Virtuozzo

CRIU

# Restoring process trees with child-sub-reapers, nested pid-namespaces and inherit-only resources.

Pavel Tikhomirov
Software developer at Virtuozzo

LPC, Dublin, Ireland, Sept 12-14, 2022

# Agenda

- Brief overview of Process tree checkpoint/restore
- Algorithms for simple cases
- Complex case (PR_SET_CHILD_SUBREAPER / CLONE_PARENT)
- Way to solve it: CABA (Closest Alive Born Ancestor)

# Brief overview of Process tree checkpoint/restore

- CRIU knows nothing about the history of creation
- Process tree can change when processes fork/exit
- On father exit, children are reparented to reaper:
  - To pid namespace/host init (CLONE_NEWPID)
  - To child-subreaper (PR_SET_CHILD_SUBREAPER)
  - To alive thread of father
- Session (sid) is an example of special resource which:
  - can only be created by specific process - session leader (pid == sid)
  - non session leader can only inherit session
  - can inherit “born session” from session leaders

# Algorithms for simple cases

Virtuozzo

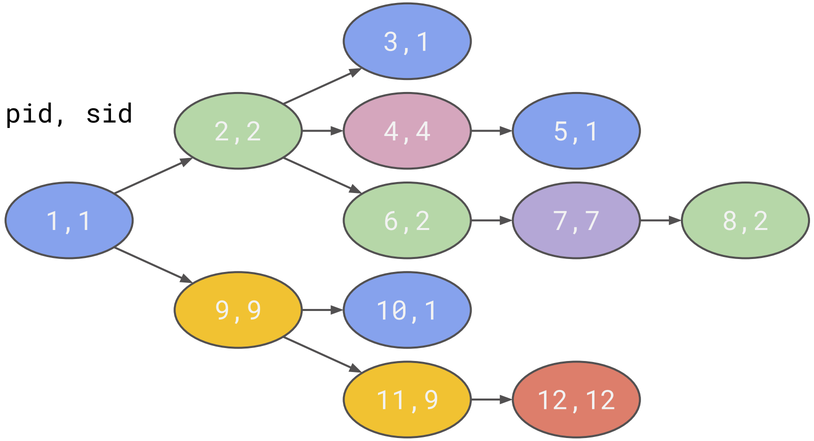
# Algorithms for simple cases: fork (clone) and setsid

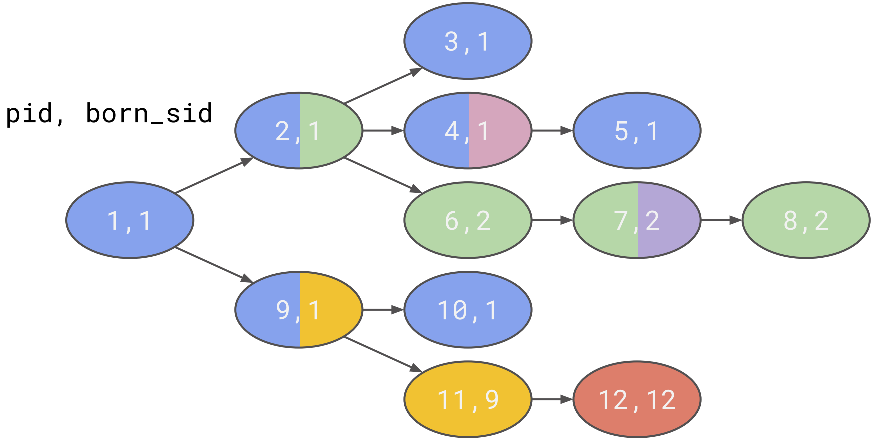
1. Traverse process tree and add all non-leaders to the search list.
2. While search list is not empty:
   a. Pop first item
   b. For session leaders we check their born_sid for others sid
   c. If checked sid is equal to either parent sid or born_sid continue
   d. Else set parent born_sid to item’s sid
3. Traverse process tree forking processes in process tree order
   a. First fork children with sid or born_sid equal to parent born_sid
   b. Setsid in parent
   c. Then fork other children (with sid or born_sid equal to parent sid)

note: this does not handle CLONE_PARENT from session leader

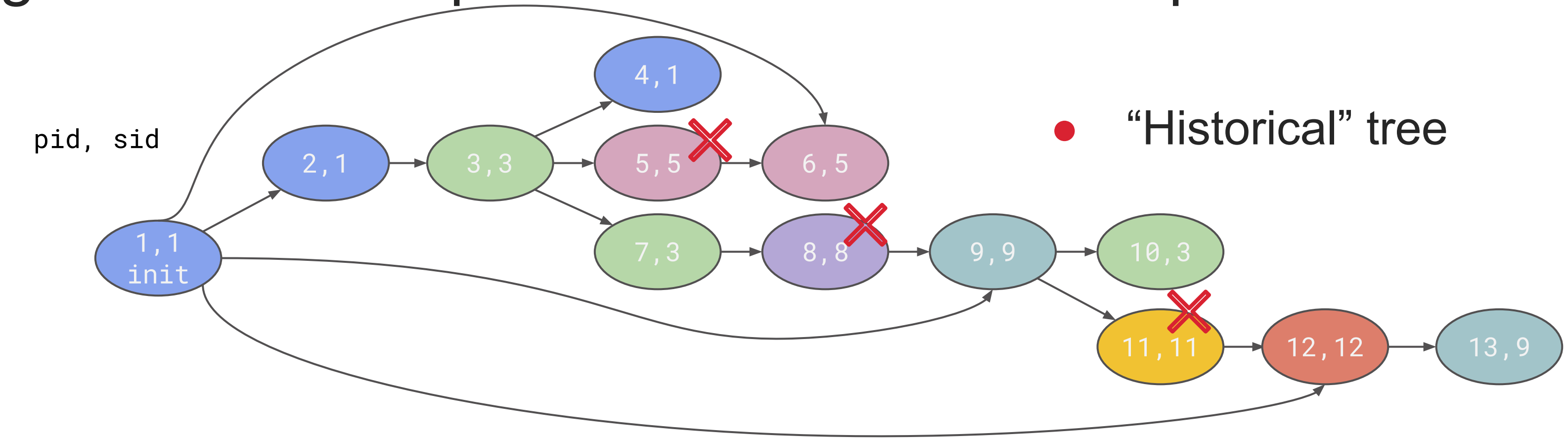
# Algorithms for simple cases: fork and setsid

## Born sid example:

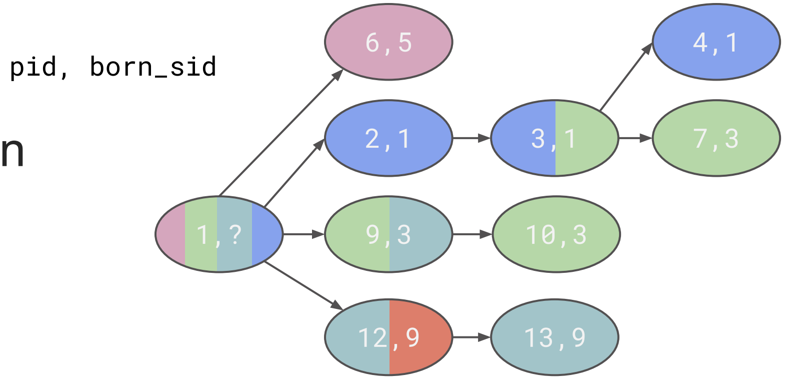




# Algorithms for simple cases: + exit and one pidns



- “Historical” tree



- Final tree (after reparenting)
- Previous algorithm can’t handle born session “collision” here
- Assume no “external” sids

✗ - exit

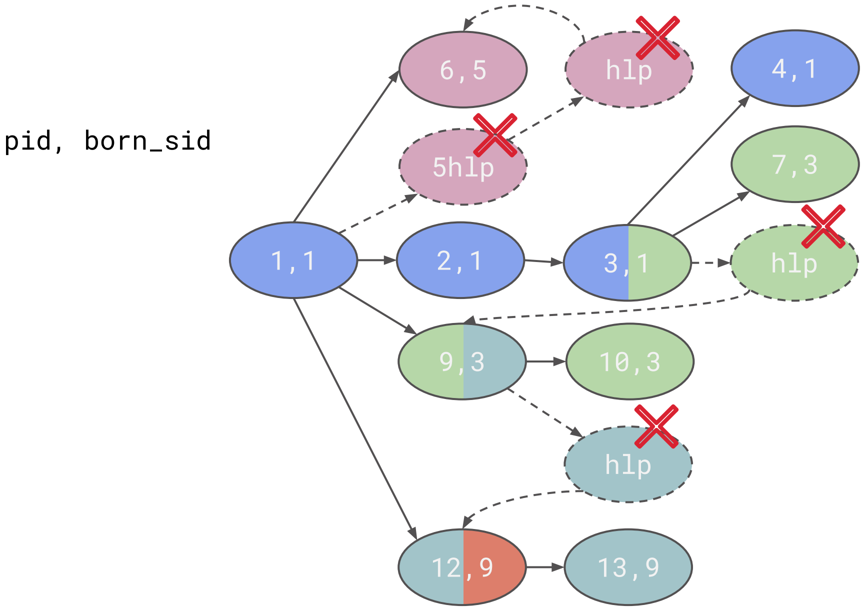
# Algorithms for simple cases: + exit and one pidns

1. If we have session without leader
   a. add leader helper as a child of init
2. Set born_sid-s in a same manner as in previous algorithm except for init
3. For each process which is a child of init
   a. For session leaders we check their born_sid for others sid
   b. If checked sid is equal to init sid continue
   c. Else we search the session leader for the checked sid
   d. Add a helper process in session leader children
   e. Move the checked process into helper children
4. Fork processes the same way as in previous algorithm
5. At the end we exit from helper processes and get right tree via reparenting

note: except for highlighted this is implemented in mainstream CRIU [1], also in CRIU we create helpers for missing processes, e.g. for process groups and accessed /proc/<pid> files

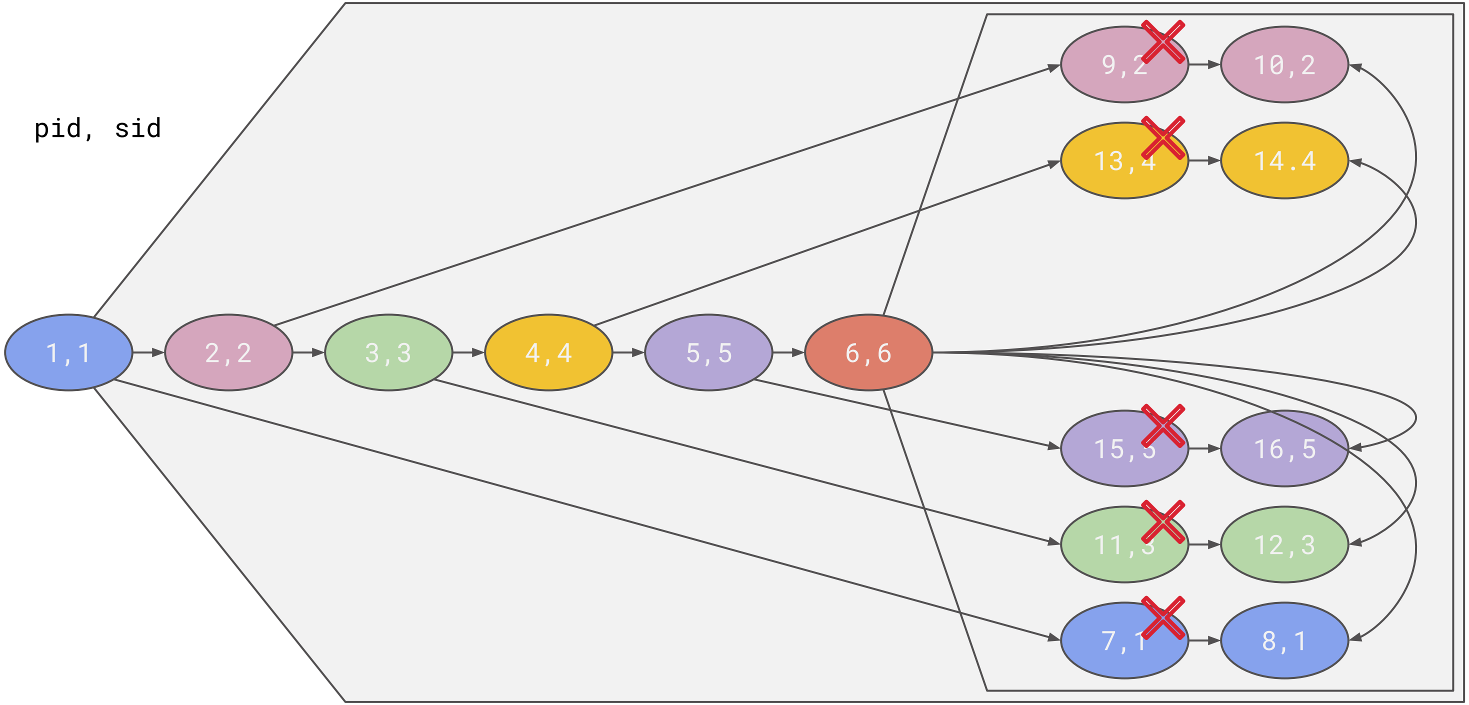
Virtuozzo

# Algorithms for simple cases: + exit and one pidns



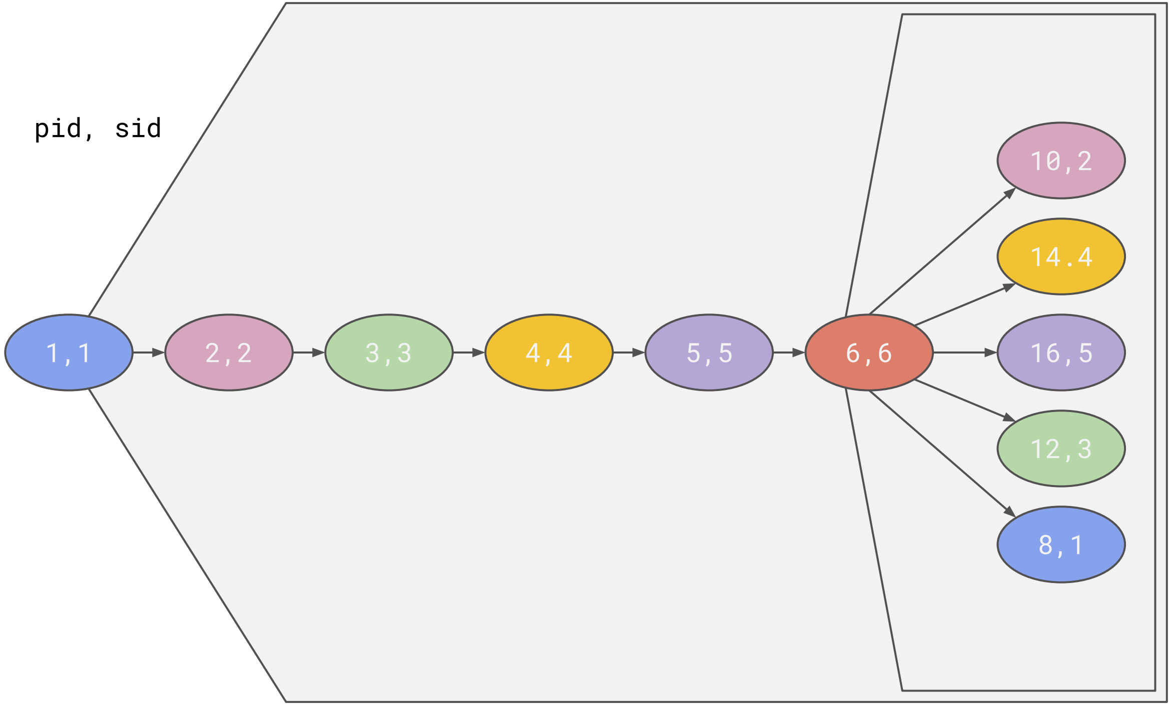
note: The tree with helpers can be different to “historical” tree
note: If there would be any other “only inheritable” resource which should be inherited from 3 to 6 or from 7 to 9 we would have problem here

# Algorithms for simple cases: + nested pidns



note: All pids/sids here and after are shown from root pid namespace

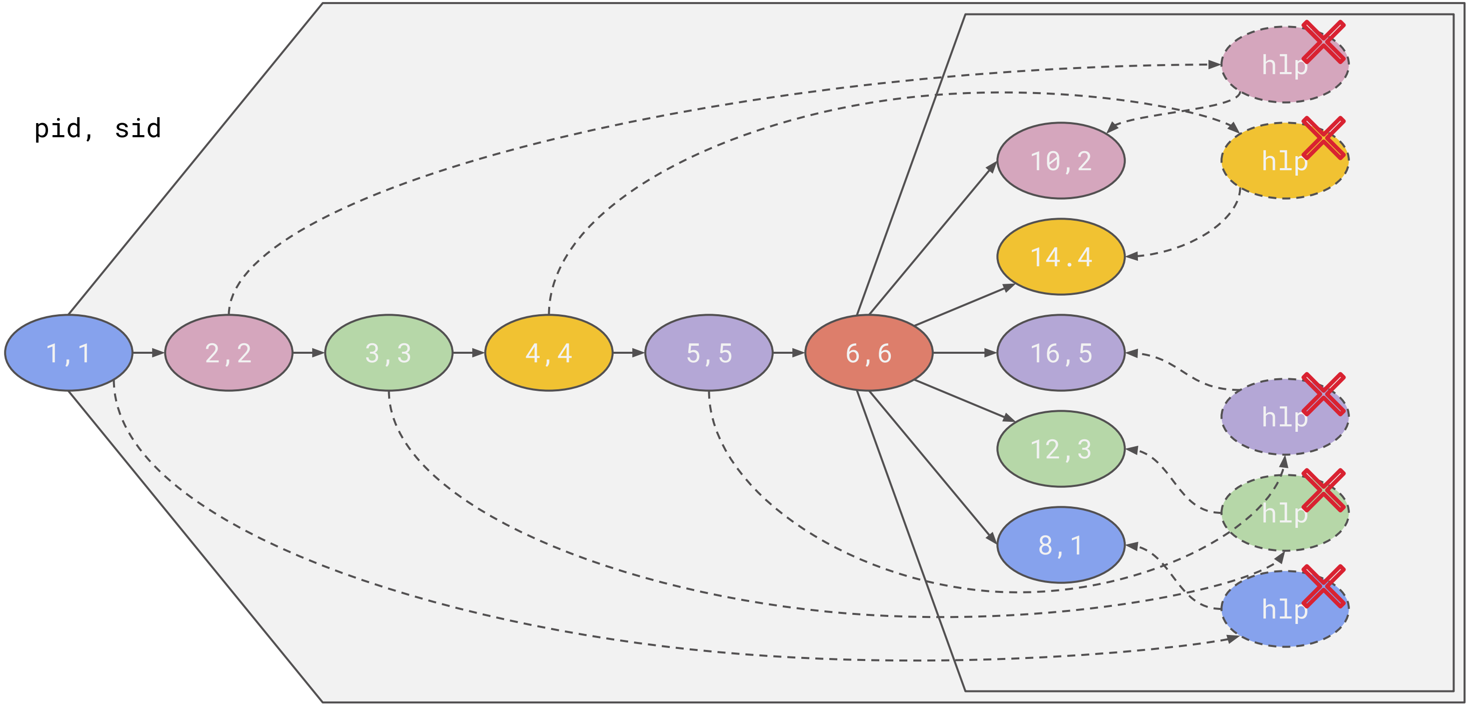
# Algorithms for simple cases: + nested pidns



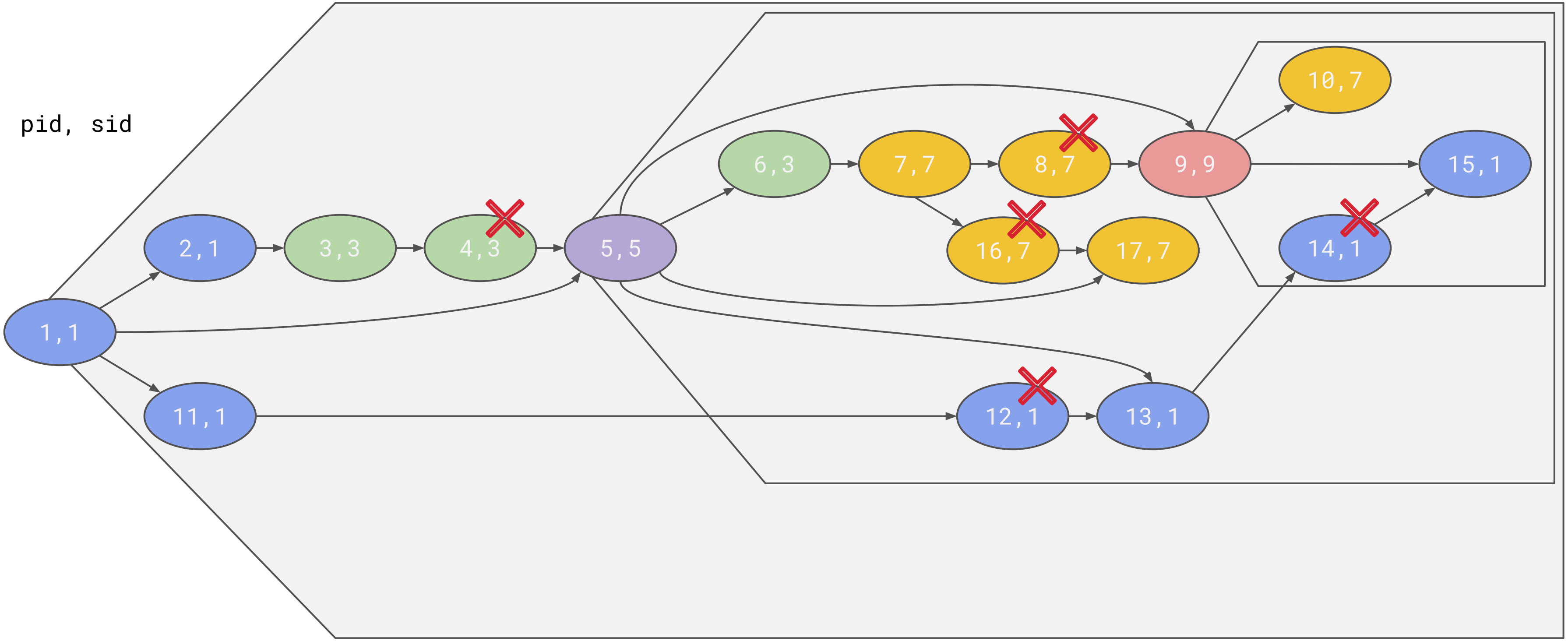
# Algorithms for simple cases: + nested pidns

1. If we have session without leader
   a. find proper pid namespace for a leader going up for each zero in multilevel sid
   b. add leader helper as a child of init of this proper pid namespace
2. For each pid namespace init - current init
   a. walk over processes subtree including other inits but excluding their children and setup “born sid” for them (excluding current init)
   b. when reparenting branches which can’t inherit sid from their init create a helper in current pid namespace connected to the session leader
3. When forking the tree we create pid namespace inits with CLONE_NEWPID
4. When parent pidns is different to child pidns and child is not init we do setns before forking
5. After all is forked all helpers exit and we get desired tree

# Algorithms for simple cases: + nested pidns



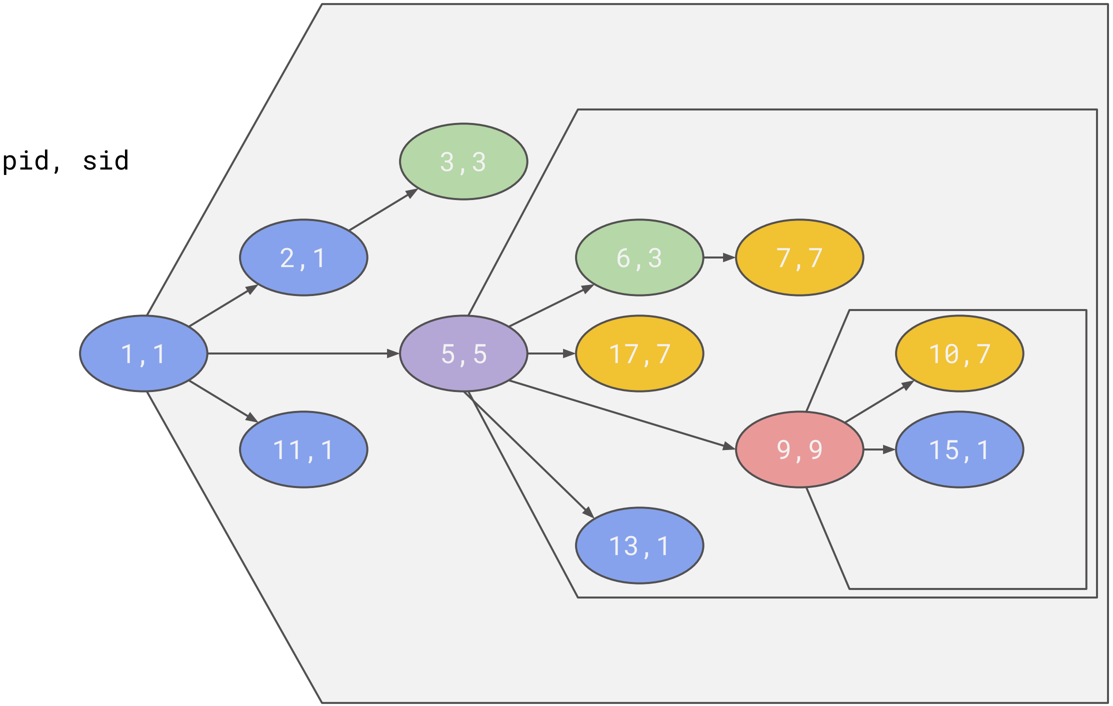
# Algorithms for simple cases: + nested pidns (other example)



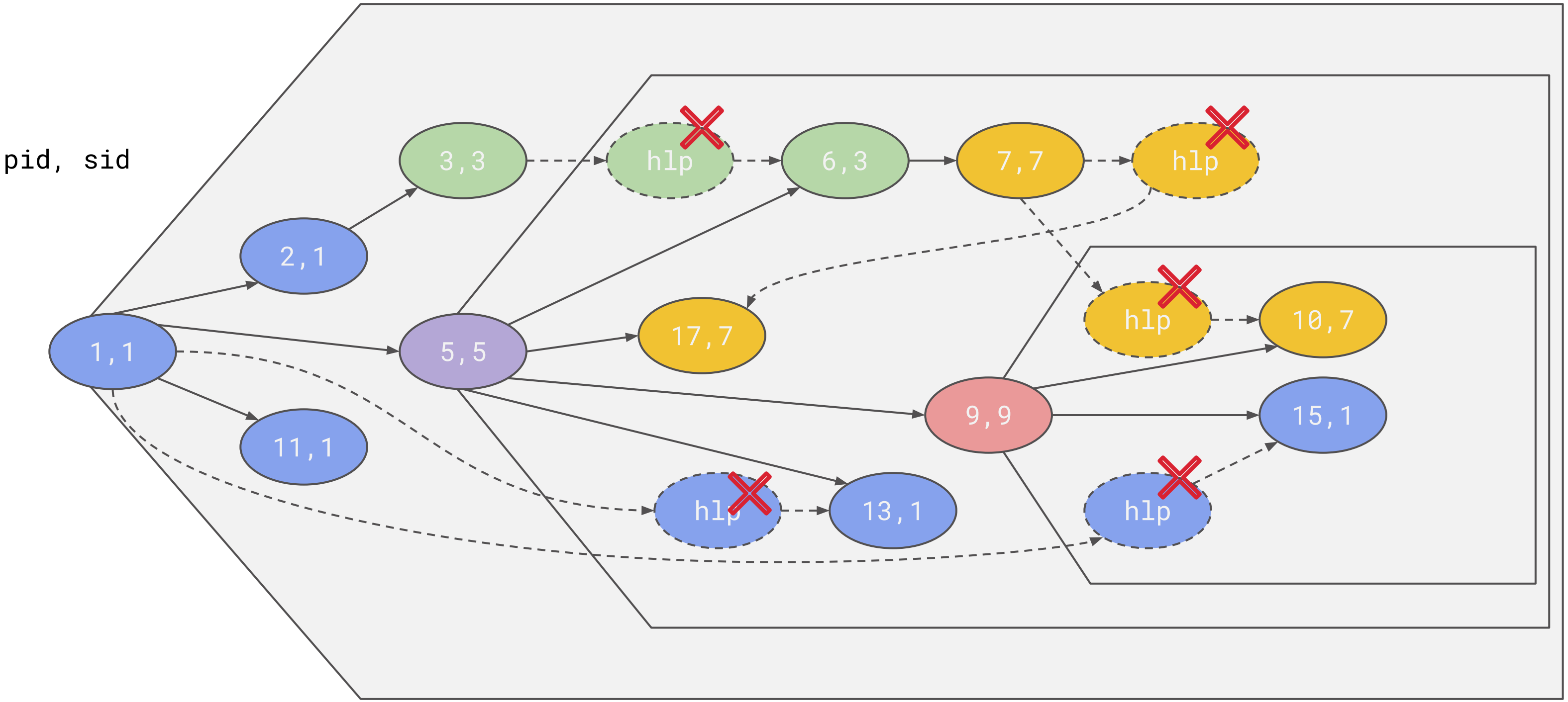
This one is almost complex

Virtuozzo

# Algorithms for simple cases: + nested pidns (other example)



# Algorithms for simple cases: + nested pidns (other example)



Virtuozzo

# Algorithms for simple cases: + nested pidns (notes)

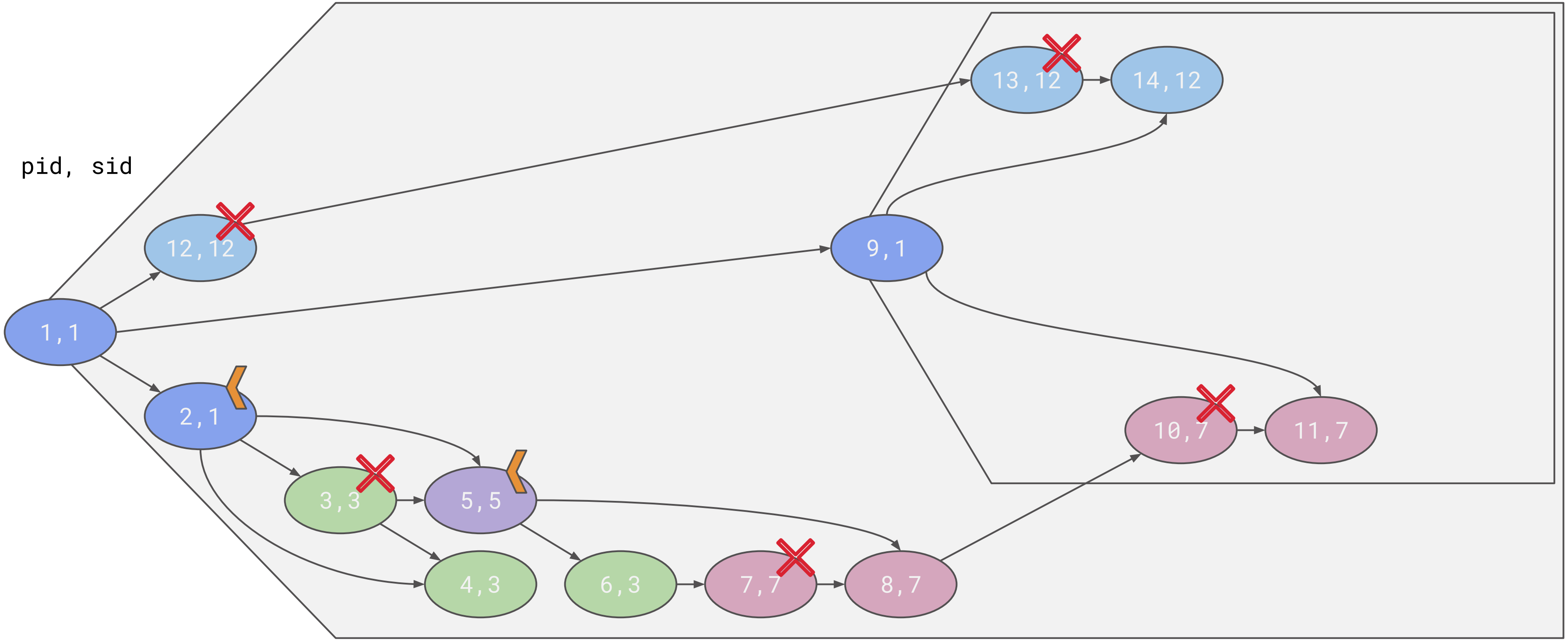
- note: we set pids via ns_last_pid or clone3 settid in each pidns level that’s why all sids are also right in all pidns levels
- note: in case of nested pid namespaces together with nested user namespaces clone3 settid does not work as it can’t set tid on pidns level owned by other userns, I tried to fix it, but got no reaction [2]
- note: process groups become also only inheritable if we enter nested pid namespace
- note: no “external” sids, always do setsid when you enter a container

# Algorithms for simple cases: + simple child-subreapers

- When process becomes a child-subreaper (PR_SET_CHILD_SUBREAPER) it can catch reparenting processes similar to init in it’s subtree
- “Simple case” is when process becomes child-subreaper just after fork and never disables this (similar to what docker shim does)

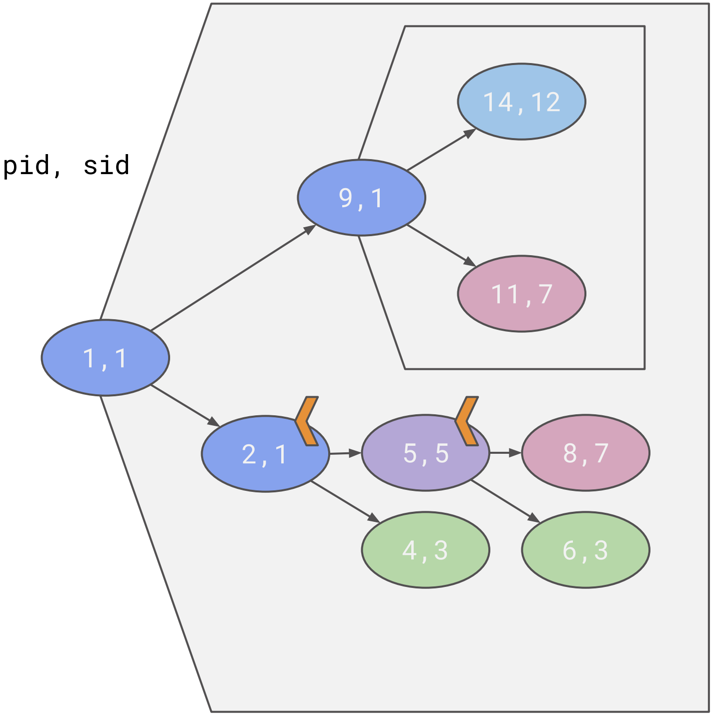
- Handle child-subreapers similar to pid namespace inits
  - though no setns analogy available, so we should check born sids for them
  - accurately handle different cases where child-subreaper does not allow altering the tree by moving init or other subreaper branch to it’s session leader (session leaders should be created as children of child-subreapers sometimes)
- For more details please just look into Virtuozzo CRIU code [3]
- When I was preparing those pictures I found two issues in algorithm =)

# Algorithms for simple cases: + simple child-subreapers



❮ - child-subreaper

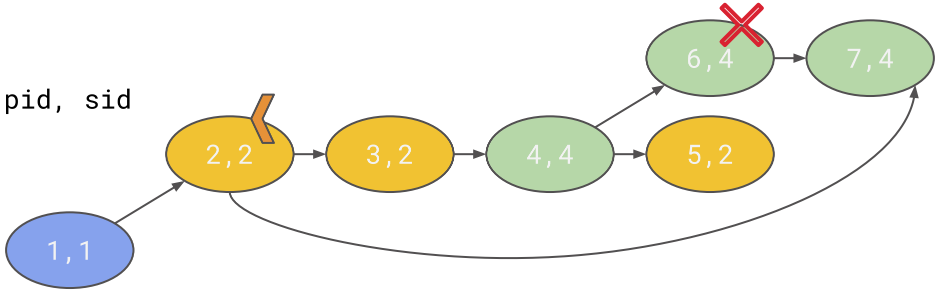
# Algorithms for simple cases: + simple child-subreapers



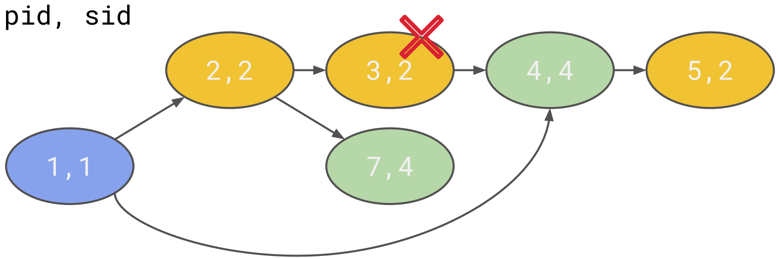
# Compex case

# Complex case: +child-subreapers (+ CLONE_PARENT)

1. 2 is child-subreaper, 6 exits



2. 2 is not child-subreaper anymore, 3 exits

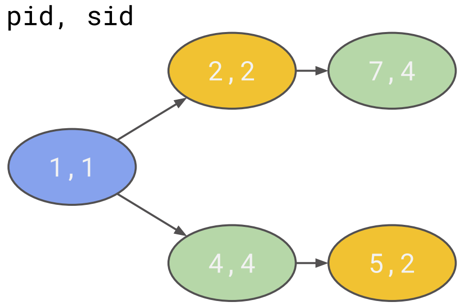


Virtuozzo

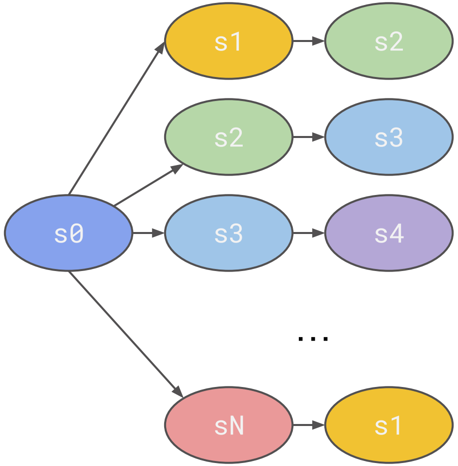
# Complex case: +child-subreapers (+CLONE_PARENT)

Why is it complex?

- Uncertainty - which session was created first? sid 2 or sid 4?
- Other similar example from 10 years ago [4]
- Imagine if we have other resource like sid inherited from 2 to 4, it means that we also need to consider this dependency when restoring the process tree
- With child-subreapers and killing the parent we can reparent any process to any of it’s ancestors in the same pid namespace as the parent
- CLONE_PARENT is equivalent to child-subreaper reparenting
- Before fixes [5] and [6] child-subreapers were even more complex (non uniform reaper, cross ns reaper)



# Complex case: +child-subreapers (+CLONE_PARENT)



note: This can be created similar to previous example

# CABA
# Closest Alive
# Born Ancestor

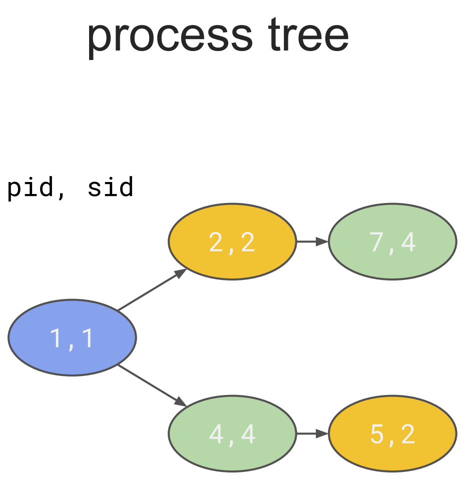
# CABA: Implementation

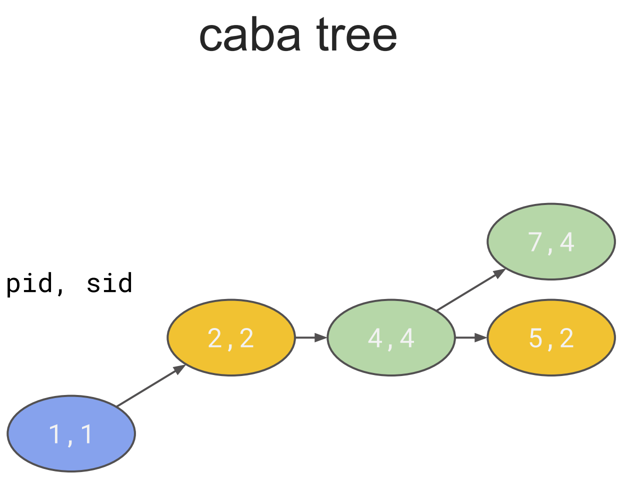
- CABA - Closest Alive Born Ancestor
- CABDs - Closest Alive Born Descendant-s

- Implementation [7] is similar to real_parent/children/sibling (caba/cabds/cabd respectively), except:
  - When process is forked its CABA is set to real_parent (or current for CLONE_PARENT) and process is added to CABDs list of its CABA
  - When father is released (waited/unhashed) its CABDs are moved to father's CABA
  - Except when thread leader of father is available, then thread leader will be new CABA of father CABDs
- "NScaba: …" line is printed to /proc/<pid>/status

Note: I have an Idea that we can enable CABA only per pidns (recursively)
Note: in v2 CABA there was real_parent only (changed in v3)
Note: in v2 CABA exited father was handled (changed in v3)

# CABA





[7] patch to mainstream has selftest with this example

Virtuozzo

# CABA: Restore based on it

1. Take CABA process tree
2. For each session without session leader add missing session leaders
    a. find the smallest subtree containing all processes of the session
    b. from root process in the subtree go up the tree until we are in enough low pid namespace to create the session leader as our child (skip sid zeroes)
    c. add session leader helper to tree as a child of the found process
        i. move the found process children which have the session in subtree to the newly created session leader (via helper)
3. Add helper parent processes for each process which parent != caba
4. Set “born_sid” to all processes
5. Restore processes according to updated CABA tree order handling “born_sid”s same as in first algorithm
6. Kill helpers temporary enabling child-subreaper flag for parent in process tree

Note: not yet implemented

# CABA: Notes

- CABA is self restoring by design
    - if we follow caba tree on restore in CRIU we get the same caba tree
- There are attempts to make a clear mathematical models of processes/resources restore like [8] where order of restore operations is determined based on combined information about all the dependencies between all the resources, unlike CRIU where order is pretty much hardcoded
- But in Linux we generally don’t have information about all resources dependencies
    - new resources can appear
    - some dependencies may be unclear or hard to describe in model
- CABA gives an ability to have as much info about process order as we can
- CRIU is not considering per-thread children as on reparent they can move unpredictably
- CABA is not affected by pid-reuse as “historical” tree would be

# Links

[1] Mainstream CRIU current implementation:
https://github.com/checkpoint-restore/criu/blob/6128eb6185b4/criu/pstree.c#L681

[2] [PATCH] clone3: add option to change owner of newly created namespaces
https://lore.kernel.org/lkml/20210402155131.119872-1-ptikhomirov@virtuozzo.com

[3] Virtuozzo CRIU simple child-subreapers + nested-pidns case:
https://github.com/OpenVZ/vzcriu/blob/1f87bcd638b5/criu/pstree.c#L1573

[4] “Real life task: How to restore process tree in Linux”, 2013, Andrei Vagin, in Russian:
https://habr.com/en/post/195330

[5] commit "prctl: propagate has_child_subreaper flag to every descendant"
https://github.com/torvalds/linux/commit/749860ce2427

[6] commit "exit: fix the setns() && PR_SET_CHILD_SUBREAPER interaction"
https://github.com/torvalds/linux/commit/c6c70f4455d1

[7] [PATCH v3 0/2] Introduce CABA helper process tree
https://lore.kernel.org/lkml/20220908140313.313020-1-ptikhomirov@virtuozzo.com

[8] Efanov, Nikolay. “On Synthetic Process Trees Reconstruction Using Graph-Based Operation Restore Model.” 2020 International Conference Engineering and Telecommunication (En&T) (2020): 1-5.