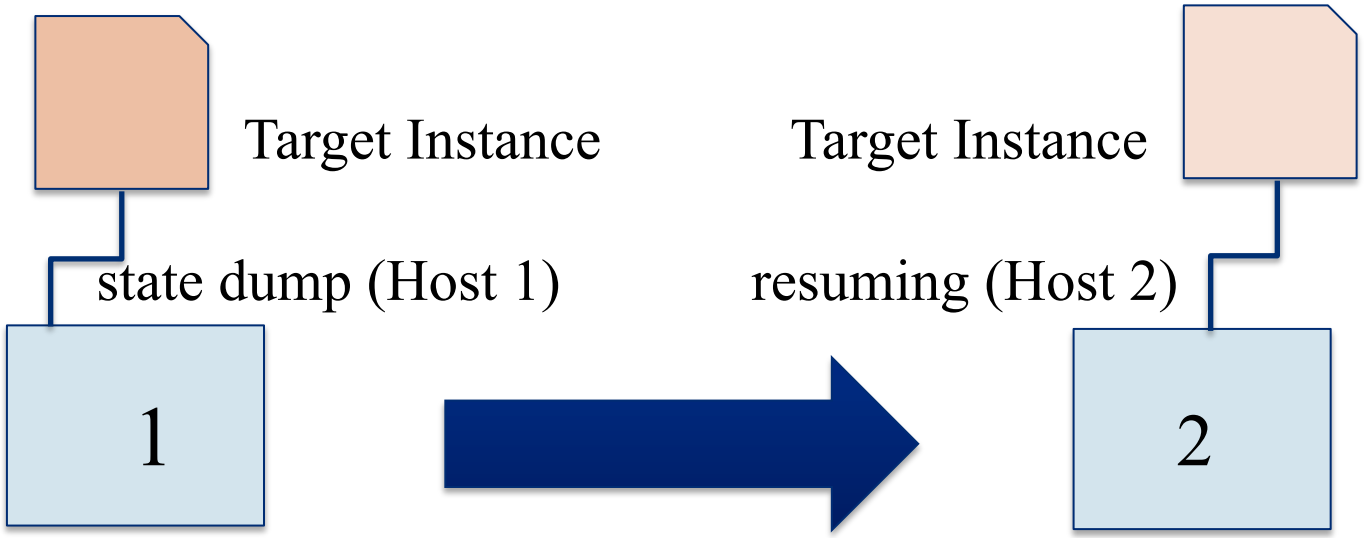
Software Engineering Conference Russia
October 2017, St. Petersburg

Constructing the formal grammar of system calls

Nikolay Efanov, МФТИ

Motivation

1. Immediate process tree analysis in Checkpoint-Restore.
2. Overhead/extra metadata minimization during Live-Migration.
3. Ideally: process tree + useful context (memory, files etc) only.



The Statement and Restrictions:

1. Reconstruct the syscall sequences which lead to some snapshotted process tree.
2. Restrictions:

- Linux syscalls only
- Input is correct. Otherwise: ”Parsing error”.
- Syscalls:
  - ✓ **Fork**: creates a child process
  - ✓ **Setsid**: creates a session with current process as leader
  - ✓ **Setpgid**: sets a group as setpgid(0,pgid)
  - ✓ **Exit**: terminates a process -> reparent is initiated
- ✓ → Basis of process tree transformations

Combinatorial Estimations

- Different process trees using **fork**:

$$F(n) = \sum_{i=2}^{n-1} i^{i-2}$$

The summation is taken from 2 in view of the exclusive SCHED and INIT processes

A number of processes can potentially get 2^{16}

Direct generation/search is not the best idea!

- Moreover …

Combinatorial Estimations

- Taking **setsid** into account:

$$F(n) = \sum_{i=2}^{n-1} i^{i-2} \cdot 2^{i-1} \cdot \sum_{j=0}^{2^l-1} 2^{(\mathbf{f}(j)\cdot\mathbf{k})}$$

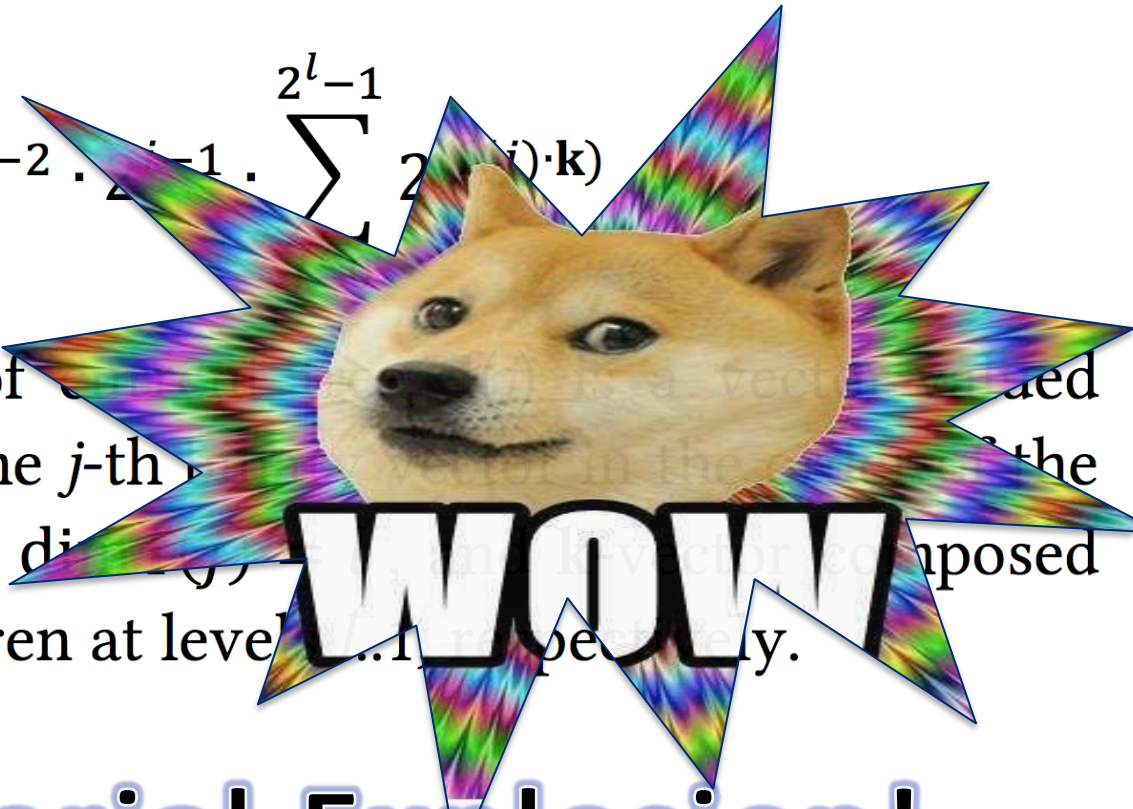
Where l is a depth of current tree, $\mathbf{f}(j)$ is a vector-valued function, which returns the j-th binary vector in the order of the incremental enumeration, $\dim \mathbf{f}(j) = l$, and $\mathbf{k}$-vector composed of a total number of children at levels $l..1$, respectively.

Combinatorial Estimations

- Taking **setsid** into account:

$$F(n) = \sum_{i=2}^{n-1} i^{i-2} \cdot 2^{i-1} \cdot \sum_{\mathbf{k}=0}^{2^{l}-1} 2^{(\mathbf{j}(i)\cdot\mathbf{k})}$$

Where l is a depth of [illegible] function, which returns the j-th [illegible] incremental enumeration, [illegible] of a total number of children at level [illegible].



- **Combinatorial Explosion!**

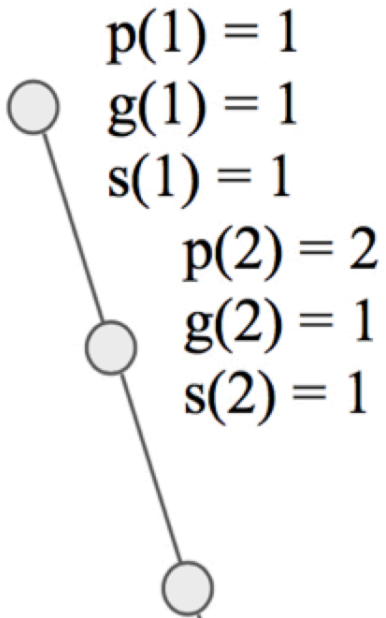
Suggestion #1: new tree notation

The new process tree single-string notation:

- Single process: “p g s [...]”, where:
  - Numbers: p – PID, s – SID, g – PGID
  - ‘[‘,’]’ – terminal limiters of children-containing list .
  - list for a root of any subtree contains all of descendants (thus, **fork** is notation-based)
- Example:

  String: “1 1 1 [2 1 1 […]]”

  is equivalent to such tree →



Suggestion #2: syscall grammar

The grammar rules for **fork, setsid, setpgid**,~~exit*~~:

- Notation-based form:
  - **fork**(* * * [*]) --> * * *[* \2 \3 [] \4].
  - **setsid**(* * * [*]) --> \1 \1 \1 [\4].
  - {p p * [*], **setpgid**(p, * * \1 [*]) | **setpgid**(p, p * * [*])} --> {p p \1 [\2], \3 p \1 [\5] | p p \2 [\3]}, where '{','}' : configuration exists or existed there.

- The grammar is context-sensitive (**setpgid** rule)
- **Setpgid** rule is separable into independent context-sensitive and context-free cases.

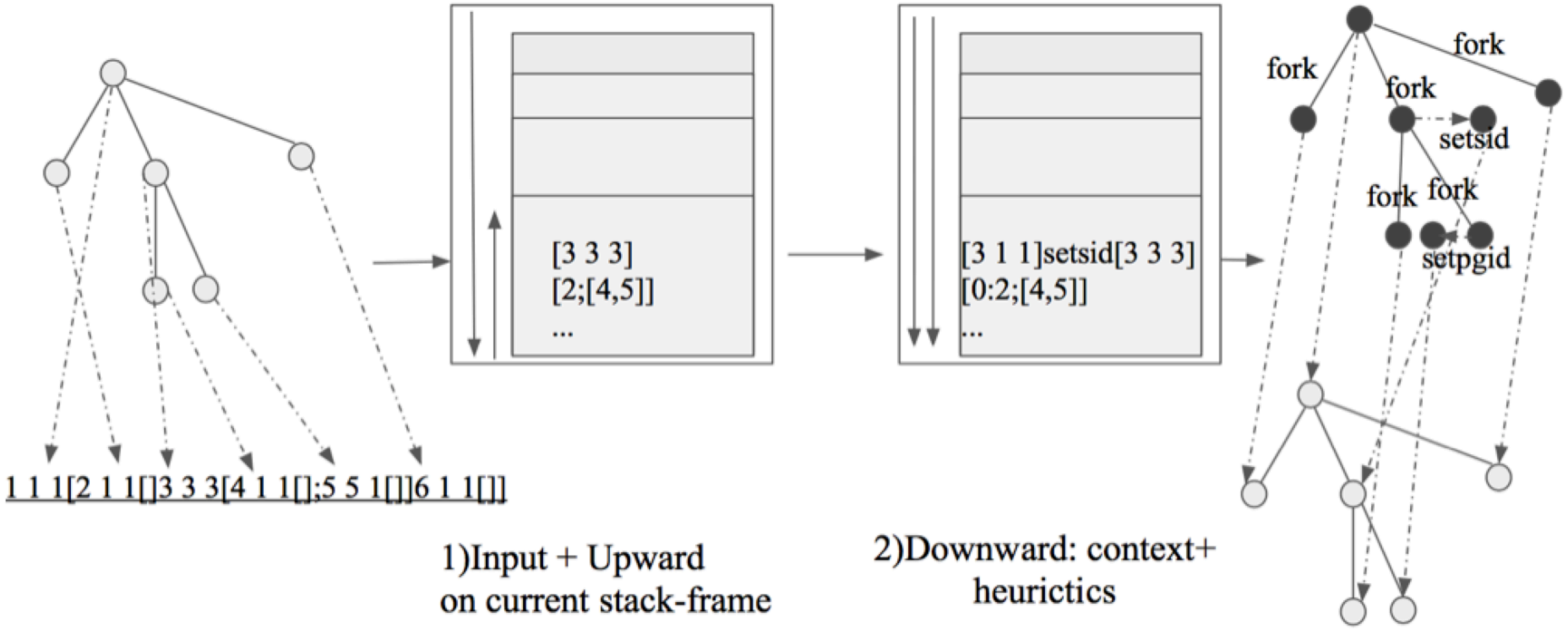
Grammar Shortening: exit

The grammar rule for **exit**:

- Notation-based form:
  - {1 * * [*], **exit**(* * * [*])} --> 1 \1 \2 [\3 \7].
- The grammar is type-0 (**exit** is shortening rule)
- 3 heuristics (reverse-reparenting):
  1) Is session(child) unique? --> create ‘exited’ process with PID==session number, then --> reverse reperenting.
  2) Otherwise: no ‘exited’ processes in session(child)? --> attach the ‘exited’ process to session leader, then --> reparenting via **setpgid**, **setsid**.
  3) Else: look up for suitable ‘exited’ process session(child) --> attach. No suitable processes? --> use 2).

Grammar Analyzer

- Two-stage analyzer: O(Nlog(N)log(S)log(P))
  - Stage 1: context-free analysis --> intermediate pre-analysed tree
  - Stage 2: context checking --> final syscall sequences restoring
    - +AVL-Based structures: p,g,s logging



Example: Stack In More Detail

"1 1 1 [5 4 3 [ ] 3 3 3 [4 4 3 [ ] 9 3 3 [ ]]]"

0:[1, 1, 1]

…

4:[3, 1, 1], 'setsid', [3, 3, 3] # setsid(fork(INIT))

5:[1, [7, 9]] # links in a metadata line

6:[4, 3, 3], 'setpgid', [4, 4, 3] # context-free setpgid()

7:[5,[]]

…

2:[5, 4, 3]

3:[11,[]] # reverse-reparented

…

6:[4, 3, 3], 'setpgid', [4, 4, 3]

7:[5,[11]] # 'exited' process is added (PID==10)

…

10:[10, 4, 3],'exit()' # 'exited'

11:[11, [3]] #

[1,1,1]

|__ [3 1 1] →setsid()→[3 3 3]

| |__[9 3 3]

|__[4 3 3]→setpgid(4)→[4 4 3]

|__[10 4 3]→exit()

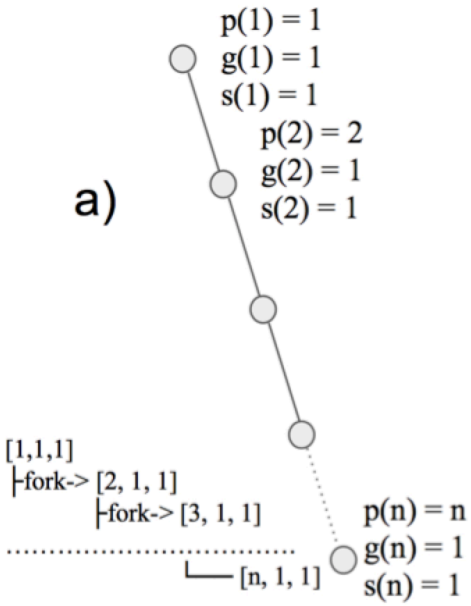
|__[5 4 3]

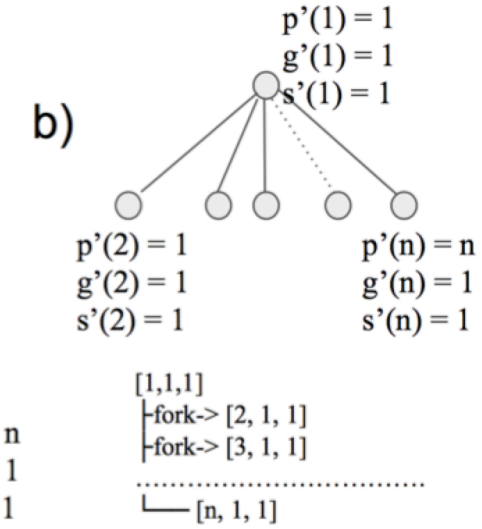
'|__' as a 'fork()'

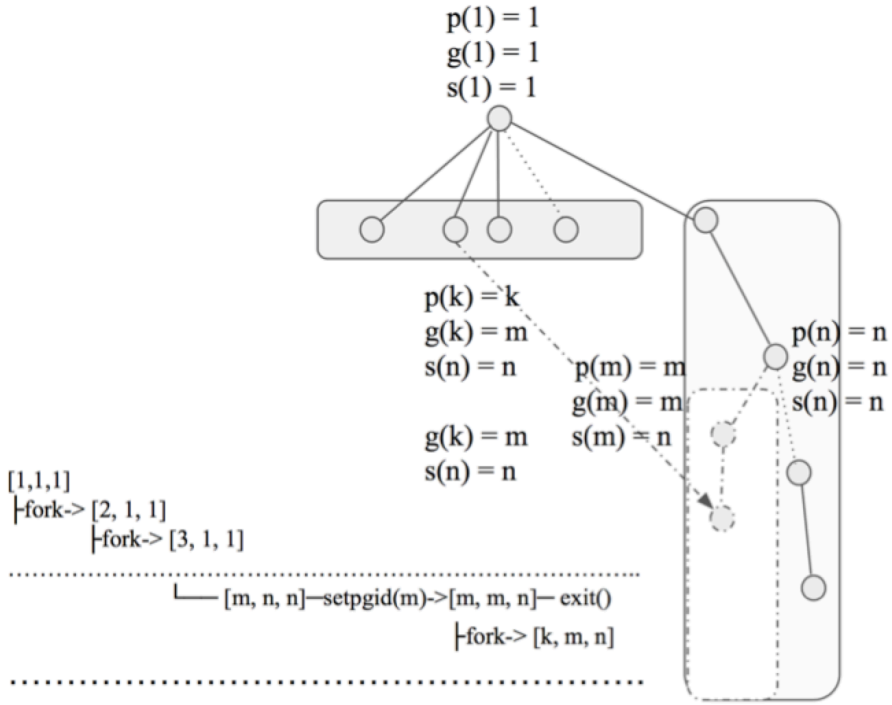
Experiment Setting

- Analyzer vs profile-based techniques:
  - ‘slow’: strace-based
  - ‘progressive’: perf-based
- Two profiles of tests:
  - ”Simple-load”: two simple cases without reparents
  - “Heuristic”: high-rated reparents

Test Profiles and Cost Metrics:







"Simple-load" profile of tests

Reverse reparent-highloaded "heuristic" test

$$C_{overall} = C_s + C_r$$

where C_s , C_r – suspending and resuming overheads respectively.

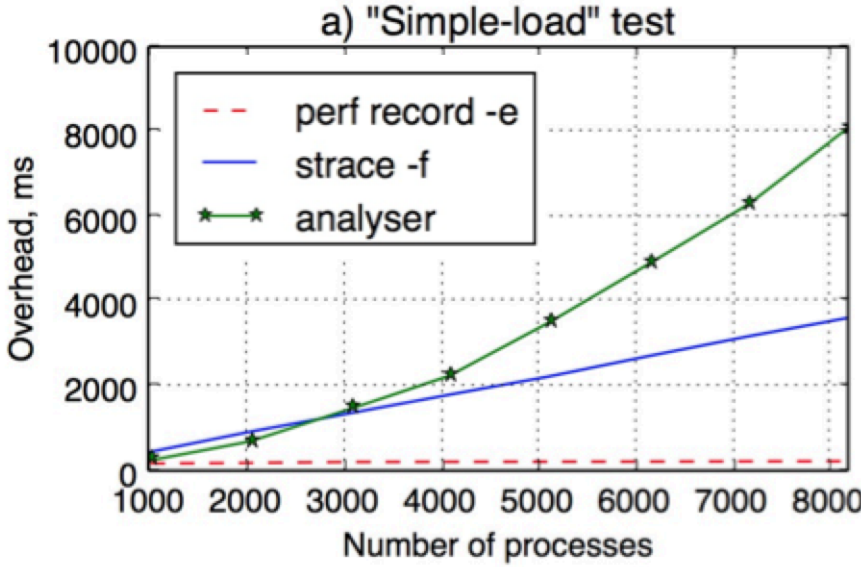
Experimental Results

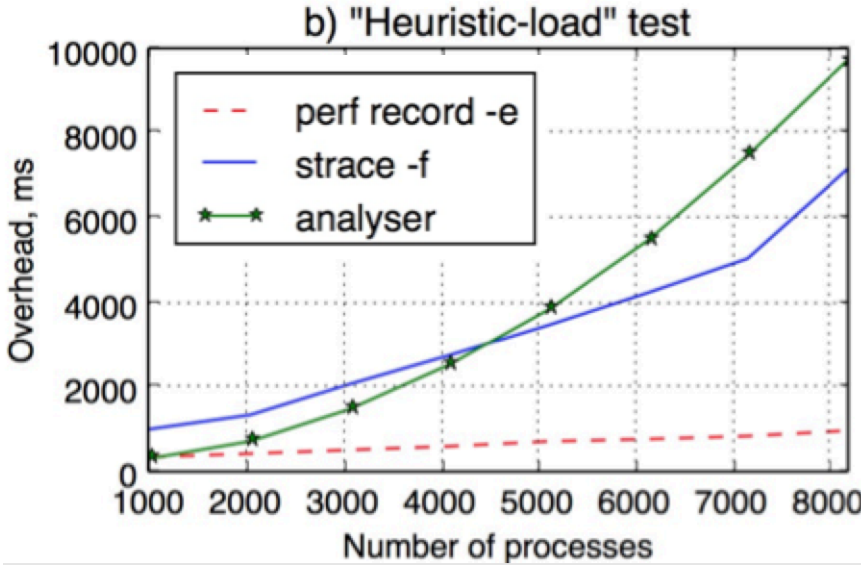
Table 1: The "Simple-load" test results

Number of Processes	Strace	Perf	Analyzer
1024	447ms	173ms	245ms
2048	921	184	693
3072	1356	206	1454
4096	1798	210	2243
5120	2228	213	3500
6144	2708	217	4878
7168	3174	223	6284
8192	3609	225	8070

Table 2: The "Heuristic-load" test results

Number of Processes	Strace	Perf	Analyzer
1024	1010ms	350ms	316ms
2048	1342	428	718
3072	2070	515	1479
4096	2753	601	2529
5120	3475	718	3843
6144	4242	771	5481
7168	5049	843	7483
8192	7150	972	9723





Conclusions

- The solution is feasible for the simplest syscalls restoring
- Is better than ptrace on < 3000-4000 processes
- Complex rules should be handled accurately

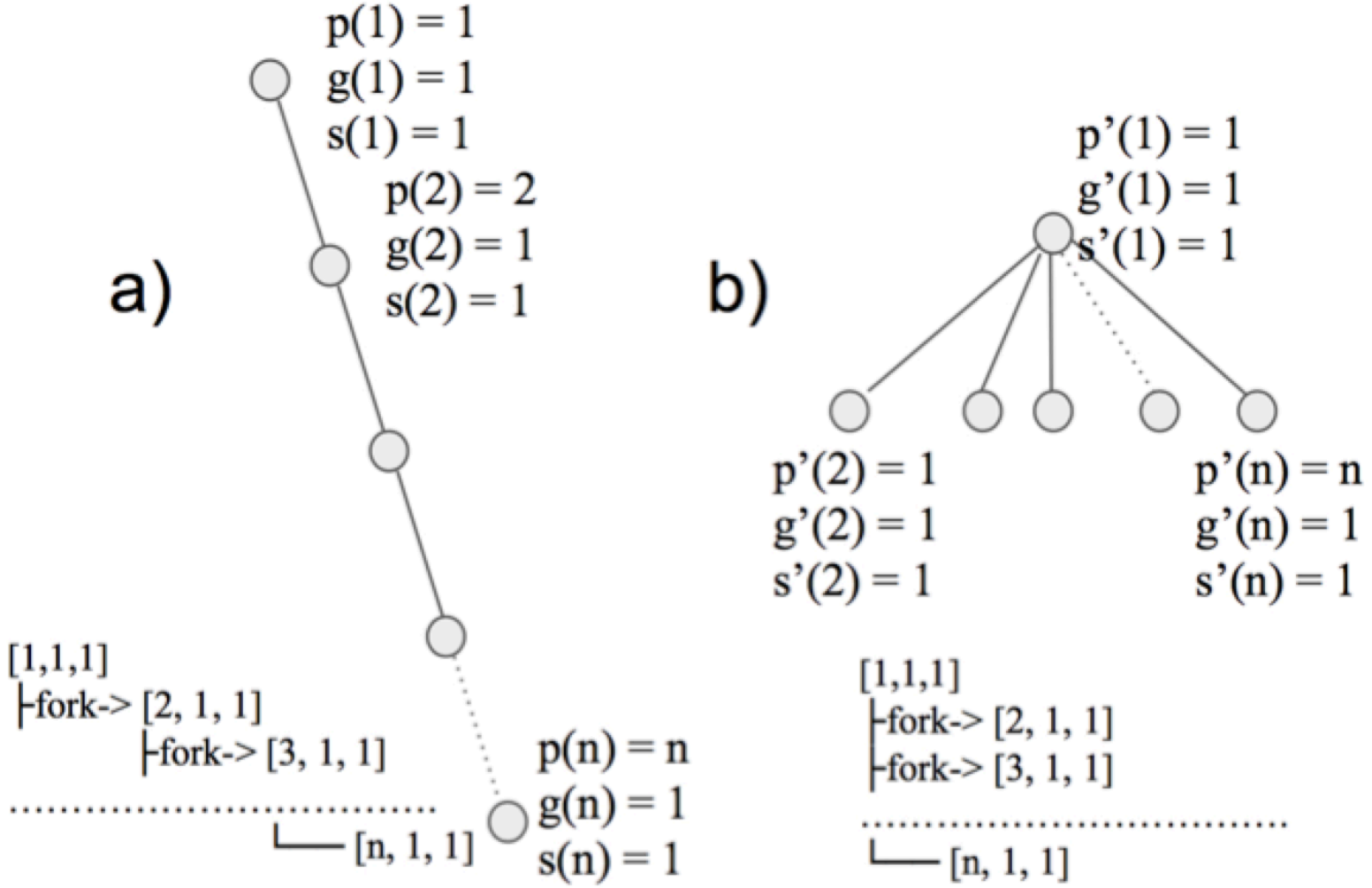
  → drawback
- Ways to improve:
  - New syscalls/features (file descriptors, memory etc)
  - New methods desingning

Thank You For Attention !

Have any questions?

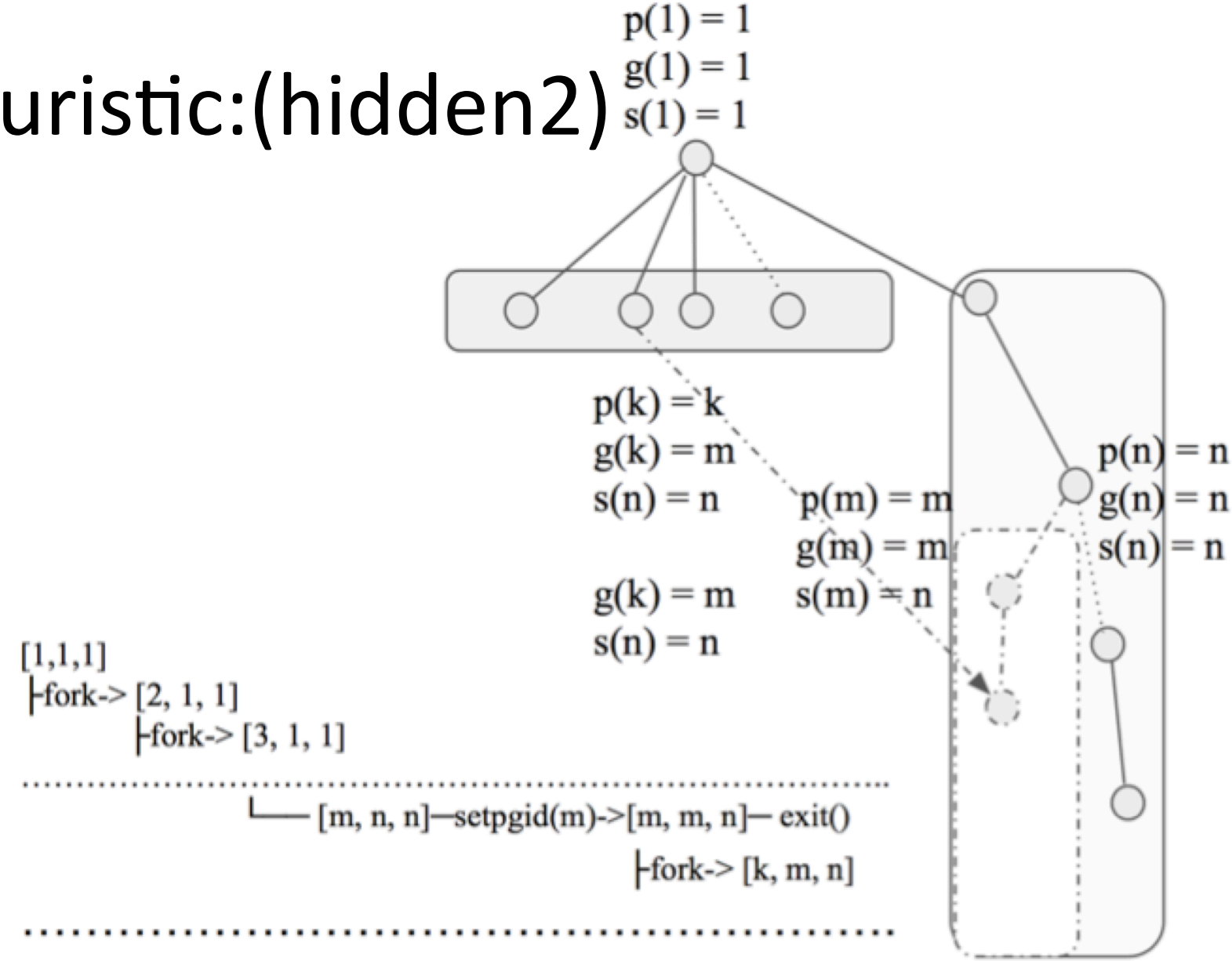
nefanov90@gmail.com

Simple-load (Hidden slide)



"Simple-load" profile of tests

Heuristic:(hidden2)



Reverse reparent-highloaded "heuristic" test