From Fault identification and Abstraction to updates on the System State and Health Monitoring

Naser Ezzati Jivan
Michel Dagenais
Department of Computer and Software Engineering

May 11, 2010
École Polytechnique, Montreal

Table of Content

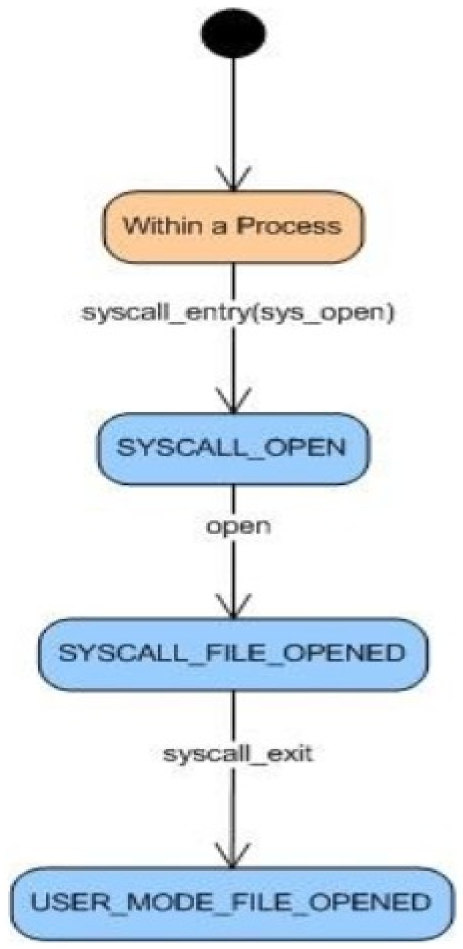
Trace Abstraction

- Generating high-level compound events from low-level raw events that:
  - Are more readable than the raw events
  - Still depict the system behavior
  - Can be used to detect faults and anomalies (By comparing them to learned normal or faulty behaviors)
    - Behavior abstraction in malware analysis (Beaurcamps, *et al*, 2010)
    - A layered architecture for detecting malicious behaviors (Martignoni *et al 2008)*

Abstraction of System Call Traces

- Techniques for the Abstraction of System Call Traces (Waseem Fadel, Dr. Abdelwahab Hamou-Lhadj)

Raw Events		Abstract Event
syscall_entry		
fs.open	⇒	“file open”
syscall_exit		



Fault Identification

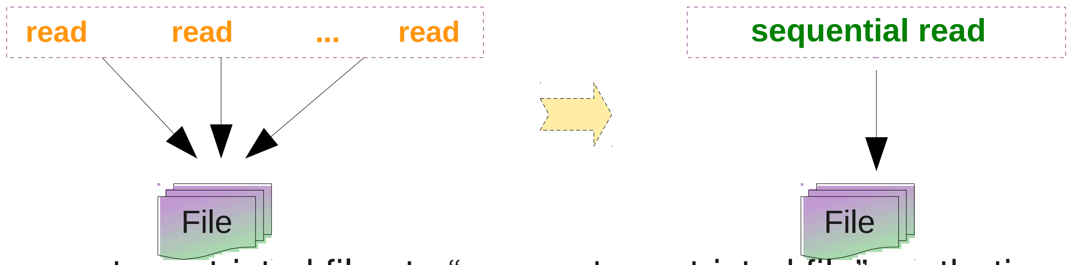
- Automated Fault Identification framework (Hashem Waly, Dr. B´chir Ktari)

```
scenario chroot(){
  event e1:chroot;
  event e2:open where (pid == e1.pid);
}
```

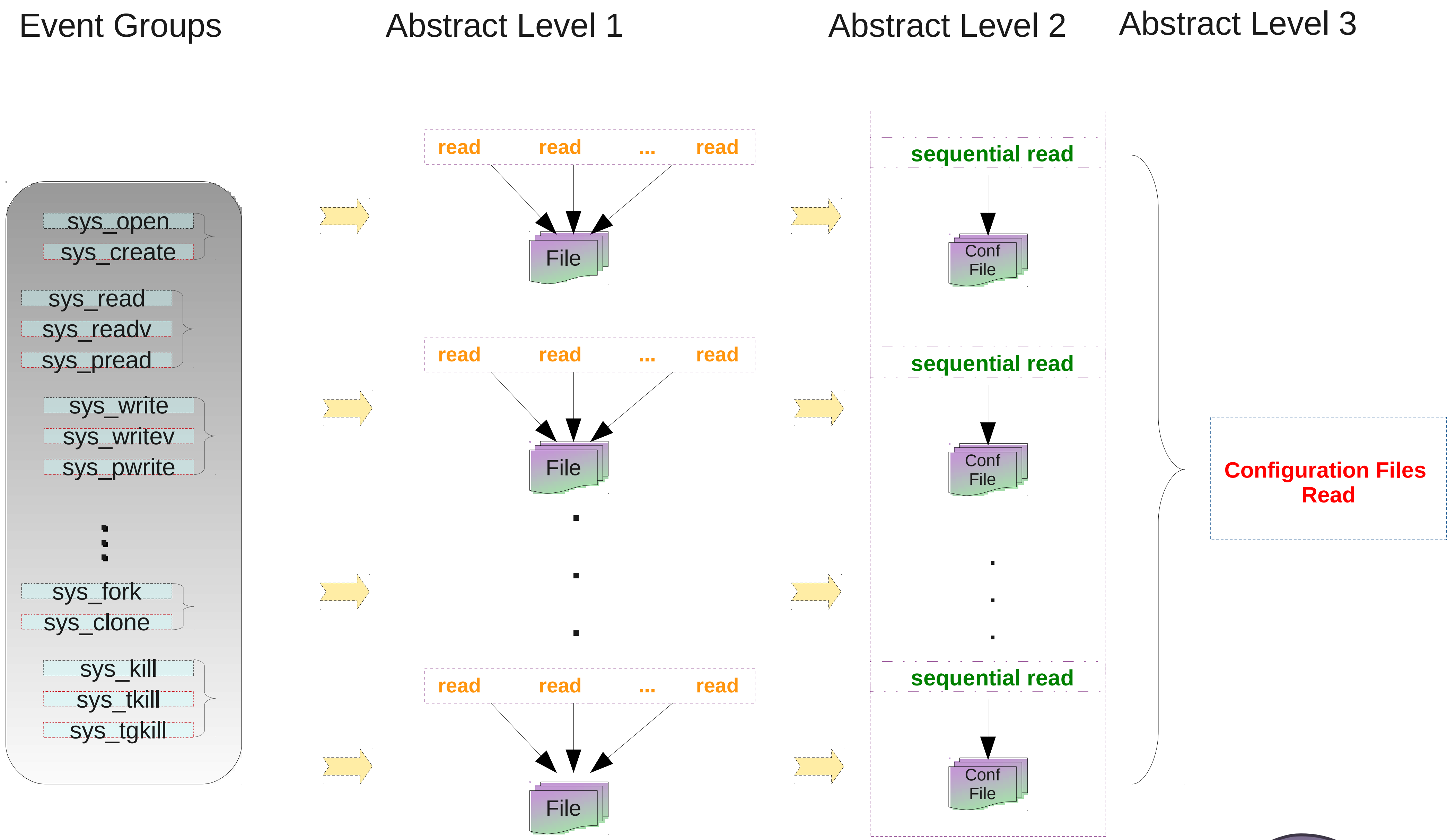
More Abstraction (1)

- Trace Abstraction vs Event Abstraction
  - Define patterns over group of similar events.
    - Both “create” and “open” system call can be used to open files.

- Using arguments of the system-calls and events besides considering the hierarchy:
  - Abstract all sequential “READ” operations of a specific file to a “SEQUENTIONAL READ” abstract event.



  - Abstract all write access to restricted files to “access to restricted file” synthetic event.
    - "/etc/passwd"
    - "/etc/utmp"
  - Abstract a set of “SEQUENTIONAL READ” event of files with extension name “**.conf”** to a “CONFIGURATION FILES Read” synthetic event.

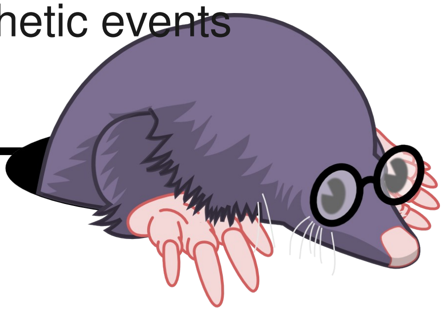

Event Groups
Abstract Level 1
Abstract Level 2
Abstract Level 3
sys_open
sys_create
sys_read
sys_readv
sys_pread
sys_write
sys_writev
sys_pwrite
sys_fork
sys_clone
sys_kill
sys_tkill
sys_tgkill
read
read
...
read
File
sequential read
Conf File
Configuration Files Read

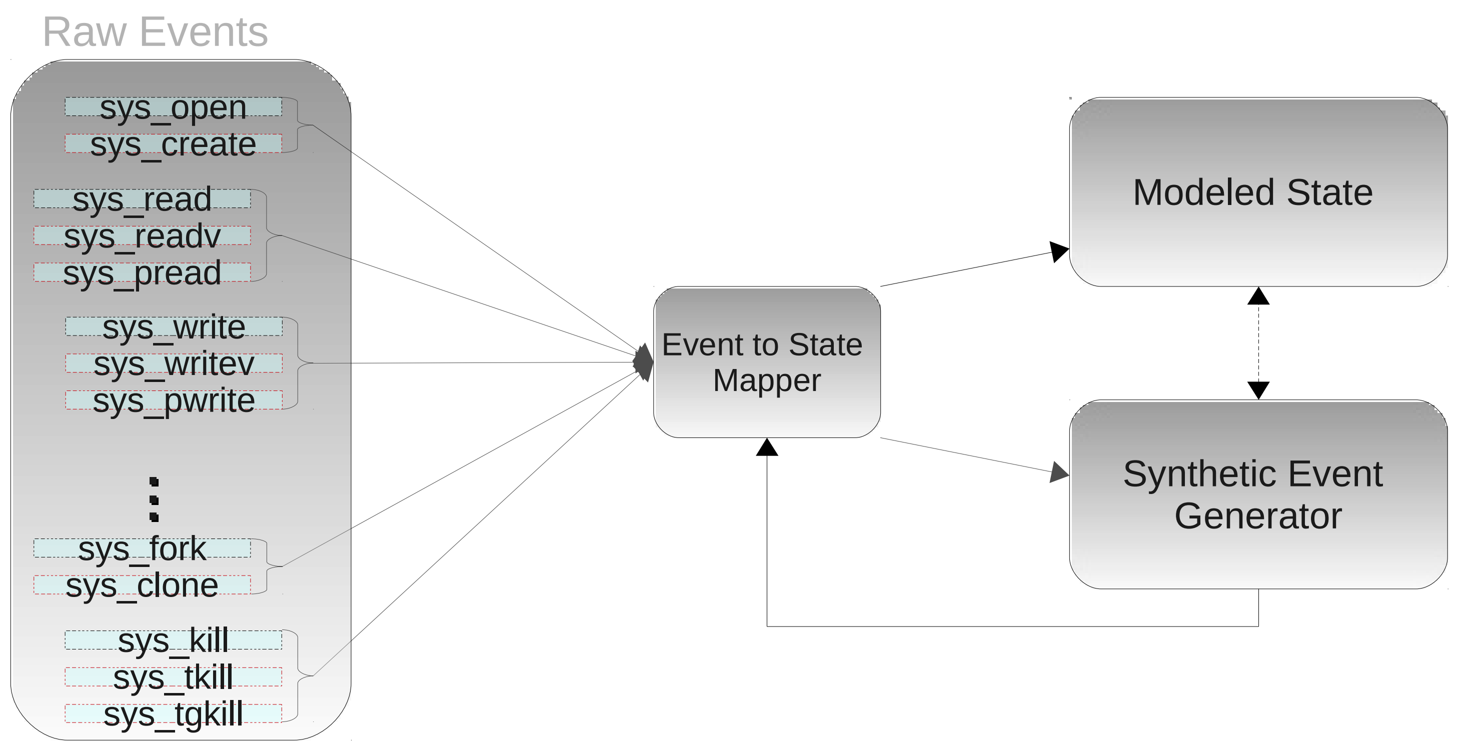
More Abstraction (2)

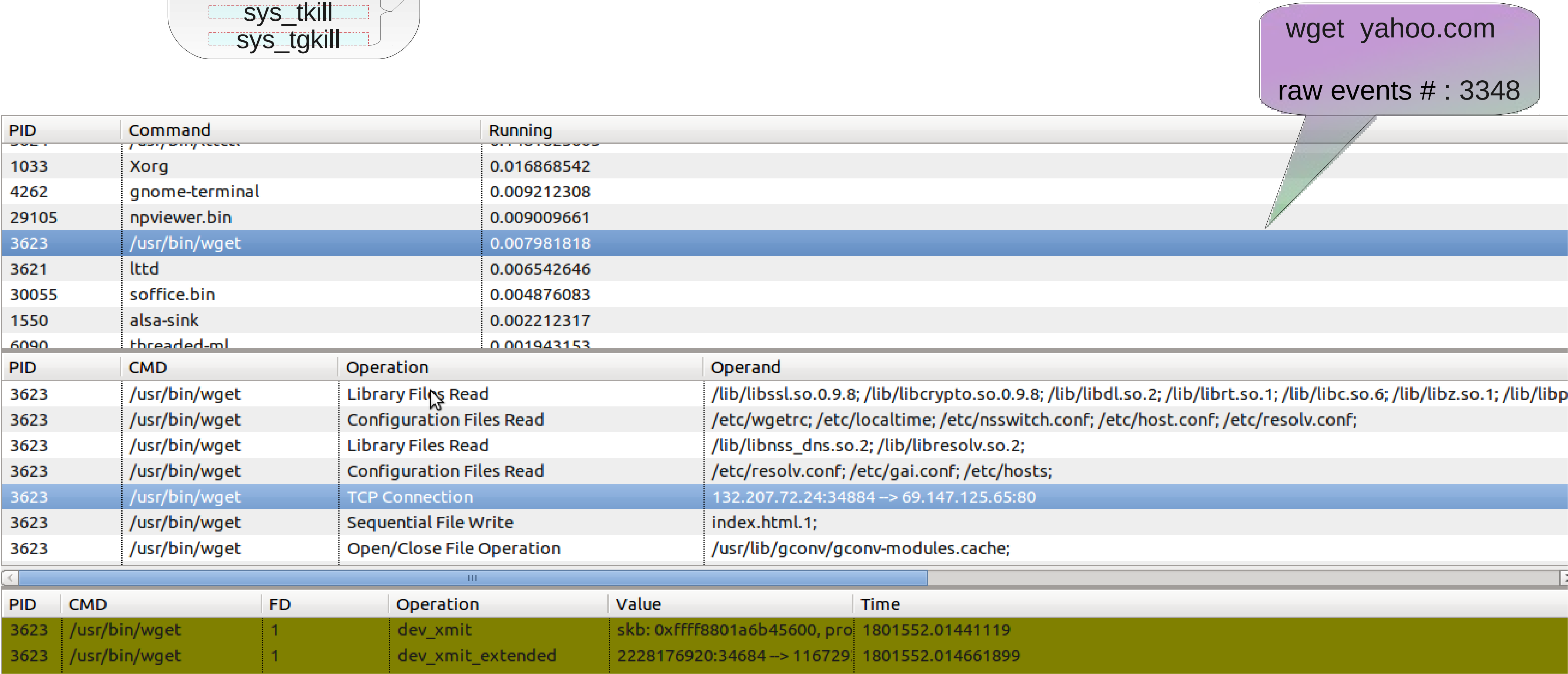
- Using states and state changes:
  - Modeled state keeps the state of the traced system

```
$machine/processes/$pid/state
                        exec_name
                        files
                        fds
```

  - For each raw event, a set of state changes can be defined
    - Fs.read, fs.readv and fs.pread change the state of a file to READ state.
    - Kill, tkill, tgkill system calls change the state of a process to “KILLED”
    - Schedule event changes the state of the current and the scheduled process.
  - State changes can also be associated for synthetic events. For example, “TCP connect” synthetic events can change the state of sockets to CONNECTION REQUESTED, ESTABLISHED, DATA TRANSFER, CLOSED , …
  - The Idea is to define patterns of states and states changes to create synthetic events







PID	Command	Running
[illegible]	[illegible]	[illegible]
1033	Xorg	0.016868542
4262	gnome-terminal	0.009212308
29105	npviewer.bin	0.009009661
3623	/usr/bin/wget	0.007981818
3621	lttd	0.006542646
30055	soffice.bin	0.004876083
1550	alsa-sink	0.002212317
6090	threaded-ml	0.001943153

PID	CMD	Operation	Operand
3623	/usr/bin/wget	Library Files Read	/lib/libssl.so.0.9.8; /lib/libcrypto.so.0.9.8; /lib/libdl.so.2; /lib/librt.so.1; /lib/libc.so.6; /lib/libz.so.1; /lib/libp
3623	/usr/bin/wget	Configuration Files Read	/etc/wgetrc; /etc/localtime; /etc/nsswitch.conf; /etc/host.conf; /etc/resolv.conf;
3623	/usr/bin/wget	Library Files Read	/lib/libnss_dns.so.2; /lib/libresolv.so.2;
3623	/usr/bin/wget	Configuration Files Read	/etc/resolv.conf; /etc/gai.conf; /etc/hosts;
3623	/usr/bin/wget	TCP Connection	132.207.72.24:34884 --> 69.147.125.65:80
3623	/usr/bin/wget	Sequential File Write	index.html.1;
3623	/usr/bin/wget	Open/Close File Operation	/usr/lib/gconv/gconv-modules.cache;

PID	CMD	FD	Operation	Value	Time
3623	/usr/bin/wget	1	dev_xmit	skb: 0xffff8801a6b45600, pro	1801552.01441119
3623	/usr/bin/wget	1	dev_xmit_extended	2228176920:34684 --> 116729	1801552.014661899

More Abstraction (3)

Using statistics

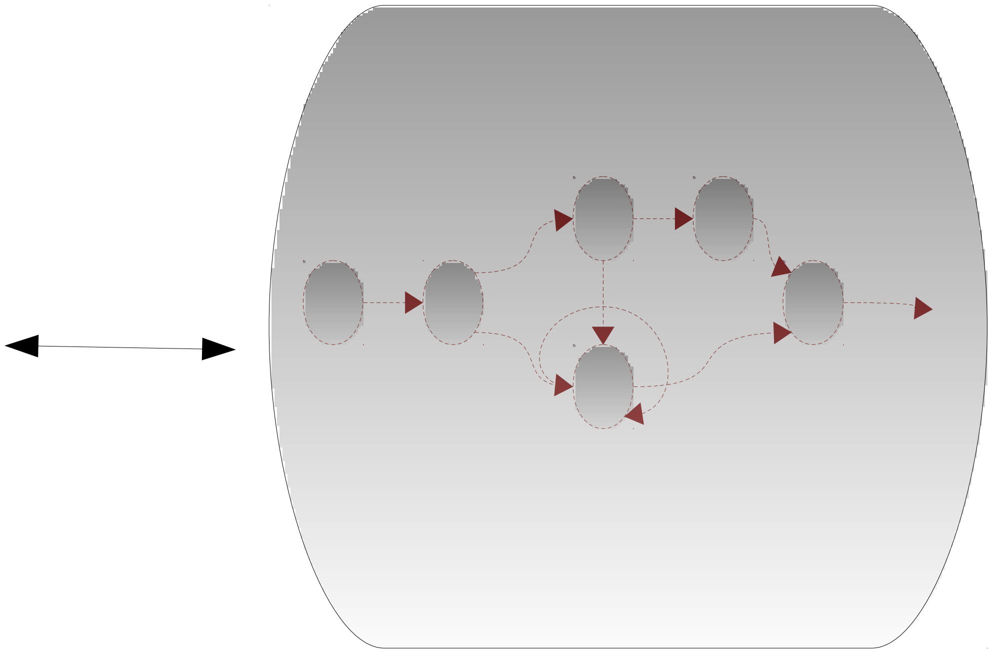
- For system resources, we may store statistics for whole trace or particular time intervals
  - CPU usage per process
  - I/O throughput per process
    - Number of bytes read and written
- We can define patterns based on these statistics to create synthetic events
  - Detecting Denial of service attacks
    - “Fork Bomb” attack
    - “SYN Flood” attack
  - Detecting “Port Scanning”
  - ...

Attribute Tree

State and State Values

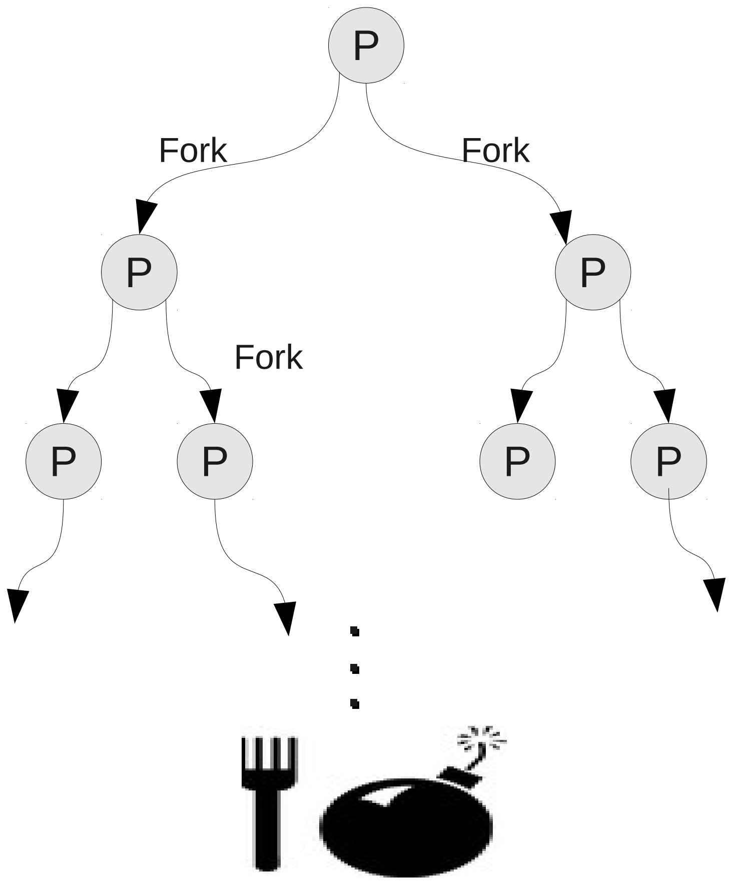
```
$machine/processes/$pid/state
                        execmode_stack
                        exec_name
                        fds
                            $fd1
                                filename
                                type
                            $fd2
                            ...
        /cpus/$cpuid/current_process
                        pid
        /disks
        /memory
```

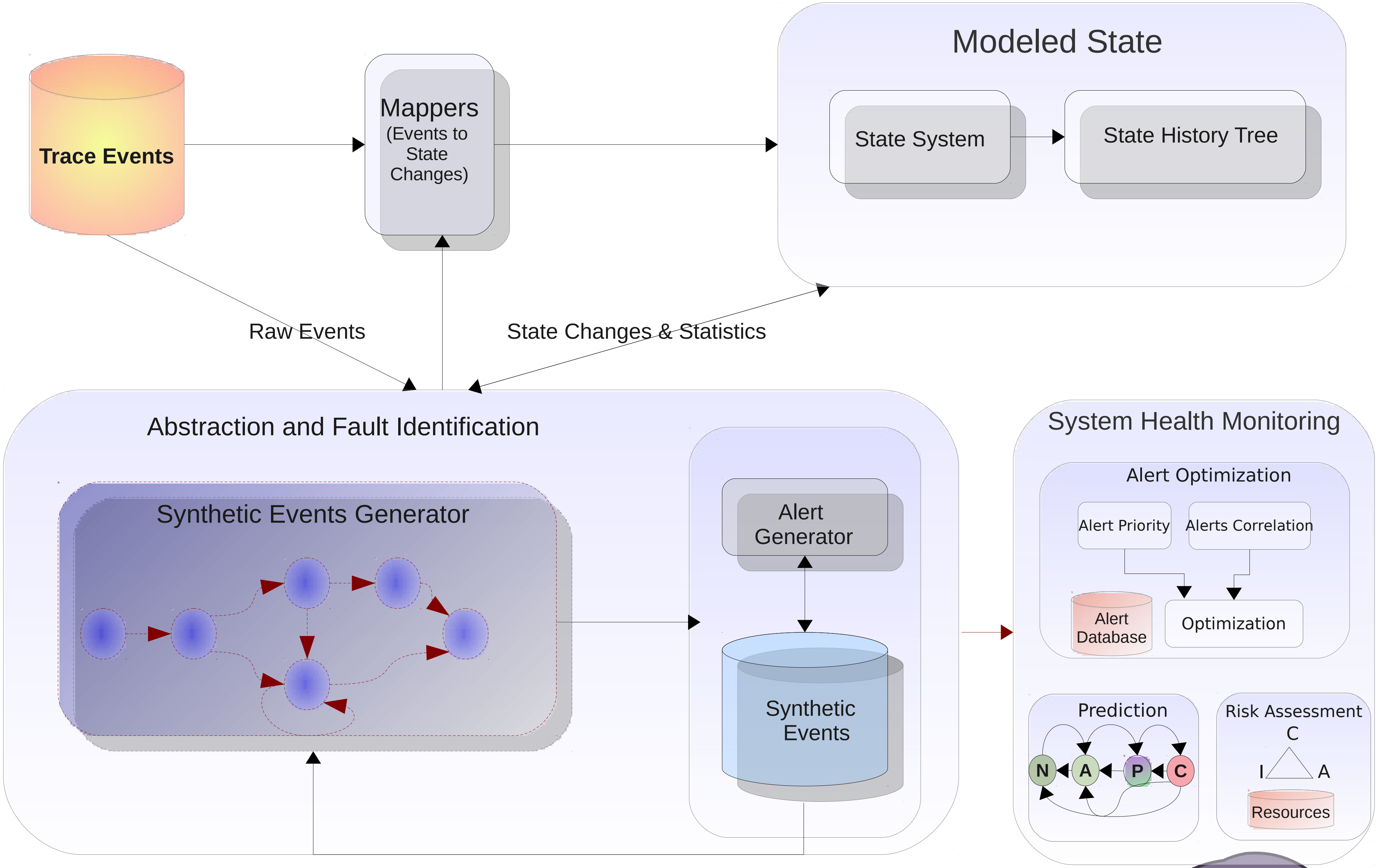
State Machines

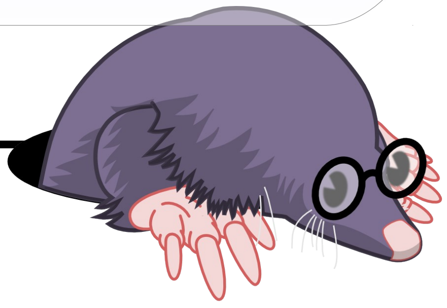


Examples

- Fork bomb
  - This attack performs recursive forks and creates a large number of processes.
  - By keeping track of the number of running processes and the number of fork operations per process, we can detect this attack.
- Syn flood attack
- ...



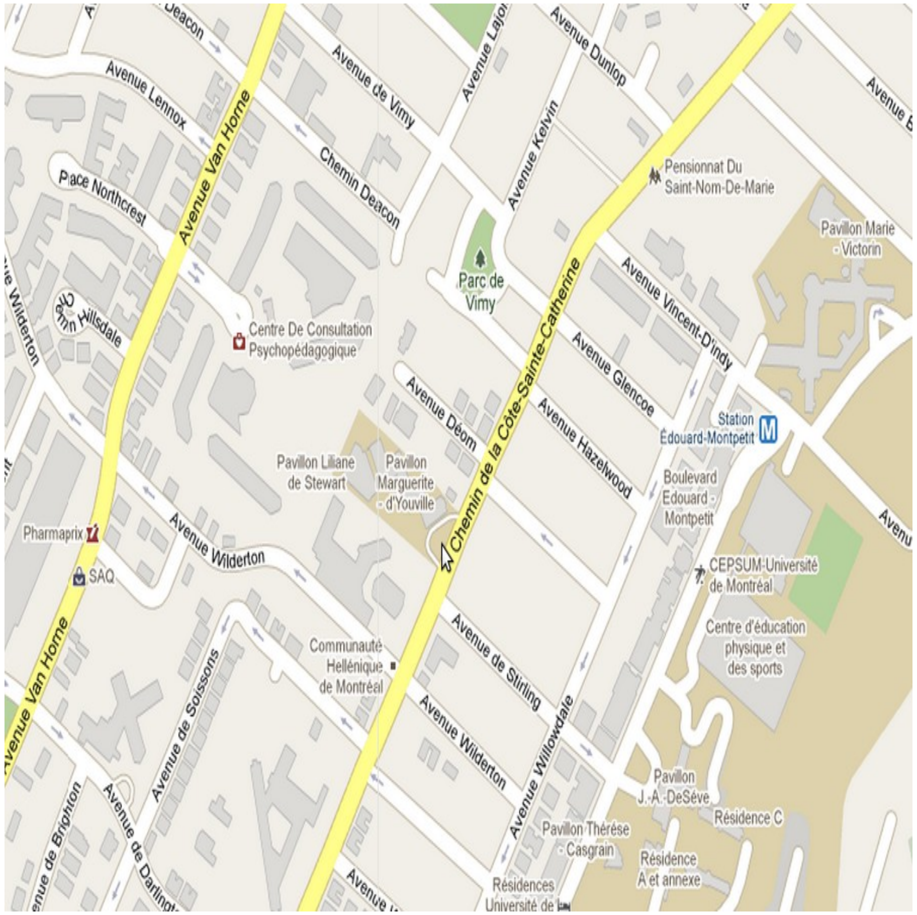

Trace Events
Mappers
(Events to State Changes)
Modeled State
State System
State History Tree
Raw Events
State Changes & Statistics
Abstraction and Fault Identification
Synthetic Events Generator
Alert Generator
Synthetic Events
System Health Monitoring
Alert Optimization
Alert Priority
Alerts Correlation
Alert Database
Optimization
Prediction
N
A
P
C
Risk Assessment
C
I
A
Resources



Which level of details?

Canada
Hudson Bay
YT
NT
NU
BC
AB
SK
MB
ON
QC
WA
MT
ND
MN
OR
ID
WY
SD
WI
MI
NE
IA
NV
UT
CO
IL
IN
OH
PA
NY
VT
ME
NB
NS
NH
MA
CT
RI

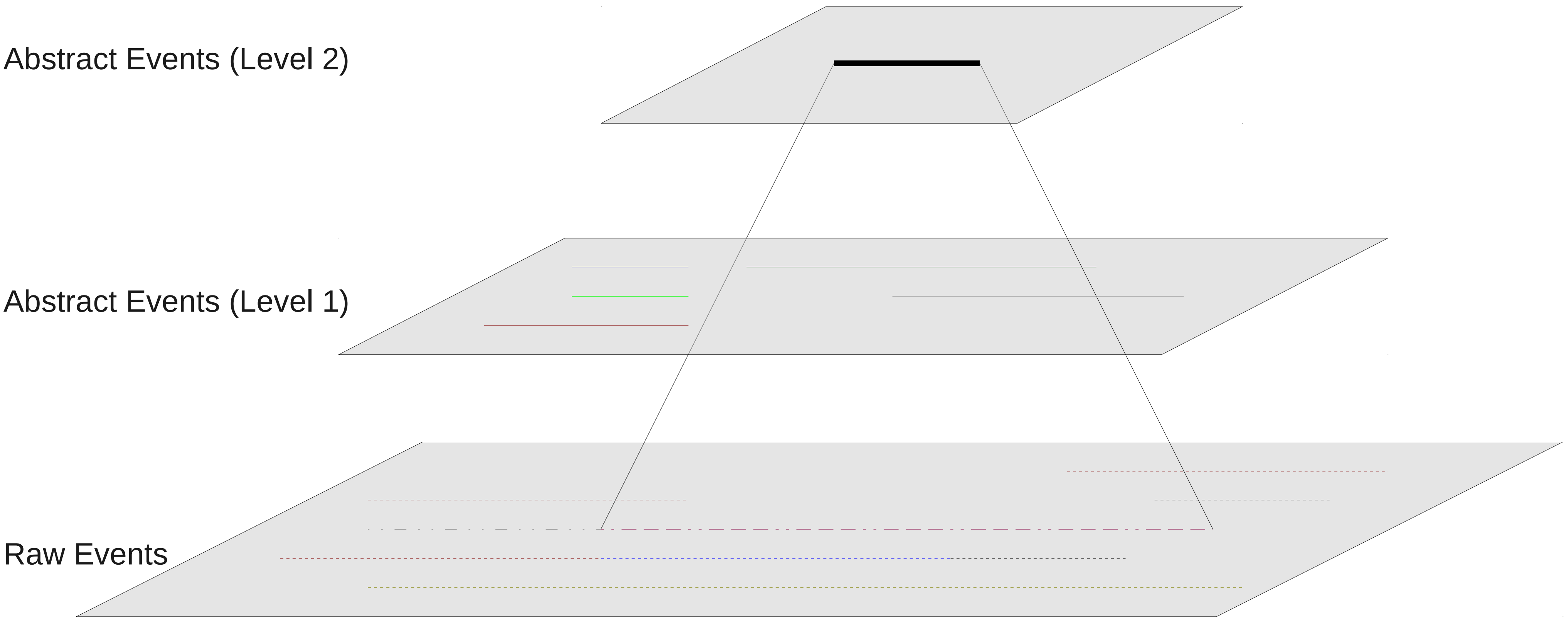
OR

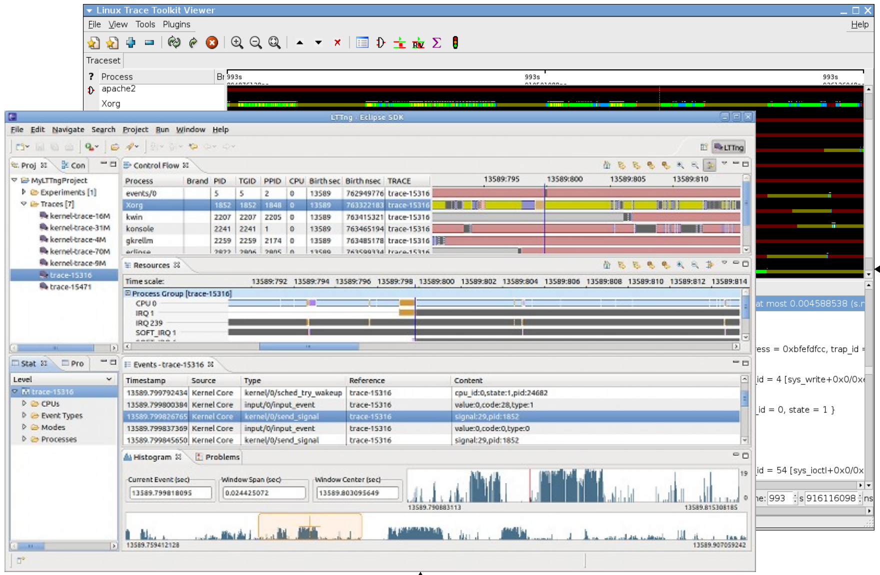


Linking events at different levels

- For better understanding of the system:
  - Users need to access easily and efficiently to different levels of events
    - Synthetic or raw events
  - Users need to navigate from the high level events to low level events and from low level views to high level views
    - Which high level event does this event belong to?
    - What are the related raw events to a given synthetic event?
- Solution: creating a “Link Index”
  - R-Tree

Linking events at different levels

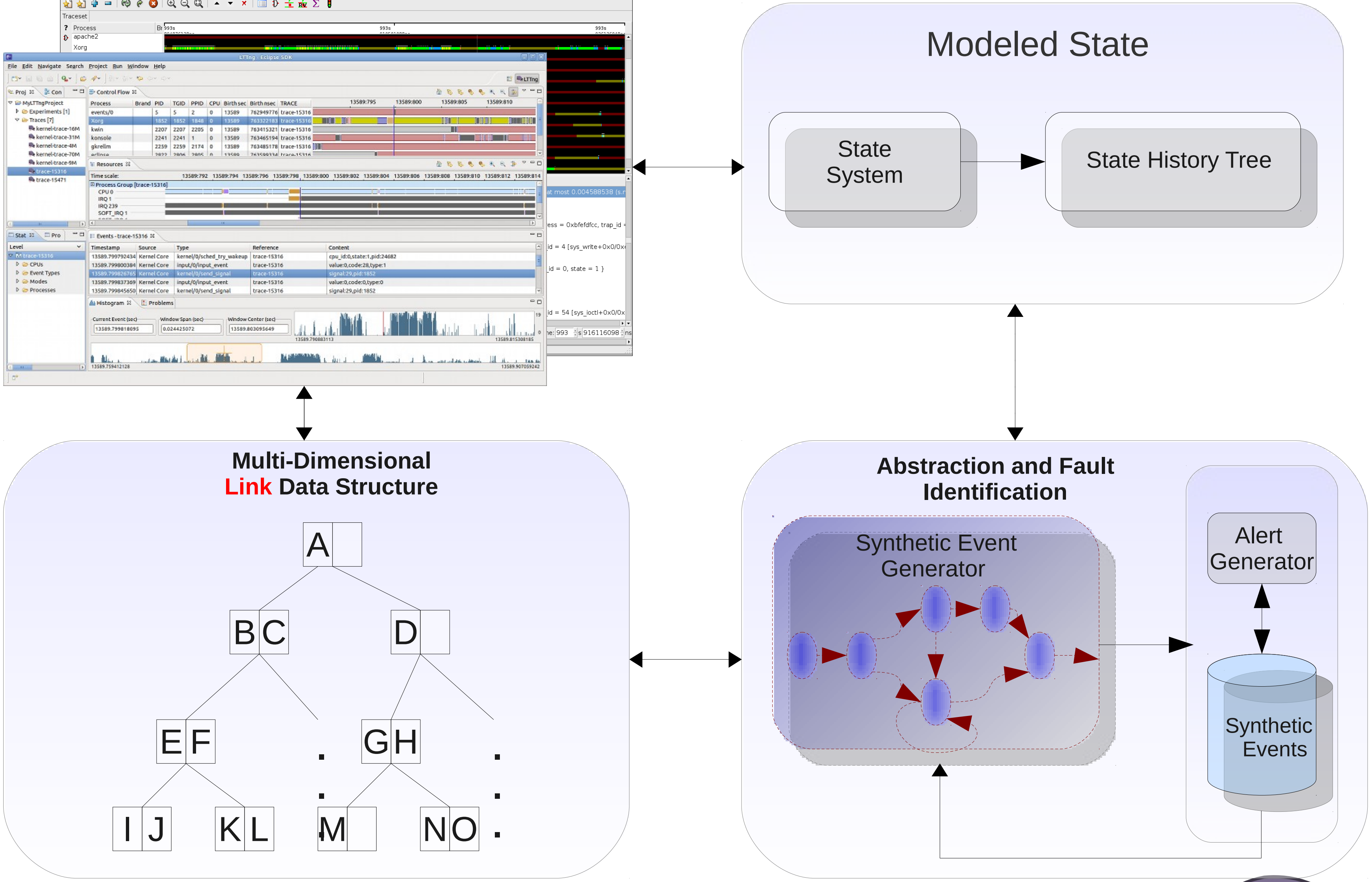




Process	Brand	PID	TGID	PPID	CPU	Birth sec	Birth nsec	TRACE
events/0		5	5	2	0	13589	762949776	trace-15316
Xorg		1852	1852	1848	0	13589	763322183	trace-15316
kwin		2207	2207	2205	0	13589	763415321	trace-15316
konsole		2241	2241	1	0	13589	763465194	trace-15316
gkrellm		2259	2259	2174	0	13589	763485178	trace-15316

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

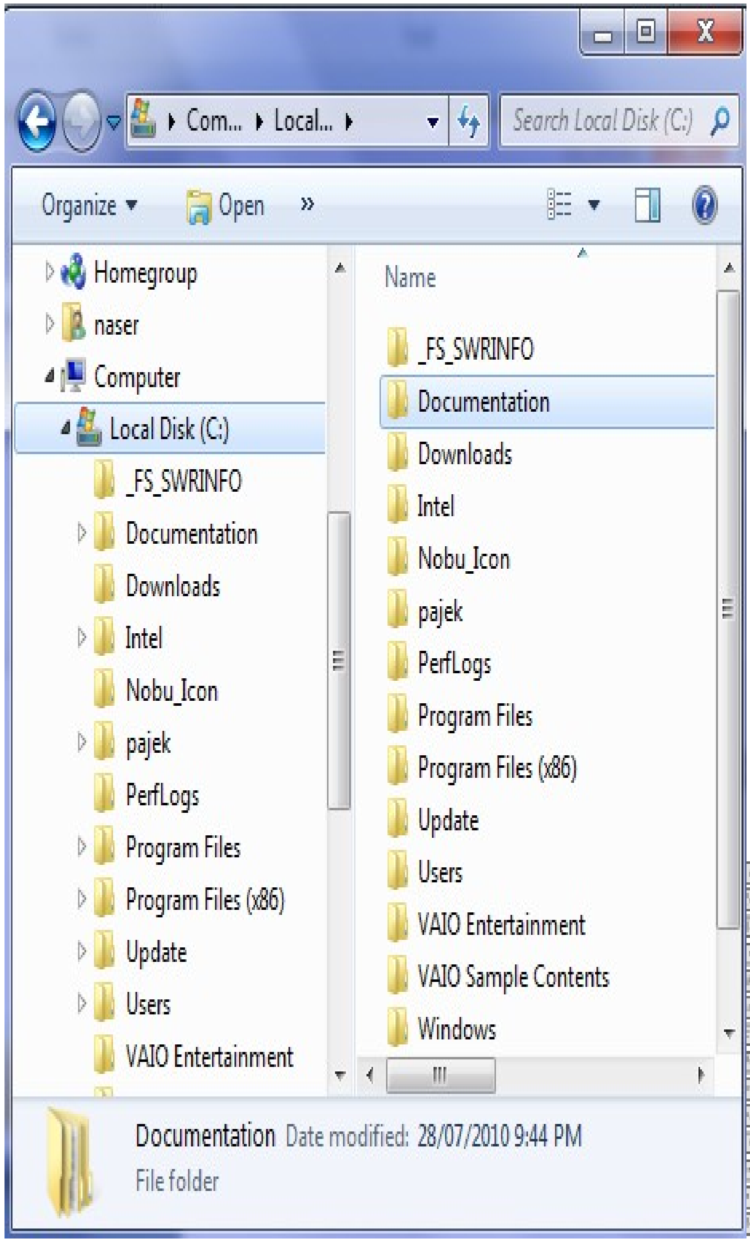
Current Event (sec): 13589.799818095
Window Span (sec): 0.024425072
Window Center (sec): 13589.803095649

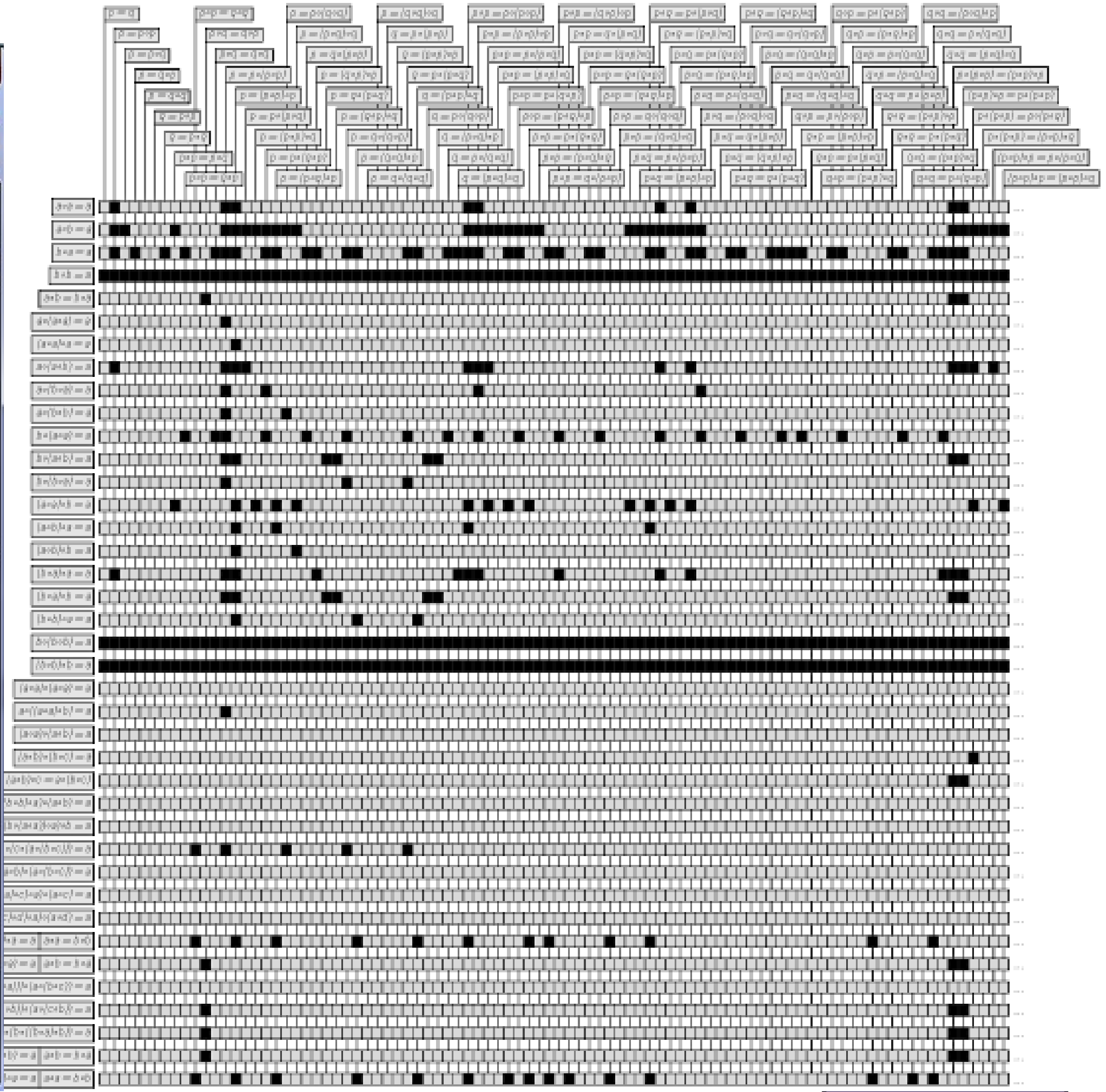


Modeled State
State System
State History Tree
Multi-Dimensional Link Data Structure
A
B C
D
E F
G H
I J
K L
M
N O
Abstraction and Fault Identification
Synthetic Event Generator
Alert Generator
Synthetic Events

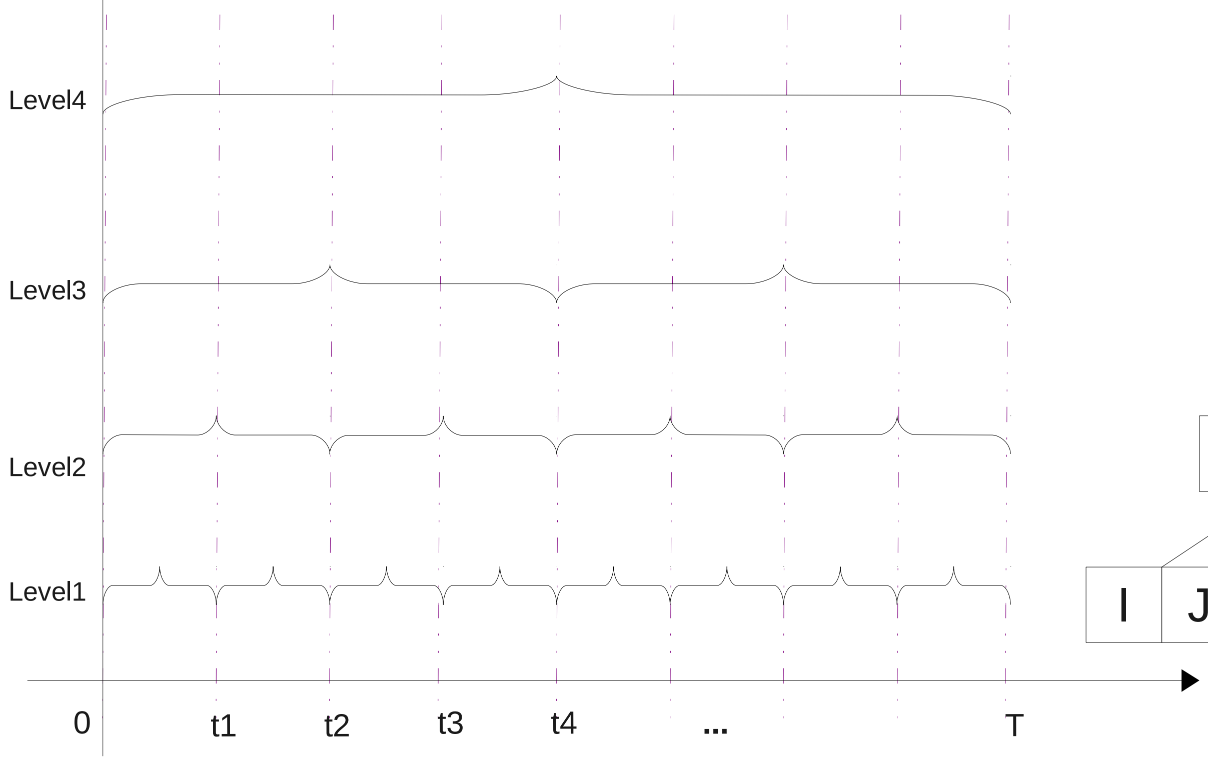
Features

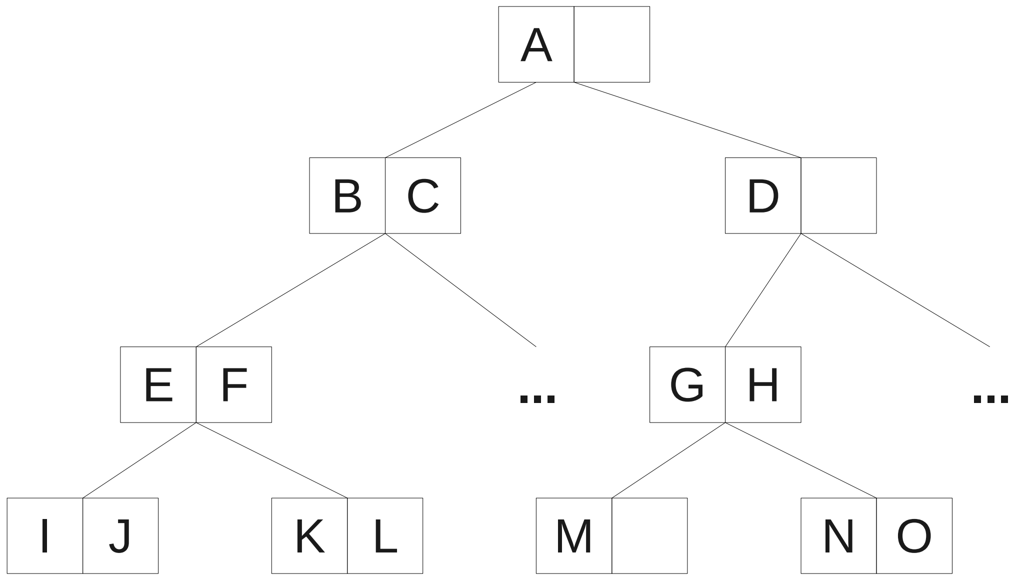
- Focusing
  - Showing relevant information of a specific synthetic event or a behavior
- Zooming
  - Is a technique to cope with large amount of information.
  - Showing more details(Google Map) and enlarging the content.

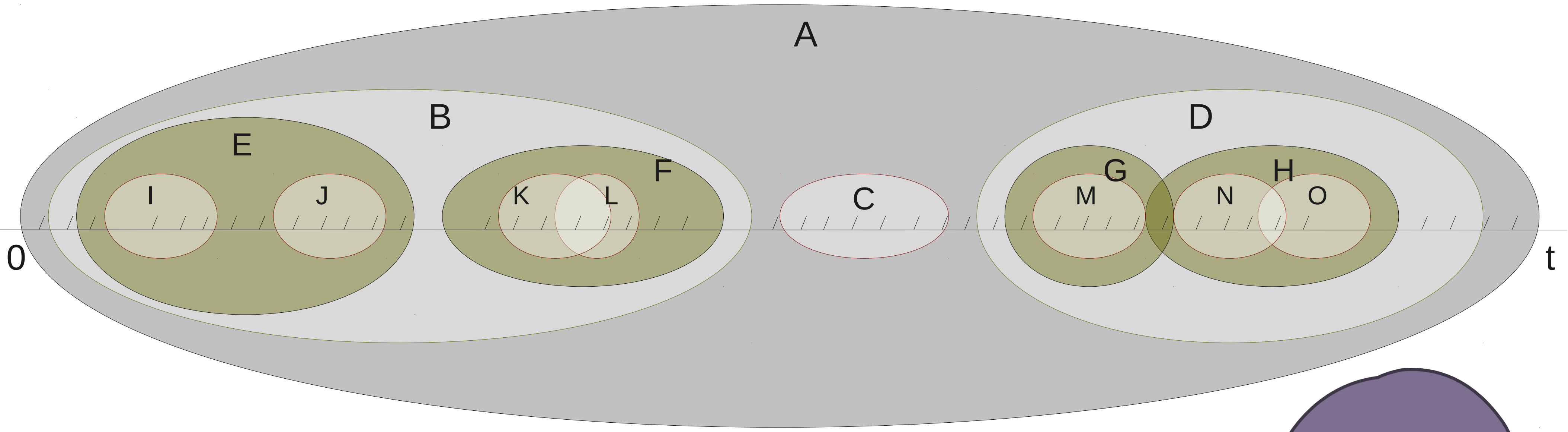




http://www.wolframscience.com/nksonline/page-812


Level4
Level3
Level2
Level1
0
t1
t2
t3
t4
...
T


A
B
C
D
E
F
...
G
H
...
I
J
K
L
M
N
O


A
B
D
E
F
G
H
I
J
K
L
C
M
N
O
0
t

Conclusion and Future Work

- We used arguments of events, states and state changes and also statistics to create synthetic events.
- Using the "modeled state" information for abstraction can help to create more useful synthetic events and also to detect the wide range of attacks and faults.
- During the high level events creation, we need to keep some information for linking the high level and low level events.
- Developing the proposed link data structure will be the next step of the project.

References (1)

[1] J. P. Black, M. H. Coffin, D. J. Taylor, T. Kunz, and T. Basten, “Linking Specification, Abstraction, and Debugging,” CCNG Technical Report E-232, Computer Communications and Networks Group, University of Waterloo, November 1993.

[2] M. Auguston, A. Gates, M. Lujan, “Defining a program Behavior Model for Dynamic Analyzers,” Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering, SEKE’97, Madrid, Spain, pages 257-262, June 1997.

[3] W. Fadel, “Techniques for the Abstraction of System Call Traces,” M.Sc.A. dissertation., Concordia University, 2010.

[4] A. Hamou-Lhadj and T. Lethbridge, “A Survey of Trace Exploration Tools and Techniques,” In Proceedings of the 14th IBM Conference of the Centre for Advanced Studies on Collaborative Research, IBM Press, pages 42-55, 2004.

[5] W. D. Pauw, R. Helm, D. Kimelman, and J. M. Vlissides, “Visualizing the behavior of object-oriented systems,” In Proc. 8th Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), pages 326– 337. ACM, 1993.

[6] D. B. Lange and Y. Nakamura, “Object-oriented program tracing and visualization,” IEEE Computer, 30(5), pages 63–70, 1997.

[7] D. F. Jerding, J. T. Stasko, and T. Ball, “Visualizing interactions in program executions,” In Proc. 19th Int. Conf. on Software Engineering (ICSE), pages 360–370, ACM, 1997.

[8] A. Chan, R. Holmes, G. C. Murphy and A. T. T. Ying, “Scaling an Object-oriented System Execution Visualizer through Sampling,” Proc. of the IEEE Int. Workshop on Program Comprehension (ICPC'03), 2003.

[9] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera, “Scalable Performance Analysis: The Pablo Performance Analy sis Environment,” in Proc. Scalable Parallel Libraries Conf. IEEE Computer Society, pages 104–113, 1993.

[10] T. Syst, K. Koskimies, and H. M¨ller. Shimba, “An environment for reverse engineering Java software systems,” Software - Practice and Experience, 31(4): pages 371–394, 2001.

[11] W. D. Pauw and S. Heisig, “Zinsight: a visual and analytic environment for exploring large event traces.” In Proceedings of the 5th international symposium on Software visualization (SOFTVIS '10). ACM, New York, NY, USA, pages 143-152, 2010.

References (2)

[12] J. L. Lin, X . S. Wang, and S. Jajodia, “Abstraction-based misuse detection: High-level specification and adaptable strategies,” In 11th IEEE Computer Security Foundations Workshop, Rockport, MA, 1998.

[13] S. Eick and A. Karr, “Visual Scalability,” Journal of Computational & Graphical Statistics, vol. 11, no. 1, pages 22–43, 2002.

[14] W. Nagel, A. Arnold, M. Weber, H. Hoppe, and K. Solchenbach, “VAMPIR: Visualization and Analysis of MPI Resources,” Supercomputer, vol. 12, no. 1, pages 69–80, 1996.

[15] V. Pillet, J. Labarta, T. Cortes, and S. Girona, “PAR- AVER: A tool to visualise and analyze parallel code,” in Proceedings of Transputer and occam Developments, WOTUG-18., ser. Transputer and Occam Engineering, vol. 44. Amsterdam: [S.l.]: IOS Press, pages 17– 31, 1995.

[16] C. W. Lee, C. Mendes, and L. Kale, “Towards scalable performance analysis and visualization through data reduction,” Parallel and Distributed Processing, IEEE International Symposium on, pages 1–8, April 2008.

[17] S. P. Reiss, “Visualizing Java in action,” In Proc. Symp. on Software Visualization (SOFTVIS), pages 57–65, ACM, 2003.

[18] O. Zaki, E. Lusk, W. Gropp, and D. Swider, “Toward scalable performance visualization with jumpshot,” Int. J. High Perform. Comput. Appl., vol. 13, no. 3, pages 277–288, 1999.

[19] W. D. Pauw and S. Heisig, “Zinsight: a visual and analytic environment for exploring large event traces.” In Proceedings of the 5th international symposium on Software visualization (SOFTVIS '10). ACM, New York, NY, USA, pages 143-152, 2010.

[20] C. Bennett, D. Myers, M. A. Storey, D.M. German, D. Ouellet, M. Salois, and P. Charland, “A Survey and Evaluation of Tool Features for Understanding Reverse Engineered Sequence Diagrams,” Journal of Software Maintenance and Evolution: Research and Practice, March 2008.

[21] P. Uppuluri, “Intrusion detection/prevention using behavior specifications,” Ph.D. dissertation, State University of New York at Stony Brook, United States - New York, 2003.

[22] S. Eckmann, G. Vigna and R. Kemmerer, “Statil: An attack language for state based intrusion detection system,” 2000.