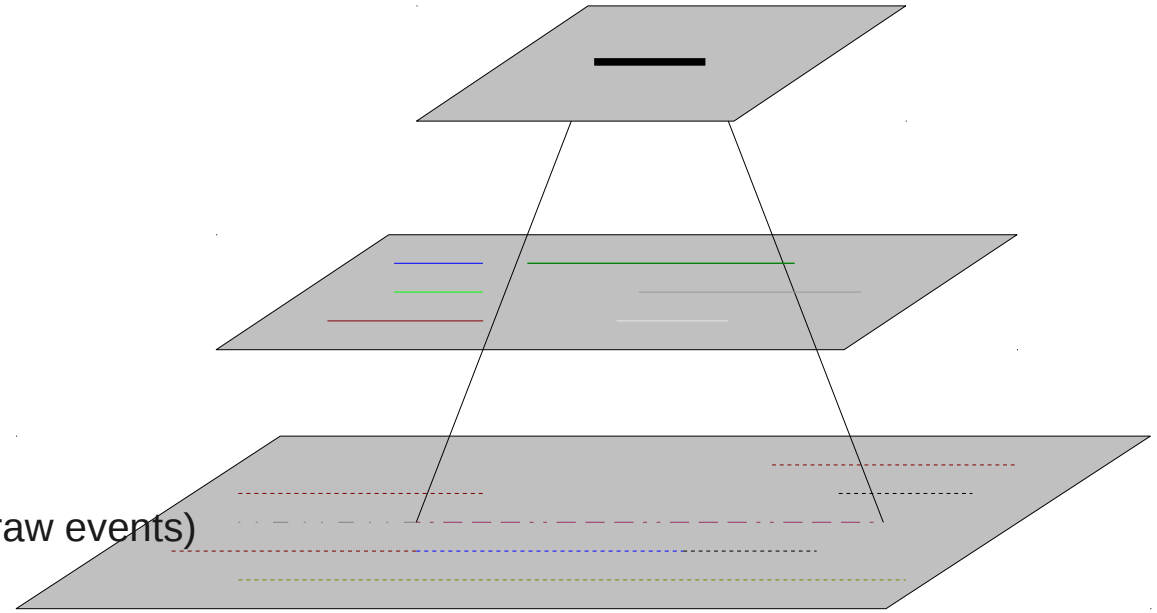
Linking different levels of the hierarchy



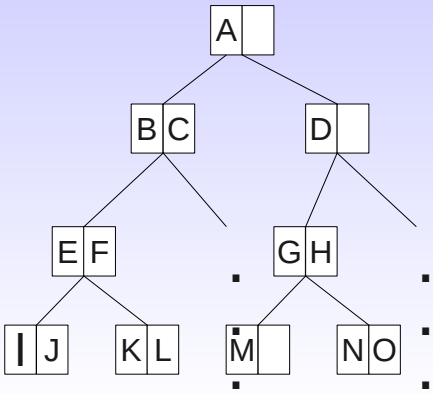
Level 0

Level 1

Level 2(raw events)

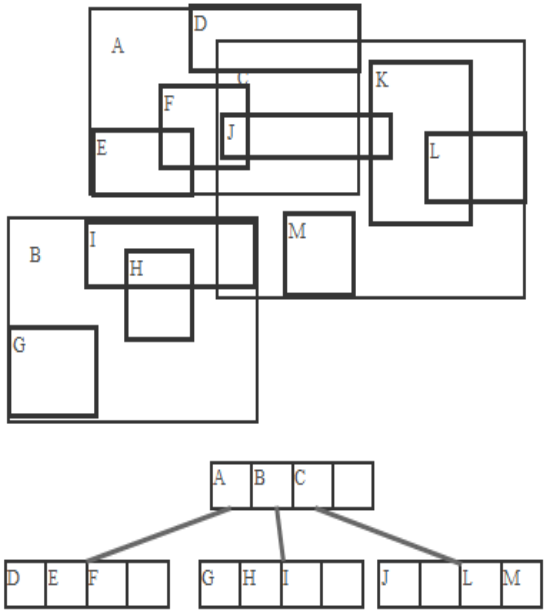


Multi-Dimensional
Link Data Structure



Indexing

- Good storage purposes,
- Efficient enough to work with large data set



Data Structures

- R-tree is a special index that designed for doing range queries.
- Organize spatial objects into k-dimensional rectangles. Each node in the tree corresponds to smallest k-dimensional rectangle that encloses child nodes.
- If an object is spatially contained in several nodes, it is only stored in one node.
- Problem is that to find some object you might have to go through several rectangles or whole database.
- R-trees are most commonly used in spatial systems where each entry is a rectangle.
- R+-trees: try not to overlap the rectangles, the overlapping objects appear in all of the rectangles.
-

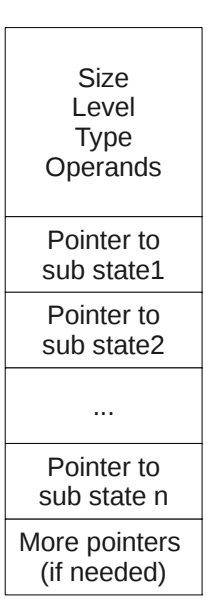
Implementation

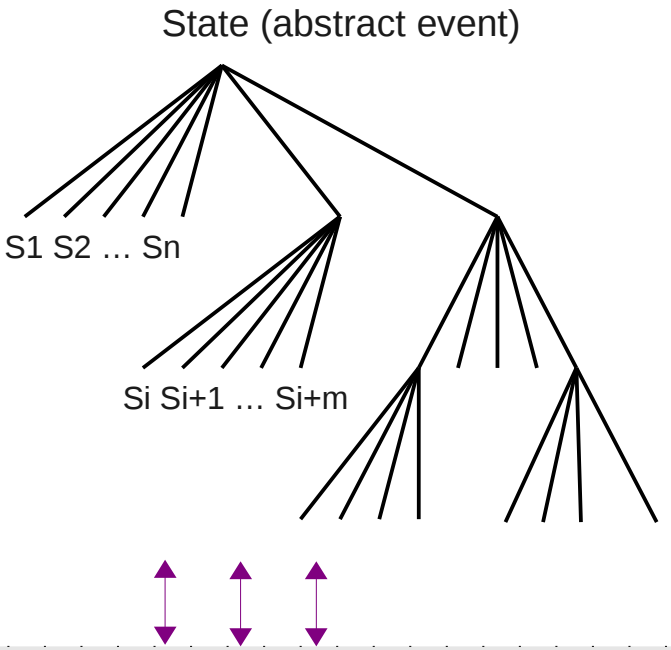
We have implemented the data structure as an extension to the “state system”

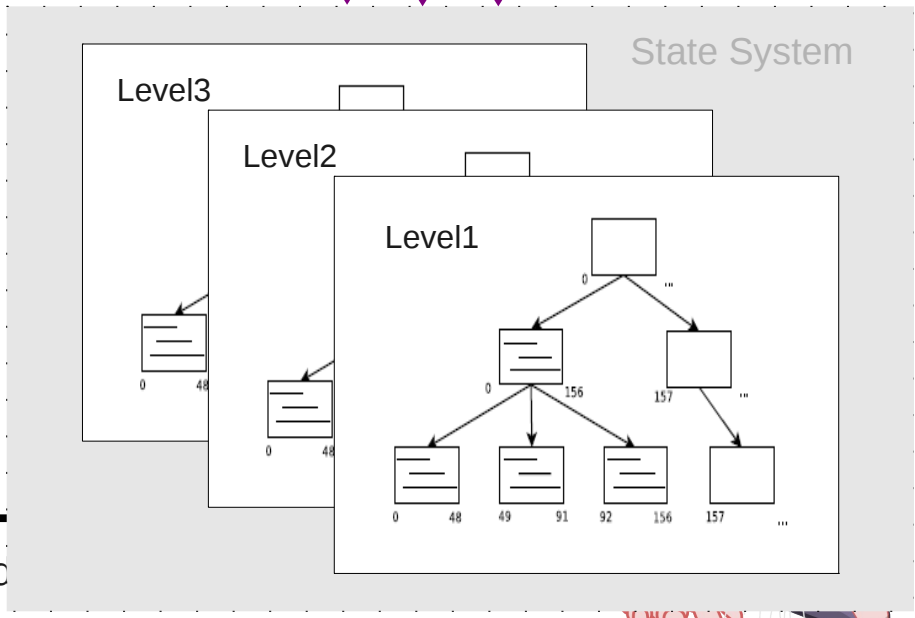
- Two approaches:
  - Storing all the abstract events
    - Once we created and stored the abstract events, we can use them without reading the trace again.
    - We model the abstract events as states
      - When there is a “connection” abstract event for a process P1: The state of the attribute P1 is “connected”
  - Storing the intermediate states (needed by pattern matching) and enough information for regenerating the abstract events
    - We store enough information about the intermediate states so that we can re-generate abstract events for any given time range of the trace

Implementation

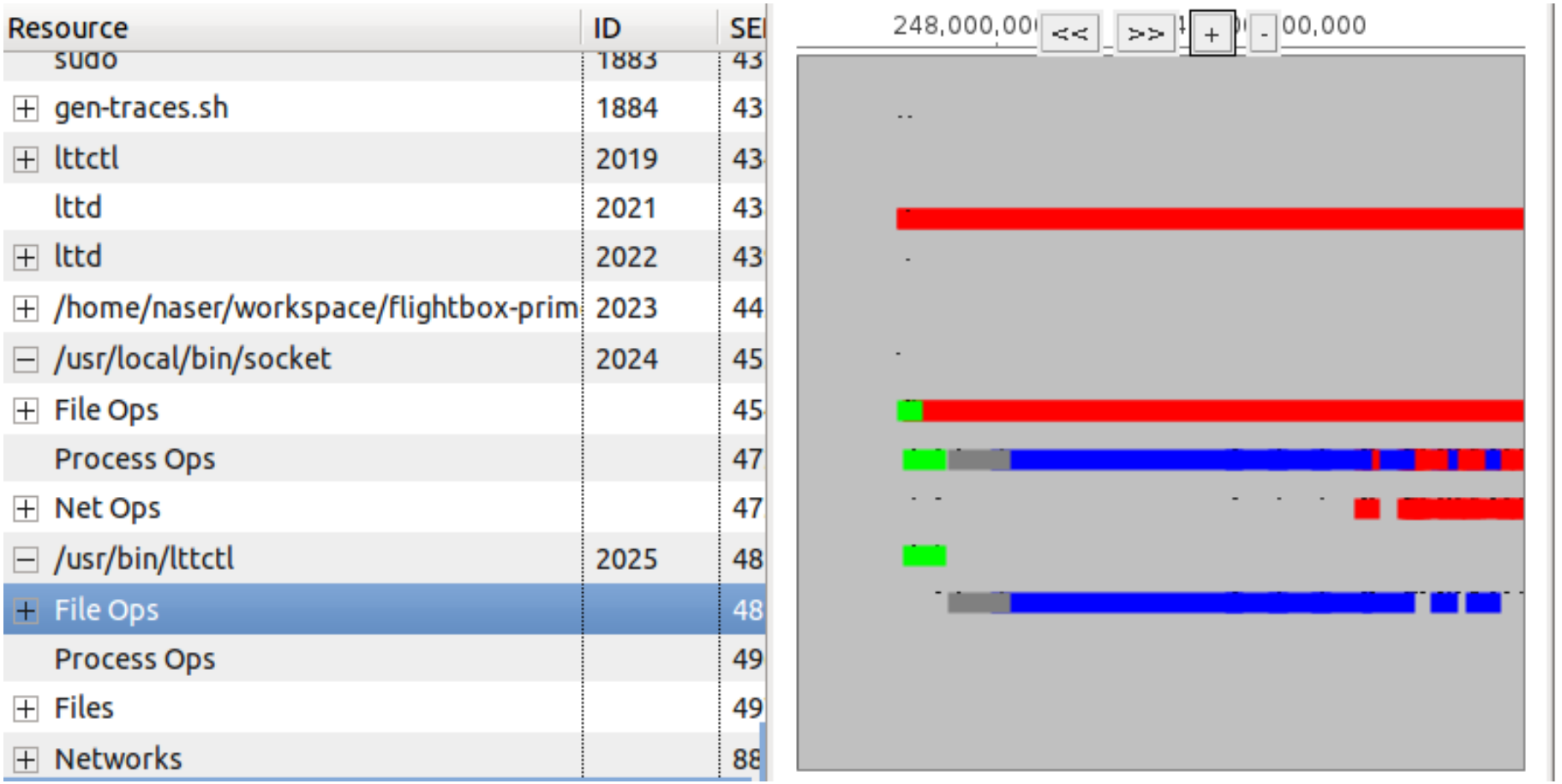
- The data structure supports multi levels of states:
  - A process open a file (state of the process at this level is “opening a file”)
  - At a higher level it can be member of a “sequential file read/write” event. Thus state of the process will be “sequentially reading a file” in the higher level
- States keep track of addresses of their children and also their direct parent







Demo


Resource
ID
SE
sudo
1883
43
gen-traces.sh
1884
43
lttctl
2019
43
lttd
2021
43
lttd
2022
43
/home/naser/workspace/flightbox-prim
2023
44
/usr/local/bin/socket
2024
45
File Ops
45
Process Ops
47
Net Ops
47
/usr/bin/lttctl
2025
48
File Ops
48
Process Ops
49
Files
49
Networks
88
248,000,00
00,000

Levels

- System-call level (open, read, send, …)
- Classification of system calls
- Operation level (TCP connection, HTTP request, DNS request, a file download, a port scan, sequentially file read and …)
- Operation type: file operation, network operation, …
- Statistical overview

Visualization

- We are using “timeline” to visualize the states (abstract events).
- States are shown by two colors:
  - One color for the operation (read, write, send, open, fork, ...)
  - One color for its type (file operation, net operation, …)
- Visual Operations:
  - Focusing
    - Showing relevant information of a specific event
    - Two dimensional focusing:
      - select a region in time
      - select a process (resource) or group of them
  - Zooming
    - Showing more details(Google Map) and enlarging the content.
- Users can navigate from a higher level view to a lower level and vice versa

Example

Will run the DEMO program!

Conclusion and Future Work

- We have defined the different levels of information that can be created to better understanding a trace
- We have discussed the data structure we have used for prototyping the demo.
- We will continue to implement and optimize the aforementioned data structure.