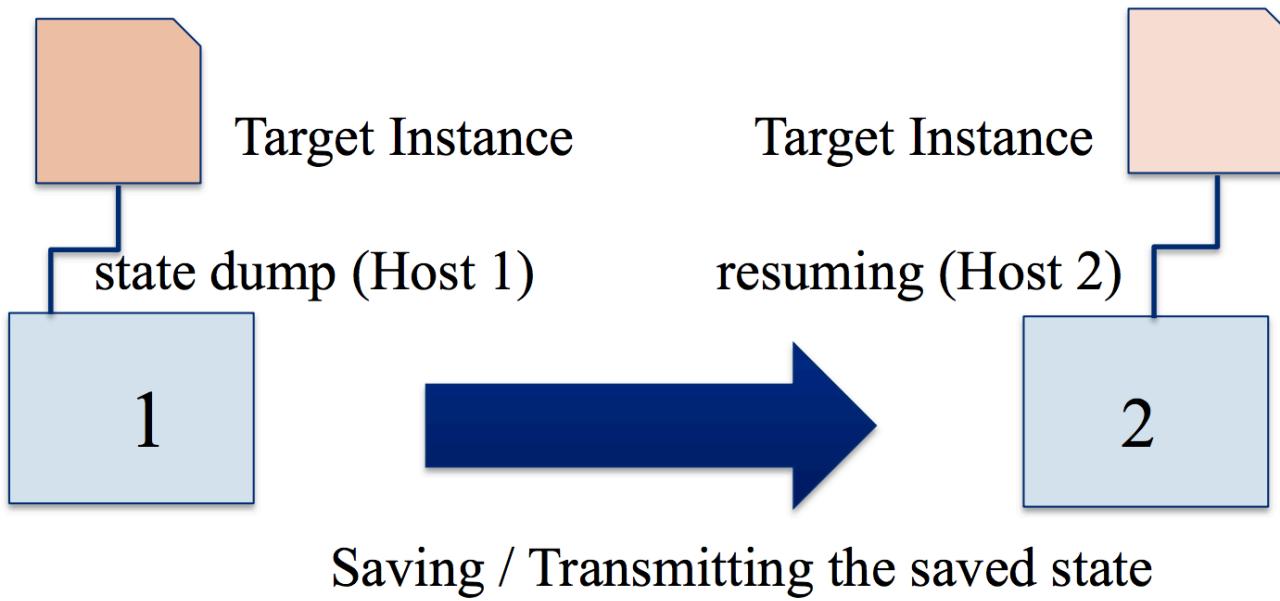


# О полурешетке состояний процессов Linux

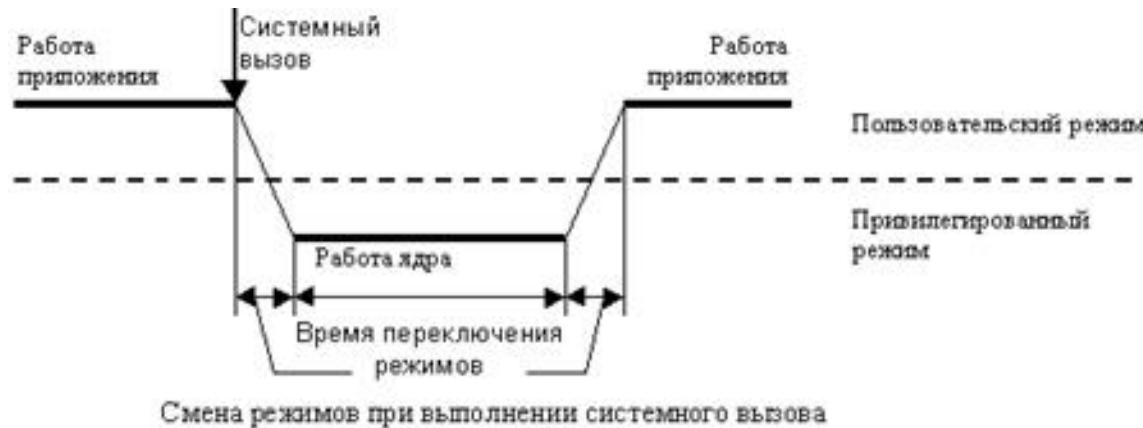
Николай Ефанов, МФТИ

# Сохранение/восстановление состояния среды исполнения



# Процесс в ОС:

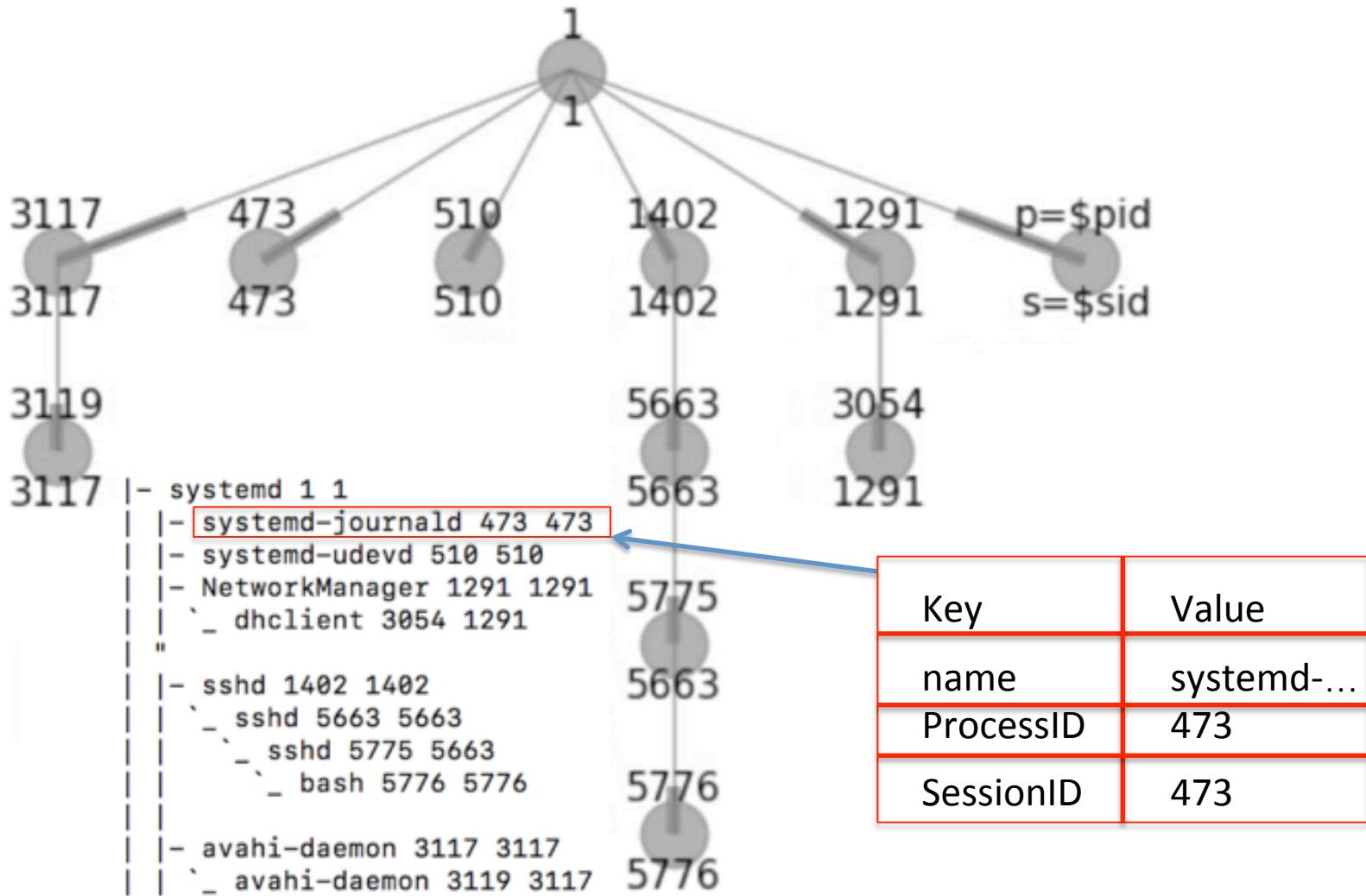
- «Среда исполнения» – «то, в чём работают программы» (исполняются потоки, выделяется память, настраивается окружение, интерфейсы)<sup>1</sup>
- Набор атрибутов: идентификаторы, выделенные ресурсы → атрибуты исполнения
- Способ изменить атрибут – обращение к ядру ОС – Системный вызов.



- (Unix-like): процессы порождаются наследованием от процессов + (опционально) заменой виртуальной памяти (fork + exec\*)  
→ Процессы сгруппированы в дерево

1. Mirkin A., Kuznetsov A., Kolyshkin K. Containers checkpointing and live migration // Proceedings of the In Ottawa Linux Symposium. 2008. V. 2.

# Дерево процессов Linux



# Текущие результаты

- Существует ряд исследовательских и промышленных решений, решающих задачу эвристически (CRIU, DMTCP, BLCR)
- Разработаны методы реконструкции на атрибутных грамматиках:

Efanov N. N., Emelyanov P. V. Linux Process Tree Reconstruction Using The Attributed Grammar-Based Tree Transformation Model // In Proceedings of the 14th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR'18). 2018. ACM, NY, USA, Article 2, 7 pages.

- Используются данные, снятые с реальных систем. Вопрос валидации произвольных деревьев не рассматривался.

# Текущая цель:

- Проанализировать промежуточные представления как частично-упорядоченные множества состояний процессов Linux
- Разработать критерий соответствия для произвольного дерева дереву процессов
  - Критерий упростит генерацию синтетических тестов, так как число различных деревьев процессов очень велико для перебора, даже с учётом индексного автоморфизма [Ефанов, Емельянов, 2017].

# Оценки числа деревьев

- Вызов fork(): порождение потомка
- В строчной записи дерева (p s [children]) [2]: (\* \* [\*]) → (\* \* [\* /2 , /3])

2. Ефанов Н. Н., Емельянов П. В. Построение формальной грамматики системных вызовов // М. Информационное обеспечение математических моделей. 2017. С. 83–90.
3. Efanov N. N., Emelyanov P. V. Constructing the formal grammar of system calls // In Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR'17). 2017. Article 12. 5 pages.
4. Cayley A. A theorem on trees.// Collected Mathematical Papers. 1897. V. 13, P. 26–28.

Оценка количества различных деревьев процессов, поддерживающих системный вызов fork() приводится в работе [2]:

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

Данный результат в точности соответствует суммированию количества различных остовных деревьев на  $n$  вершинах, получаемых из формулы Кейли [2, 3, 4], с учётом одной выделенной вершины для корневого процесса.

# Промежуточные представления

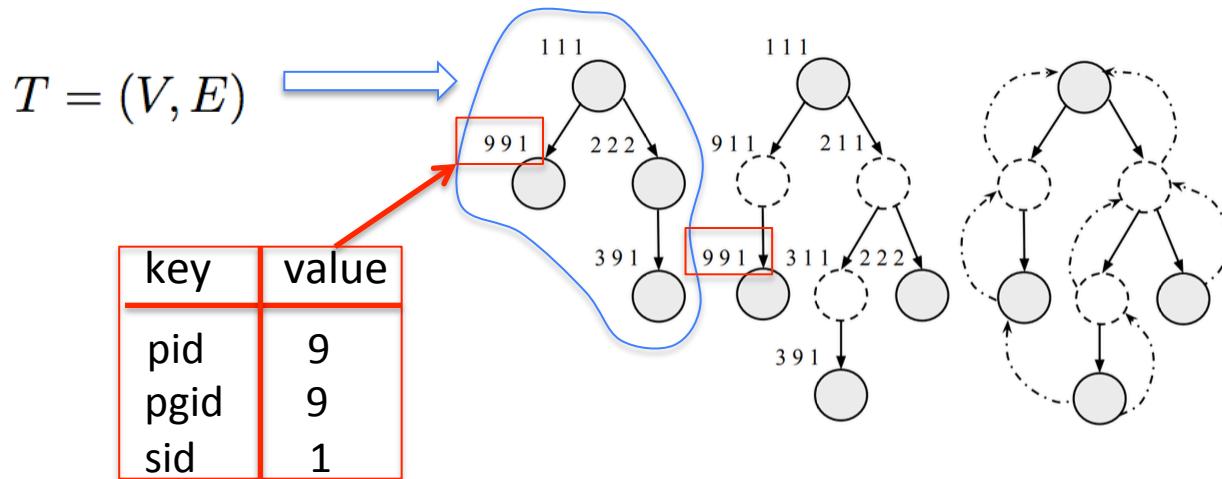


Рис. 1: Исходное и промежуточные представления (слева направо): дерево процессов  $T$ , некоторый его граф восстановления  $G(T)$  и дополненный зависимостями (мульти)граф  $G(T) \cup DEP(T)$ . Числа при вершинах – атрибуты-идентификаторы процесса  $P$ , группы  $G$  и сессии  $S$  соответственно. Корневой процесс всегда имеет идентификаторы “1,1,1”.

# Промежуточные представления

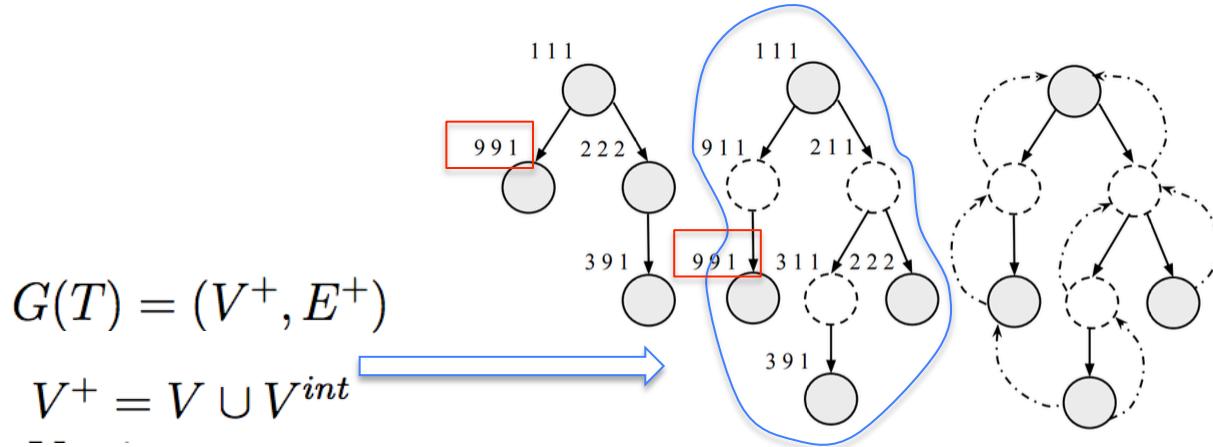


Рис. 1: Исходное и промежуточные представления (слева направо): дерево процессов  $T$ , некоторый его граф восстановления  $G(T)$  и дополненный зависимостями (мульти)граф  $G(T) \cup DEP(T)$ . Числа при вершинах – атрибуты-идентификаторы процесса  $P$ , группы  $G$  и сессии  $S$  соответственно. Корневой процесс всегда имеет идентификаторы “1,1,1”.

# Промежуточные представления

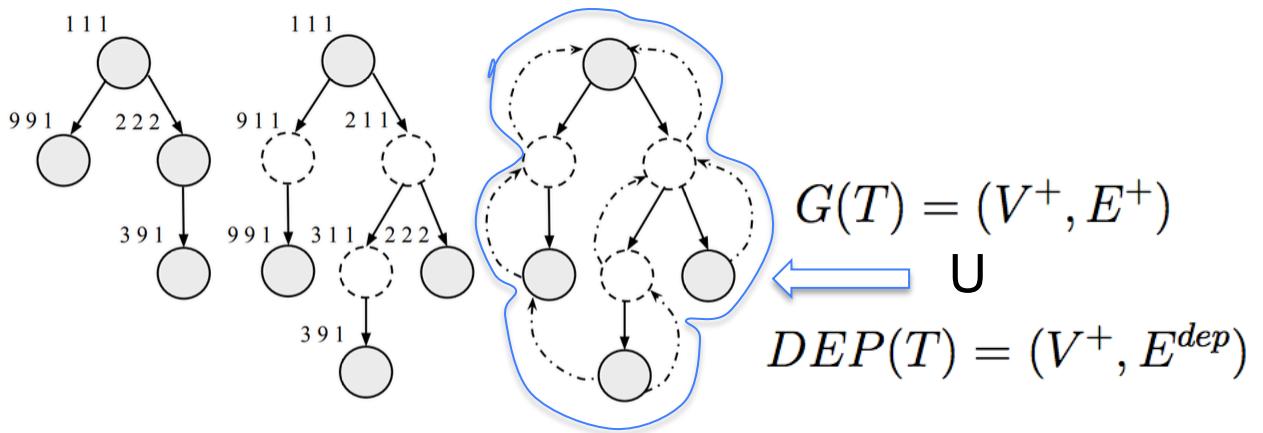


Рис. 1: Исходное и промежуточные представления (слева направо): дерево процессов  $T$ , некоторый его граф восстановления  $G(T)$  и дополненный зависимостями (мульти)граф  $G(T) \cup DEP(T)$ . Числа при вершинах – атрибуты-идентификаторы процесса  $P$ , группы  $G$  и сессии  $S$  соответственно. Корневой процесс всегда имеет идентификаторы “1,1,1”.

# Анализ зависимостей

**ОПРЕДЕЛЕНИЕ 1.** Определим зависимость между вершинами  $u, v \in V^+$  как бинарное отношение  $(u, v)$  с семантикой "для реализации  $u$  требуется сначала реализовать  $v$ ". Следовательно, зависимости определяют частичный порядок на  $V^+$ . Будем говорить, что зависимость происходит по атрибуту  $attr$ , если  $v$  либо создаёт атрибут  $attr$ , либо в его контексте происходит системный вызов, выставляющий данный атрибут в  $v$ .

Используя данное определение, рассмотрим ещё одно промежуточное представление:

**ОПРЕДЕЛЕНИЕ 2.** Графом зависимостей  $DEP(T)$  называется мультиорграф:

$$DEP(T) = (V^+, E^{dep}) \tag{3}$$

где  $E^{dep}$  - множество зависимостей между вершинами.

**ОПРЕДЕЛЕНИЕ 3.** Будем говорить, что атрибут  $attr_2$  доминирует над  $attr_1$ , если  $\forall u \in V^+ : \forall v \in D(attr_1, val(u.attr_1)) \rightarrow val(u.attr_2) = const$ .

**ОПРЕДЕЛЕНИЕ 4.** Доминирующий над  $attr_1$  атрибут  $attr_2$  назовём минимально доминирующим, тогда и только тогда, когда  $\forall u \in V^+ \exists \{v\} \in V^+ : v.attr_2 = u.attr_2, \forall V' = \{v\} \cup v', \text{ и } \forall v' \in V^+ \setminus \{v\} \rightarrow v'.attr_2 \neq const$ .

# Анализ зависимостей

В.Е. Карпов, К.А. Коньков. Основы операционных систем. Курс лекций. Учебное пособие. М.: Интернет-университет информационных технологий, 536 стр., 2005.

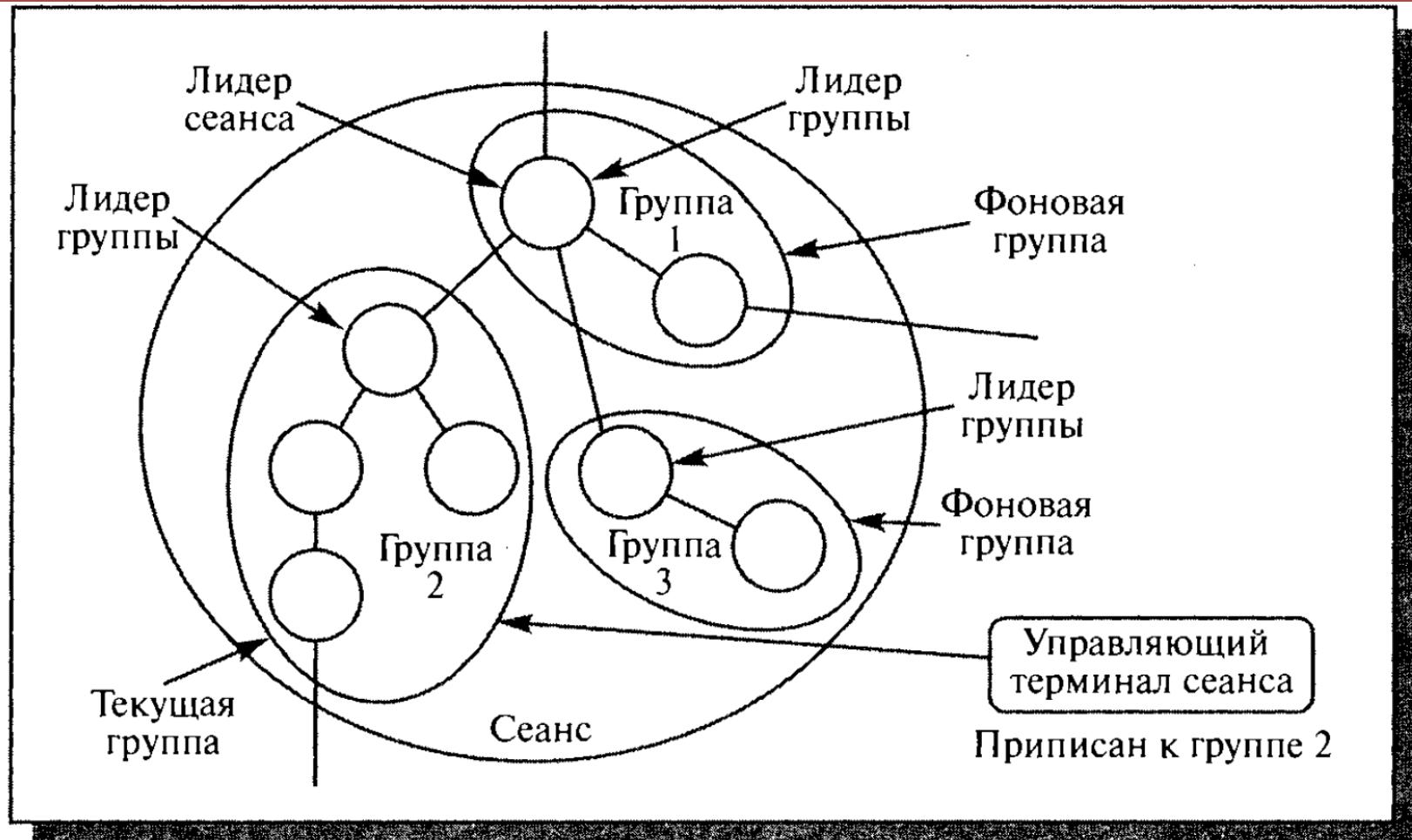


Рис. 12-13.1. Иерархия процессов в UNIX

# Анализ зависимостей

ОПРЕДЕЛЕНИЕ 5. Глубиной изоляции атрибута  $attr_1$  называется число доминирующих над  $attr_1$  атрибутов:  $depth(attr_1) = \sum_{i=2}^{|K|} attr_i : \forall u \in V^+ : \forall v \in D(attr_1, val(u.attr_1)) \rightarrow val(u.attr_i) = const.$

$\forall attr_{key} \in K \rightarrow depth(attr_{key}) < \infty$ , в противном случае, для реализации состояния с таким атрибутом пришлось бы выполнить бесконечное число системных вызовов, так как каждый новый атрибут создаётся через отдельный вызов.

ОПРЕДЕЛЕНИЕ 6. Максимальным по мощности доминирующему атрибутом называется  $attr_x : \forall v \in V^+ \rightarrow depth(v.attr_x) = 0$ .

Для деревьев процессов  $attr_x$  соответствует корневому PID-пространству имён [5], ввиду того, что построение дерева процессов начинается с корневого процесса  $v_{init}$ , который находится в корневом PID-пространстве, и вызовы, и первые вызовы создающие вложенные пространства, происходят именно в нём.

Свойства объектов, введённых в определениях 5 и 6, следуют из технических особенностей прикладной задачи, поэтому принимаются без доказательства.

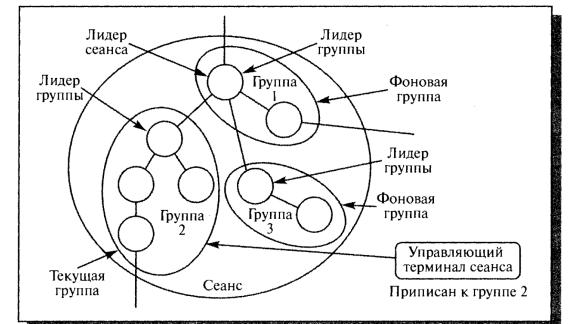
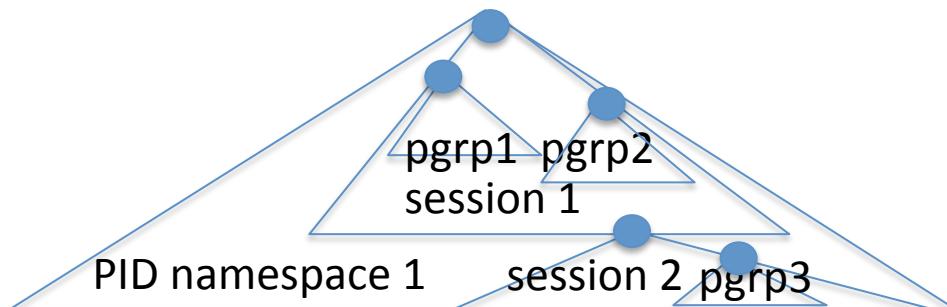


Рис. 12-13.1. Иерархия процессов в UNIX

# Анализ зависимостей

**ОПРЕДЕЛЕНИЕ 5.** Глубиной изоляции атрибута  $attr_1$  называется число доминирующих над  $attr_1$  атрибутов:  $depth(attr_1) = \sum_{i=2}^{|K|} attr_i : \forall u \in V^+ : \forall v \in D(attr_1, val(u.attr_1)) \rightarrow val(u.attr_i) = const.$

$\forall attr_{key} \in K \rightarrow depth(attr_{key}) < \infty$ , в противном случае, для реализации состояния с таким атрибутом пришлось бы выполнить бесконечное число системных вызовов, так как каждый новый атрибут создаётся через отдельный вызов.

**ОПРЕДЕЛЕНИЕ 6.** Максимальным по мощности доминирующему атрибутом называется  $attr_x : \forall v \in V^+ \rightarrow depth(v.attr_x) = 0$ .

Для деревьев процессов  $attr_x$  соответствует корневому PID-пространству имён [5], ввиду того, что построение дерева процессов начинается с корневого процесса  $v_{init}$ , который находится в корневом PID-пространстве, и вызовы, и первые вызовы создающие вложенные пространства, происходят именно в нём.

Свойства объектов, введённых в определениях 5 и 6, следуют из технических особенностей прикладной задачи, поэтому принимаются без доказательства.

**УТВЕРЖДЕНИЕ 1.**  $V^+$  с отношением зависимости образует конечную верхнюю полурешетку.

# Верхняя полурешетка зависимостей

УТВЕРЖДЕНИЕ 1.  $V^+$  с отношением зависимости образует конечную верхнюю полурешетку.

ДОКАЗАТЕЛЬСТВО. Выберем произвольные состояния  $x, y, z \in V^+ : x \neq y$ . Рассмотрим бинарную операцию минимальной верхней границы  $\sqcup$  на  $V^+ : x \sqcup x = x; x \sqcup y = y \iff (x, y) \in E^{dep}$ .  $\sqcup$  по определению идемпотентна. Докажем, что  $\forall x, y \in V^+ \exists z \in V^+ : (x, z) \in E^{dep}, (y, z) \in E^{dep}$ , доказывая параллельно коммутативность и ассоциативность  $\sqcup$ . Рассмотрим ситуации:

1.  $(x, y) \in E^{dep}$  - тогда, очевидно,  $c = y$ . Случай  $(y, x)$  симметричен.
2. Покажем, что  $\{(x, y), (y, x)\} \subset E^{dep} \iff x = y$ .
  - Допустим, что  $\exists x, y \in V^+ : \{(x, y), (y, x)\} \subset E^{dep}$ , и  $x \neq y$ . По определению зависимости, для реализации  $x$  нужно реализовать сначала  $y$ , но для этого нужно реализовать состояние, идентичное  $x$ . То есть на данном шаге нужно выполнить минимум 2 системных вызова, и требуется снова воспроизвести  $x$ .
  - Пусть выполнено  $k - 1$  итераций воспроизведения  $x, y$ , тогда нужно снова воспроизвести  $x$ , и выполнено  $2(k - 1)$  системных вызовов.
  - После шага  $k$  требуется снова воспроизвести  $x$ , следовательно, число системных вызовов не ограничено снизу, что противоречит самой схеме восстановления.

В результате заключаем, что циклические зависимости запрещены, и допустимы лишь формально в случае  $x = y$ , откуда следует коммутативность:  $y = x \sqcup y = y \sqcup x = x$ .

# Верхняя полурешетка зависимостей

УТВЕРЖДЕНИЕ 1.  $V^+$  с отношением зависимости образует конечную верхнюю полурешетку.

3. Пусть  $(x, y, attr_1), (y, x, attr_1) \notin E^{dep}$ .

- Рассмотрим минимальный доминирующий атрибут:  $attr_2$ . Если  $y.attr_2 = x.attr_2$ , и  $\exists x'(attr_2) \in D(attr_2, x.attr_2)$  - создатель ресурса, описываемого атрибутом  $attr_2$ , доказательство завершено. Иначе,  $y.attr_2 \neq x.attr_2$ , и процедура повторяется.
- Пусть выполнено  $k - 1 = depth(attr_1)$  шагов процедуры выше, и  $attr_k$  суть атрибут корневого PID-пространства имён.
- корневое PID-пространство имён, по определению 6, и, после  $k < \infty$ -шагов,  $z = v_{init}$ .

Следовательно,  $(V^+, \sqcup)$  - конечная верхняя полурешетка, с задаваемым зависимостями естественным частичным порядком и максимальным элементом  $v_{init}$ .

□

# Некоторые важные следствия

Полурешеточная упорядоченность  $V^+$  - крайне важный результат. В частности, для проверки корректности дерева процессов.

LEMMA 1. Пусть  $CL(v.attr_1) : V^+ \rightarrow V^+$  - оператор,  $\forall v \in V^+$  определяющий  $f \in V^+$  с минимальным доминирующим атрибутом для некоторого  $attr_1$ . Тогда  $V^+$  монотонный и экстенсивный.

ДОКАЗАТЕЛЬСТВО. Из свойств ресурсов ядра ОС,  $CL$  обладает:

1. Экстенсивностью:  $v \leq CL(v.attr_1)$ . Это свойство очевидно из факта, что атрибут не может появиться в дереве, пока соответствующий ресурс не создан.
2. Монотонностью:  $\forall v, u \in V^+ : v \leq u \rightarrow CL(v.attr_1) \leq CL(u.attr_1)$ . Предположим, что минимальный доминирующий атрибут  $attr_1$  это  $attr_2$ , и  $v \leq u$ . Пусть  $u.attr_2 \neq v.attr_2$ . Тогда ресурс, описываемый  $u.attr_2$  должен быть создан ранее чем ресурс того же типа для  $v$ , либо они не сравнимы. Иначе они равны.

□

# Некоторые важные следствия

ЛЕММА 2. Пусть  $CL$  возвращает состояние, в котором минимальный доминирующий атрибут был создан. Тогда  $CL$  - оператор замыкания, со строго одной неподвижной точкой для каждого  $f = CL(v.attr_1), \forall v \in V^+, \forall attr_1 \in attr$  в случае, если  $V^+$  соответствует порождению корректного дерева процессов.

ДОКАЗАТЕЛЬСТВО. Рассмотрим  $CL(CL(v.attr_1).attr_1)$ .  $CL(CL(v.attr_1).attr_1) = CL(v.attr_1)$  если  $CL(v.attr_1)$  создатель. Следовательно,  $CL$  идемпотентный.

Случай, если  $CL$  для вершины  $v$  с заданным атрибутом  $v.attr_1$  будет иметь более одной неподвижной точки, будет означать создание более одного ресурса с одним и тем же идентификатором в одной области действия данного идентификатора, что означает конфликт идентификаторов.  $\square$

Приведём некоторые результаты, следующие из полурешеточной упорядоченности:

ЛЕММА 3. Подмножество состояний  $V^{++}$ , необходимых для восстановления произвольного  $v \in V^+$ , является главным фильтром  $V^+$ .

Следствие:  $V^{++}$  образуют полную решётку относительно наследованного порядка (по зависимостям).  $CL$  на  $V^+$ , взятые для всех атрибутов  $attr_i \in attr$  также образуют полную решётку (поточечно). [Биркгоф, Г. Теория решёток. — М. : Наука, 1984.]

# Некоторые следствия: необходимость

ЛЕММА 2. Пусть  $CL$  возвращает состояние, в котором минимальный доминирующий атрибут был создан. Тогда  $CL$  - оператор замыкания, со строго одной неподвижной точкой для каждого  $f = CL(v.attr_1), \forall v \in V^+, \forall attr_1 \in attr$  в случае, если  $V^+$  соответствует порождению корректного дерева процессов.

ДОКАЗАТЕЛЬСТВО. Рассмотрим  $CL(CL(v.attr_1).attr_1)$ .  $CL(CL(v.attr_1).attr_1) = CL(v.attr_1)$  если  $CL(v.attr_1)$  создатель. Следовательно,  $CL$  идемпотентный.

Случай, если  $CL$  для вершины  $v$  с заданным атрибутом  $v.attr_1$  будет иметь более одной неподвижной точки, будет означать создание более одного ресурса с одним и тем же идентификатором в одной области действия данного идентификатора, что означает конфликт идентификаторов.  $\square$

То есть создатели атрибутов образуют полную решетку для  $\forall v \in V^+$ . И данное условие должно быть выполнено для любого корректного дерева процессов - необходимое условие получено.

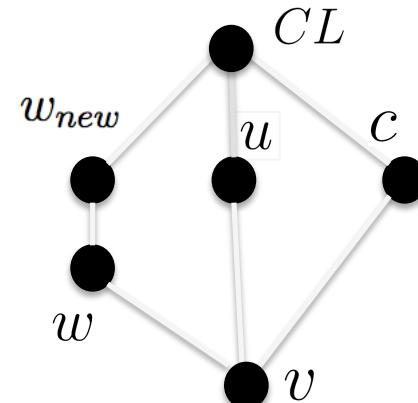
Следствие:  $V^{++}$  образуют полную решетку относительно наследованного порядка (по зависимостям).  $CL$  на  $V^+$ , взятые для всех атрибутов  $attr_i \in attr$  также образуют полную решетку (поточечно). [Биркгоф, Г. Теория решеток. — М. : Наука, 1984.]

# Предложение (достаточность)

Получим достаточное условие.

PROPOSITION 1. Для реконструкции состояния  $u$  :  $u.attr_1 = w.attr_1$  с атрибутом  $attr_1$ , созданным в состоянии  $w_{new}$  и "хранимым" в состоянии  $w$ , из предшествующего состояния  $v$  :  $u.pid = v.pid | (u.pid \neq v.pid \& syscall = 'fork')$  системным вызовом, выполненным в контексте процесса с состоянием  $c$ , достаточно выполнения:  $c.attr_2 = v.attr_2 = w_{new}.attr_2 = u.attr_2$ , и оператор  $CL(w.attr_1) = CL(c.attr_1) = CL(u.attr_1)$  возвращает состояние с минимальным доминирующим атрибутом  $attr_2$  атрибута  $attr_1$ .

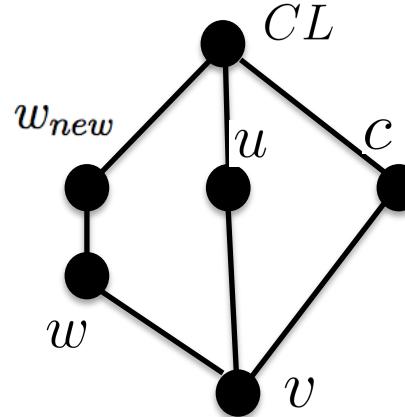
Проверка данного утверждения производится покрытием введённой конструкцией всех рассматриваемых случаев установки атрибута процессу.



Рассмотрим свойства подмножества из предложения  
(такое подмножество мы называем реконструктором):

# Свойства реконструктора

- Полная решётка



LEMMA 4.  $u, v, w, w_{new}, c, CL(u.attr_1)$  формируют полную решётку с частичным порядком, наследованным из  $V^+$ .

ДОКАЗАТЕЛЬСТВО.  $c.attr_2 = v.attr_2 = w_{new}.attr_2 = u.attr_2$ , и  $CL(u.attr_1)$  единица решетки.  $v$  наименьший элемент, и нет зависимостей от  $v$ , но он зависит от  $c.attr_2 = w_{new}.attr_2 = u.attr_2 = CL(u.attr_1)$ . То есть  $v$  - meet-элемент для любого подмножества из подмножеств  $c, u, w_{new}, CL(u.attr_1)$  и  $c, u, w, CL(u.attr_1)$ , а для  $w, w_{new}$  -  $w$  является meet-элементом. следовательно,  $u, v, w, w_{new}, c, CL(u.attr_1)$  - полная решётка с нулем  $v$  и единицей  $CL(u.attr_1)$ .  $\square$

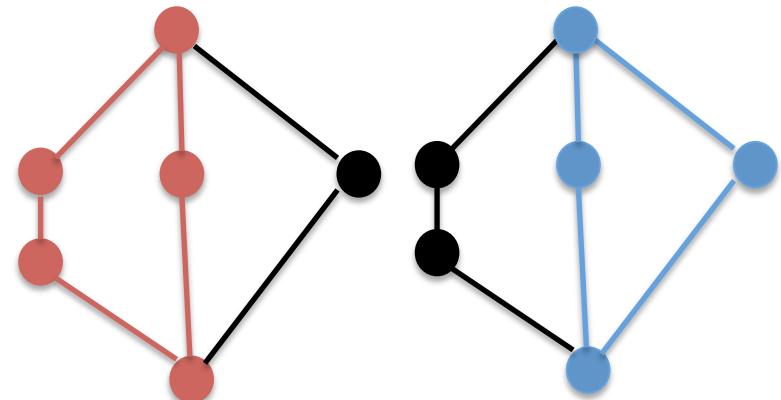
Назовём  $u, c, w, w_{new}$  генерирующим множеством  $Gen(v.attr_1)$ .

Существование  $Gen(v.attr_1) \forall v \in V^+, \forall attr_1 \in attr$ , подходящих для реконструкции - достаточное условие. Теорема (критерий корректности):

ТЕОРЕМА 1. Дерево процессов  $T$  корректно  $\iff (\exists V^+(T)$  верхняя конечная полурешётка:  $V^+(T) = V(T) \cup V^{int} : \forall v \in V^+ \setminus v_{init}) \ \& \ (\forall attr_i \in attr \ \exists Gen(v.attr_i) : Gen(v.attr_i) \subseteq V^+).$

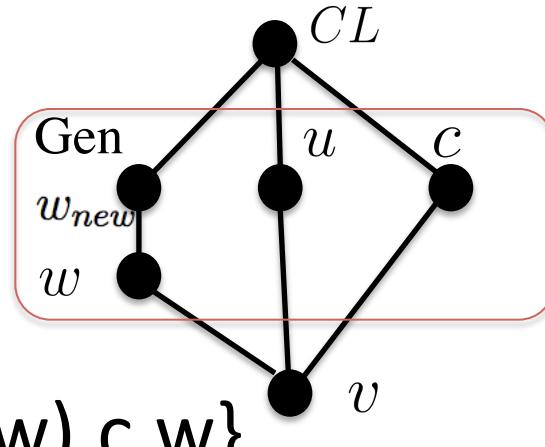
# Свойства реконструктора

- Полная решётка
- Немодулярная
- Недистрибутивная



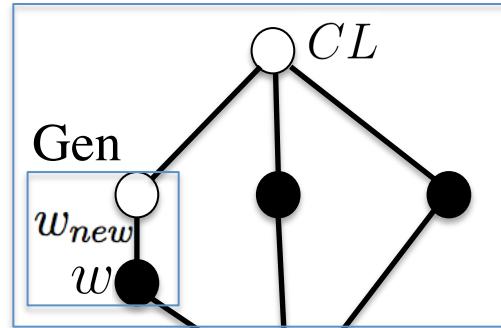
# Свойства реконструктора

- Полная решётка
- Немодулярная
- Недистрибутивная
- «Gen» - множество  $\{v(\text{new}), c, w\}$



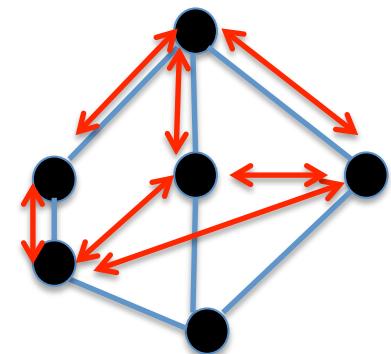
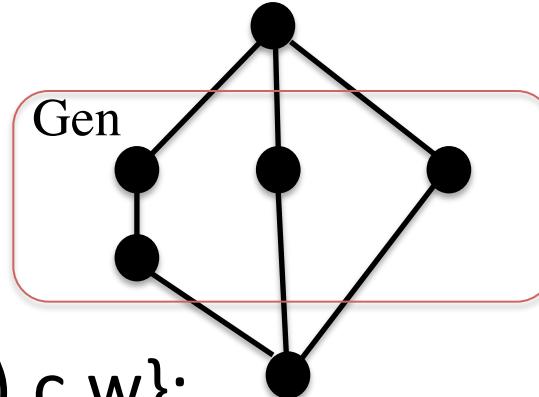
# Свойства реконструктора

- Полная решётка
  - Немодулярная
  - Недистрибутивная
  - «Gen» - множество  $\{v(\text{new}), c, w\}$ :
    - Создатель и носитель: для моментального задания атрибута достаточно носителя, но:
- 



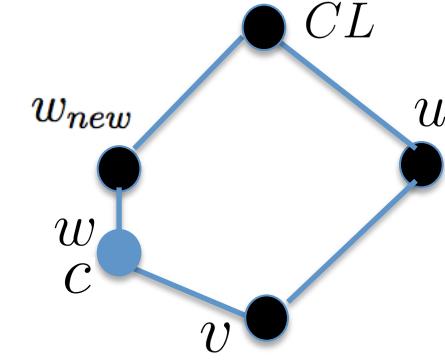
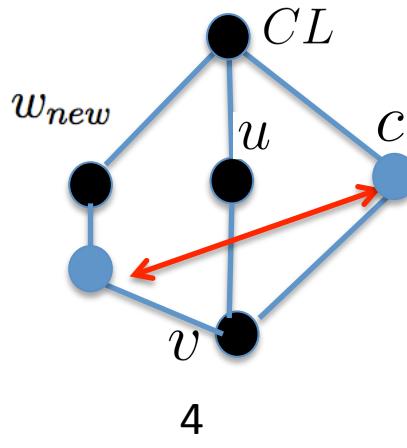
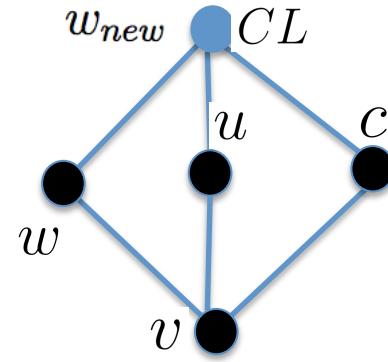
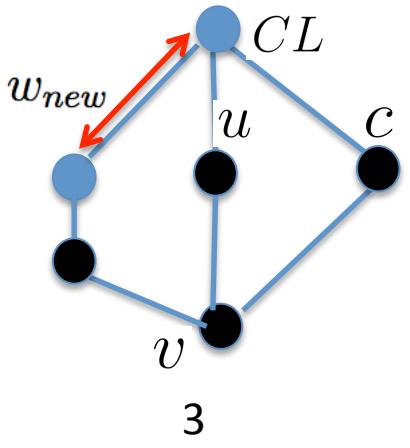
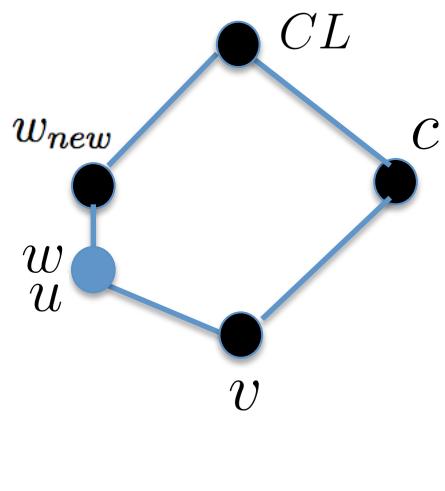
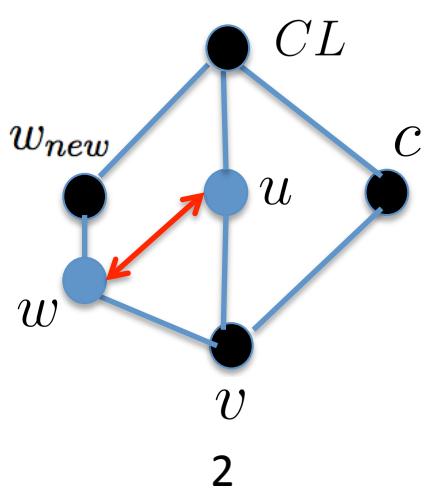
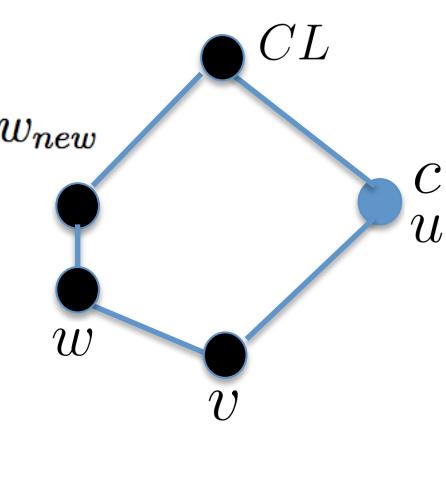
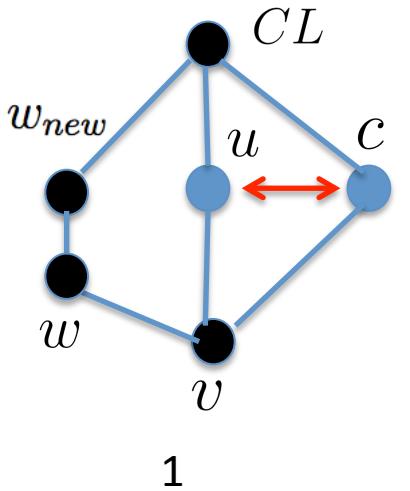
# Свойства реконструктора

- Полная решётка
- Немодулярная
- Недистрибутивная
- «Gen» - множество  $\{v(\text{new}), c, w\}$ :
  - Создатель и носитель: для задания атрибута достаточно носителя, но создатель существовал.
  - Частные случаи:
    - Конгруэнции вершин
    - Доупорядочивания



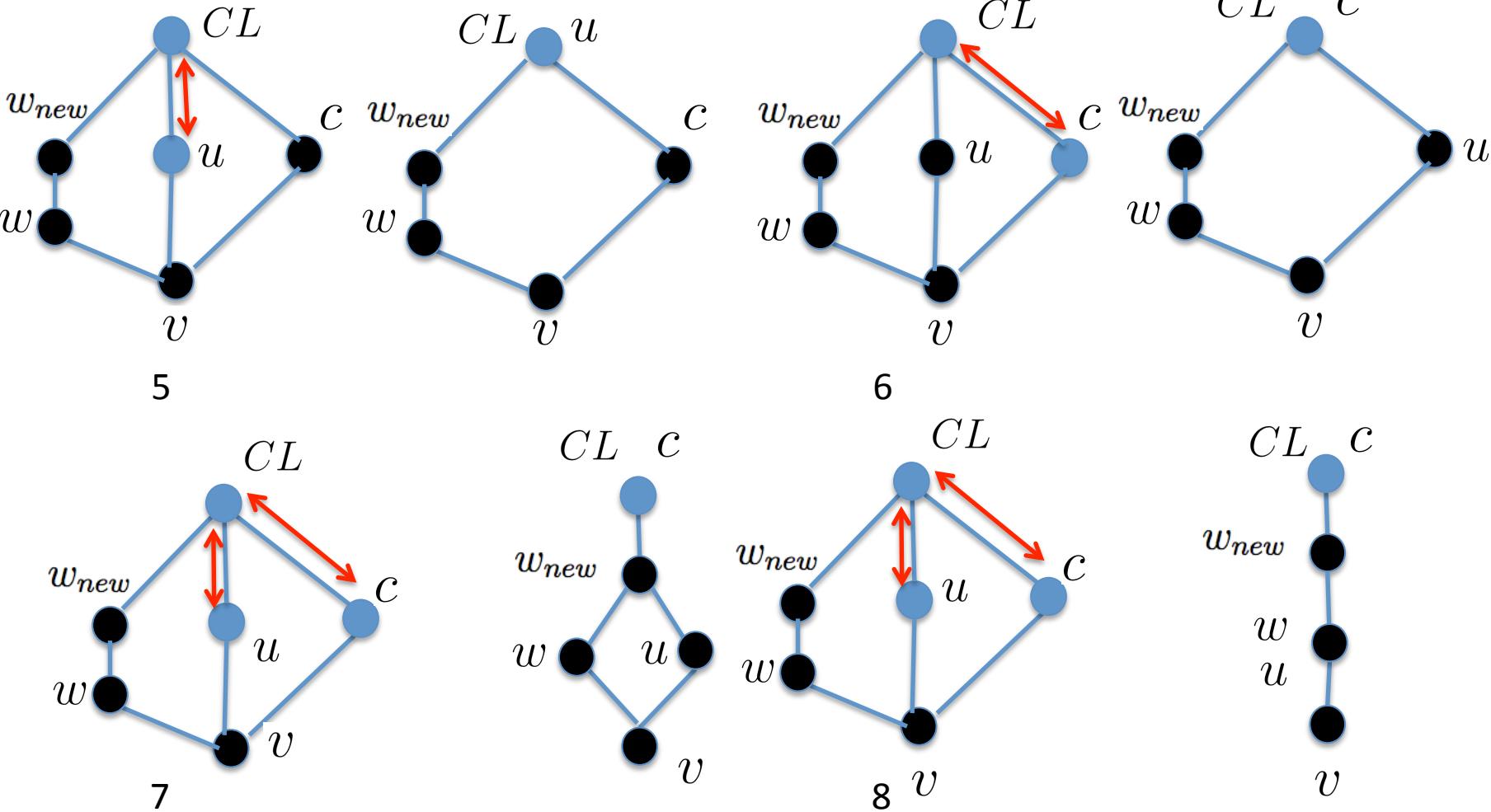
# Свойства реконструктора

- Частные случаи:



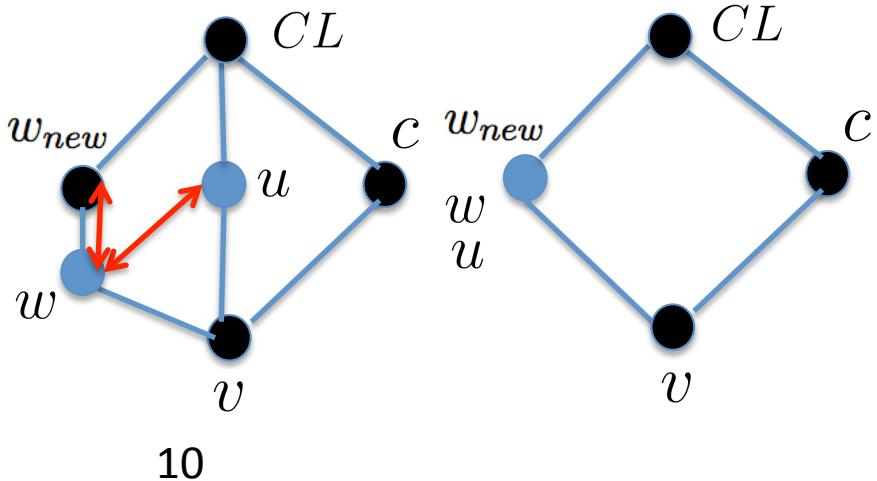
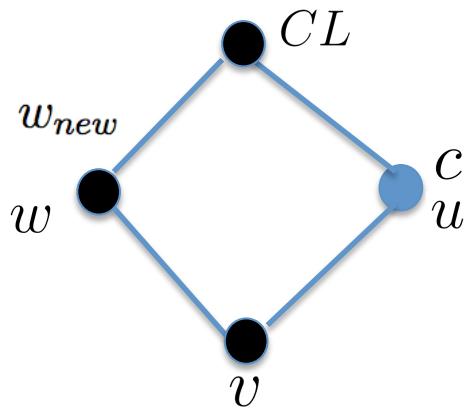
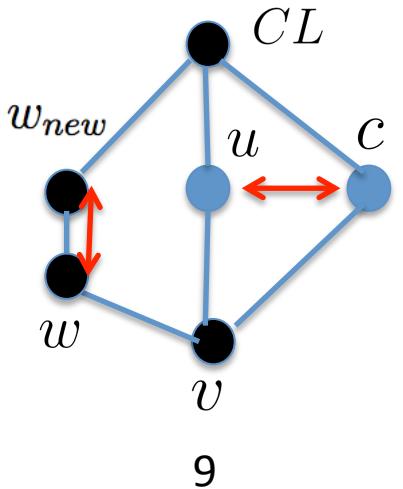
# Свойства реконструктора

- Частные случаи:

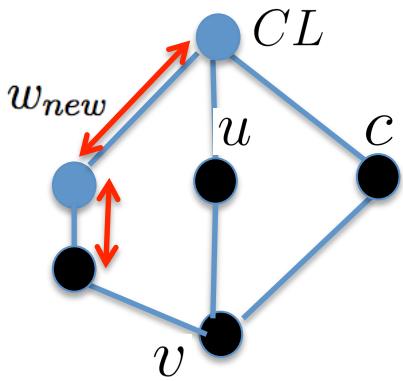


# Свойства реконструктора

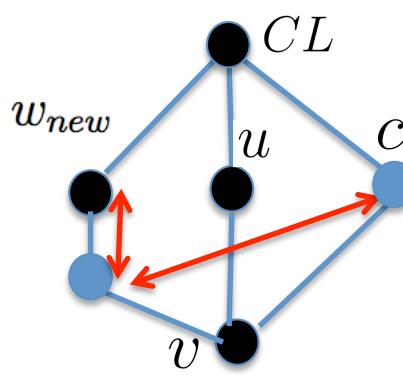
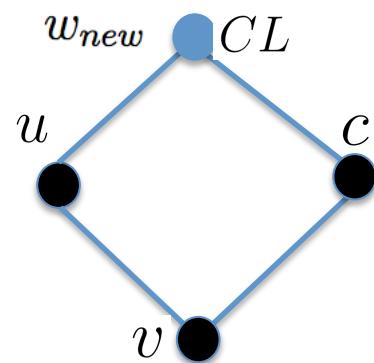
- Частные случаи:



10



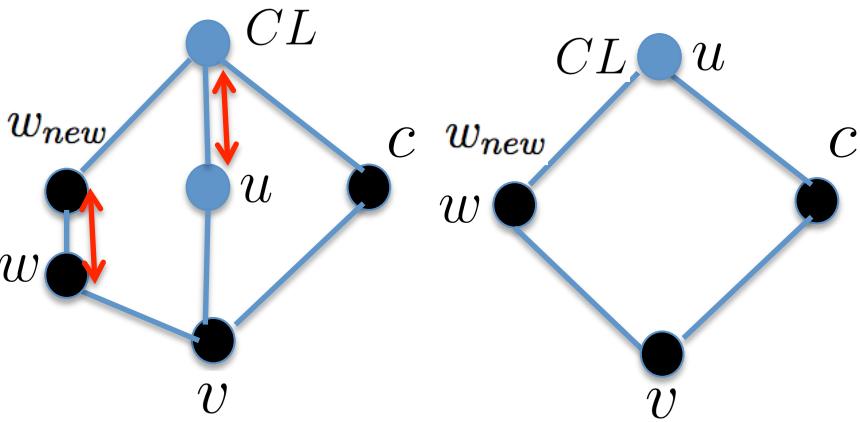
11



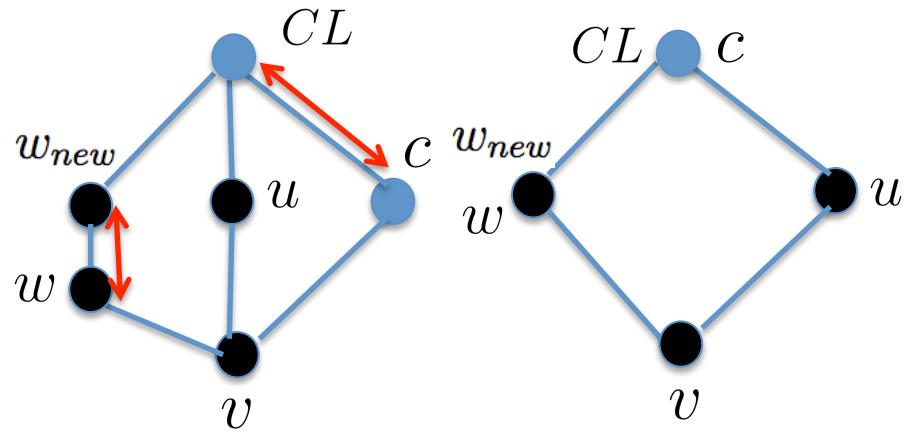
12

# Свойства реконструктора

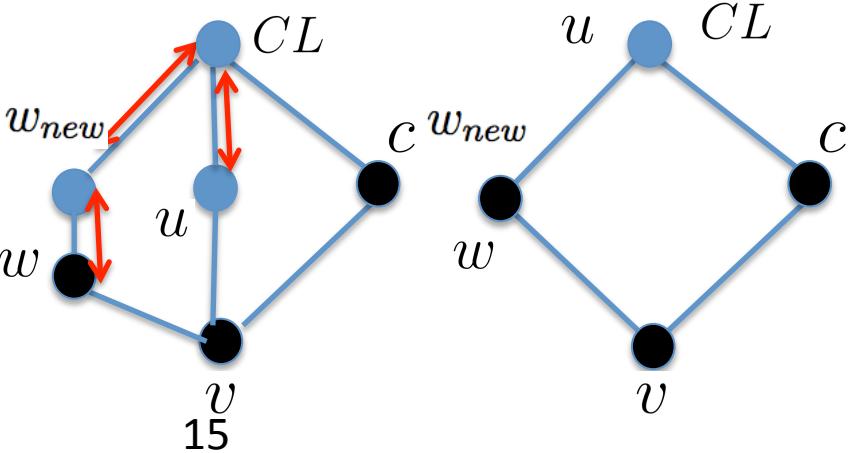
- Частные случаи:



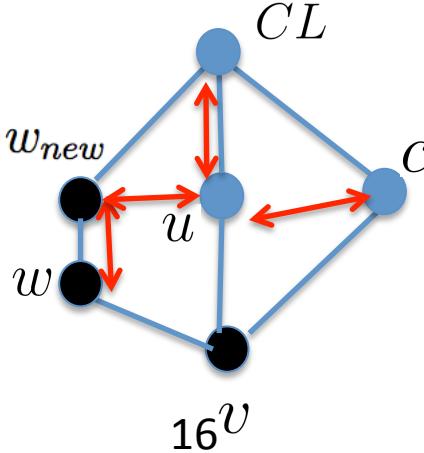
13



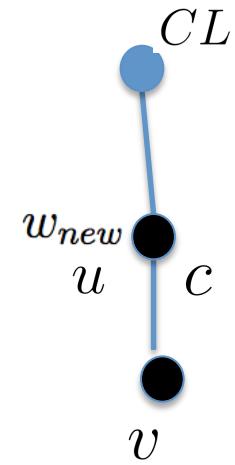
14



15



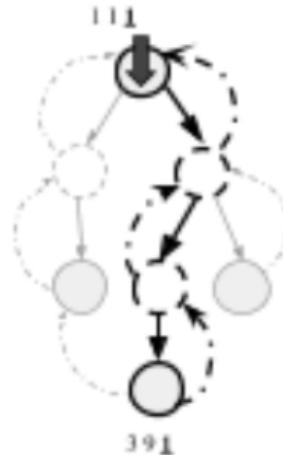
16



# Типы передачи атрибутов

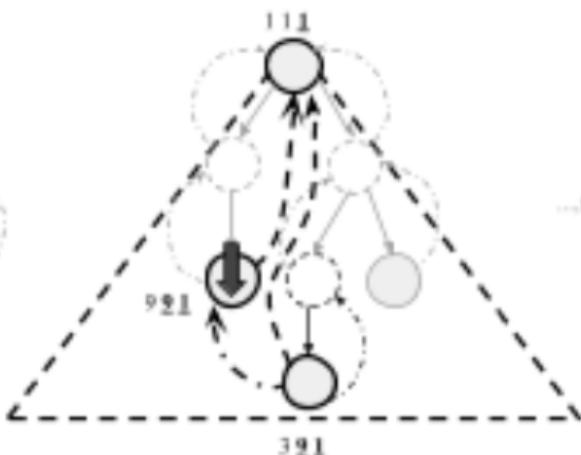
Тип I

8, 16



Тип II

1-15



Тип III

7,8,16



Тип IV

1-16



Строго наследуемые    Наследуемые в поддереве    Переназначаемые    Свободные

- - "конечная" вершина
- - "промежуточная" вершина
- ⬇ - точка создания ресурса
- ↗ - иерархические связи и системные вызовы
- ↖ - зависимости по непосредственно предшествующим вершинам
- ↗ - зависимости от дополнительных атрибутов

# В результате

- Задача выставления атрибута процессу сведена к встраиванию генератора как графа в граф полурешётки  $V^+$ .
- Следовательно, предложенный механизм позволяет не только детектировать аномалии, но и восстанавливать дерево.
- Построение формального алгоритма, реализующего такой механизм – открытый вопрос.

# Открытые вопросы

1. Рассмотреть решётку  $V^{++}$ , предложить эффективный способ дополнения  $V$  до  $V^{++}$ . Частные случаи? Когда дополнение Дедекинда-Макнила даёт подходящую решётку?
2. Построить эффективный алгоритм реконструкции (полином небольшой степени по времени, линейно по памяти)

# Выводы

- Предложен критерий корректности дерева
- Исследованы зависимости у подмножеств, восстанавливающих состояния процессов.
- Открытые вопросы – нужно продолжать исследование.

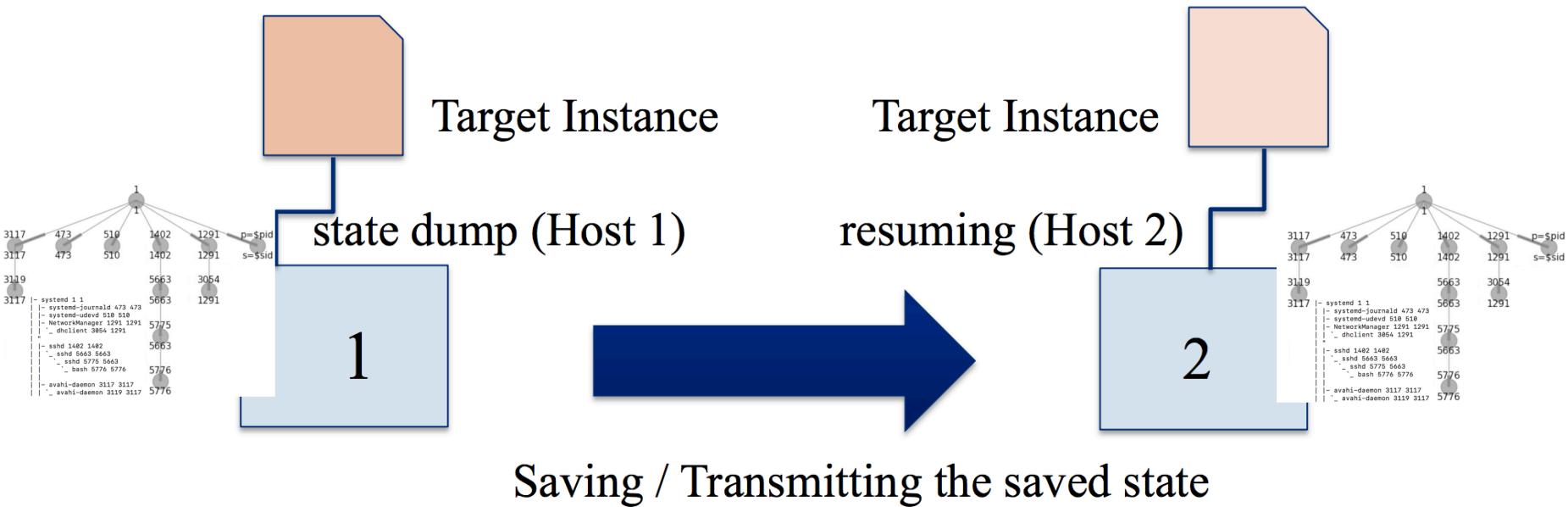
Благодарю за внимание!

Ефанов Николай, МФТИ  
[nefanov90@gmail.com](mailto:nefanov90@gmail.com)



# Цель анализа дерева процессов

- Быстрое сохранение/восстановление состояния
- Минимизация издержек при живой миграции
- Идеал: дерево процессов + контекст/данные (память, файлы, разделы ФС)



# Оценки числа деревьев

**Доказательство** методом математической индукции.

- 1)  $n = 2$ , тогда,  $F(2) = 2^0$ , что соответствует единственной конфигурации с процессами 0 и 1.

2)  $n = k - 1$ , тогда  $F(k - 1) = \sum_{i=2}^{k-2} i^{i-2}$ .

- 3)  $n = k$ . Для меньшего числа процессов все различимые конфигурации уже перечислены на предыдущем шаге. Осталось перечислить все возможные конфигурации из  $k$ -процессов, что можно сделать, согласно утверждению 1, по формуле

4)  $F(k) = \sum_{i=2}^{k-2} i^{i-2} + (k-1)^{(k-1)-2} = \sum_{i=2}^{k-1} i^{i-2}$ ,

что и требовалось доказать.

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

Данный результат в точности соответствует суммированию количества различных остовных деревьев на  $n$  вершинах, получаемых из формулы Кейли [2, 3, 4], с учётом одной выделенной вершины для корневого процесса.

# Оценки числа деревьев

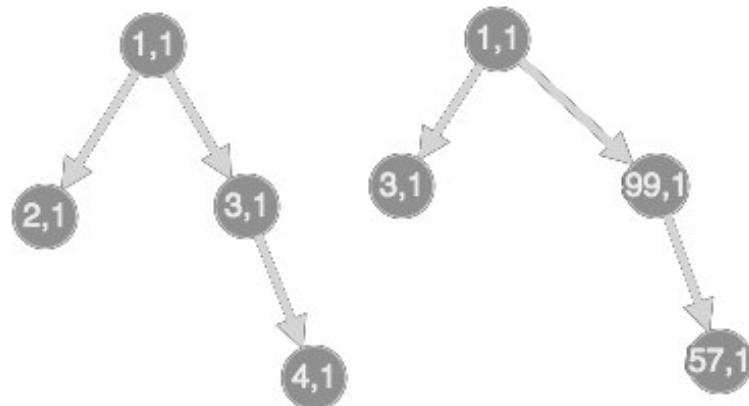
- Вызов fork: порождение потомка.

Оценка количества различных деревьев процессов, поддерживающих системный вызов `fork()` приводится в работе [2]:

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

Данный результат в точности соответствует суммированию количества различных остовных деревьев на  $n$  вершинах, получаемых из формулы Кейли [2, 3, 4], с учётом одной выделенной вершины для корневого процесса.

- С учётом различимости атрибутов:



- различные деревья, но!  
- идентичные состояния

# Оценки числа деревьев

УТВЕРЖДЕНИЕ 1. Количество генерируемых деревьев процессов без рассмотрения функциональной эквивалентности равняется:

$$F(n) = \sum_{i=2}^{n-1} \binom{2^{16} - 2}{(i-1)} (i-1)! i^{i-2} \quad (2)$$

ДОКАЗАТЕЛЬСТВО.

Зафиксируем число процессов  $N$ . Учёт одного лишь идентификатора  $P$  влечёт за собой функциональную эквивалентность деревьев, полученных из некоторого произвольного дерева перестановкой идентификаторов  $P$  в вершинах, без учёта корневой.  $P$  может изменяться от 2 до  $2^{16}$ , таким образом, выбор идентификаторов можно совершить  $(2^{16} - 2)!$  способами, а число способов назначить множество значений  $P$  ( $N-1$ ) – процессам равно числу размещений  $(N-1)$  в  $(2^{16} - 2)$ :

$$\binom{2^{16} - 2}{(N-1)} (N-1)!$$

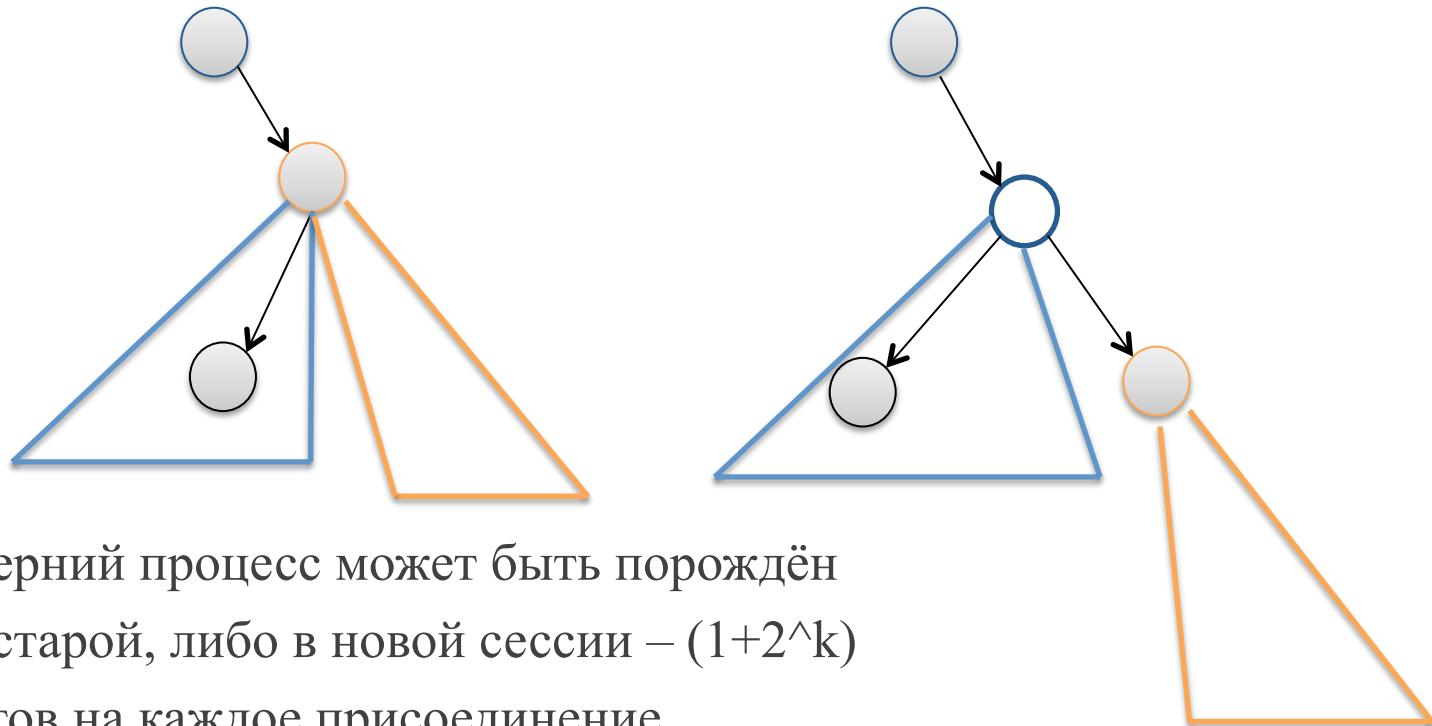
что в точности равно поправке из формулы (2) к формуле (1).

□

Таким образом, идентификаторы некорневых процессов образуют симметрическую группу  $S_{N-1}$ , что позволяет выполнять любые перестановки на множестве данных идентификаторов, а также применять алгоритмически эффективные структуры для их хранения [2, 3].

# Оценки числа деревьев

- Вызов setsid(): порождение новой сессии
- В нотации [2] : (\* \* [\*]) → (/1 /1 [/3])



- Дочерний процесс может быть порождён либо в старой, либо в новой сессии –  $(1+2^k)$  вариантов на каждое присоединение.

# Оценки числа деревьев

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

где  $\mathbf{f}(j)$  – вектор-функция размерности  $l$ -количества уровней дерева, возвращающая  $j$ -й бинарный вектор в порядке инкрементального перечисления,  $\mathbf{k}$  – вектор, составленный из суммарного количества потомков на уровнях 1 … 1 соответственно.

**Доказательство.** Каждый процесс, не являющийся лидером группы, может вызвать `setsid()`, причем только один раз. Порядок вызова для процесса с потомками также имеет значение: потомок может быть порождён как до образования новой сессии, так и после, то есть существует  $(1 + 2^{k_{children}})$ - вариантов для потомков процесса, причем каждый из них также может также сделать `setsid()`, добавляя  $2^{k_{children}}$  вариантов для каждой из конфигураций. Проводя обход в ширину из корня дерева, считающийся нулевым уровнем глубины, получим выражение для поправки:

$(1 + 2^{k_1}) \cdot 2^{k_1} \cdot \dots \cdot (1 + 2^{k_l}) \cdot 2^{k_l}$ , раскрывая скобки в котором, а также учитывая, что  $k_1 + \dots + k_l = i - 1$  для  $i$  процессов, получаем выражение  $2^{i-1} \cdot (1 + 2^{k_1} + \dots + 2^{k_1 k_2} + \dots + 2^{k_1 k_l} + \dots + 2^{k_1 \dots k_l})$ , что эквивалентно поправке из утверждения.

# Оценки числа деревьев

Рассмотрение идентификаторов сессии вносит поправку, также рассмотренную в работе [2]. Поправка на неразличимость функционально эквивалентных деревьев полученную ранее формулу как:

$$F(n) = \sum_{i=2}^{n-1} \binom{2^{16} - 2}{(i-1)} (i-1)! i^{i-2} 2^{i-1} \sum_{j=0}^{2^{l-1}} 2^{(\mathbf{f}(j)\cdot\mathbf{k})}, \quad (3)$$

где  $\mathbf{f}(j)$  – вектор-функция размерности  $l$ -количества уровней дерева, возвращающая  $j$ -й бинарный вектор в порядке инкрементального перечисления,  $\mathbf{k}$  – вектор, составленный из суммарного количества потомков на уровнях  $1 \dots l$  соответственно.

- Вывод: значительное количество различных деревьев при различности атрибутов
- Комбинаторный взрыв при переходе от fork к setsid:

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

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

- Следует искать способы решения
  - с учётом образования идентификаторами симметрической группы
  - не использовать генерацию, использовать анализ атрибутов дерева

# Подход: языки и грамматики

- Поиск грамматики в мягко-контекстно-зависимых формализмах (между 1 и 2 типом по Хомскому, типа 2 не хватает [2, 3]).
- Контекст – вида «состояние процесса» - «атрибуты состояний {любых} процессов».
- Грамматика:  $G = \langle T, N, P, \langle s \rangle \rangle$ 
  - Т – терминалы – конечные состояния (процессов)
  - Н – промежуточные состояния
  - Р – правила построения дерева + системных вызовов
  - $\langle s \rangle$  - стартовый нетерминал.
- Вход: дерево процессов D – слово в языке L, порождаемом данной грамматикой
- Выход – дерево разбора Т с метками на рёбрах – системными вызовами, приводящими к данному D

# Грамматика деревьев процессов

**Определение.** Грамматикой системных вызовов Linux является  $\{T, N, S, P\}$ , где

$T$  – строчные наборы атрибутов соответствующих состояний процессов

$N = \{\langle S \rangle, \langle branch \rangle, \langle node \rangle, \langle init \rangle, \langle mul \rangle, \varepsilon, \langle children \rangle\}$  –

$S$  – стартовый нетерминал. Здесь им полагаем входное дерево

$P$  – набор правил, приведённых в форме Бэкуса-Наура:  $O(n^3)$

$\langle S \rangle = \langle init \rangle \mid \langle init \rangle, \langle children \rangle$

$\langle children \rangle = \langle mul \rangle$

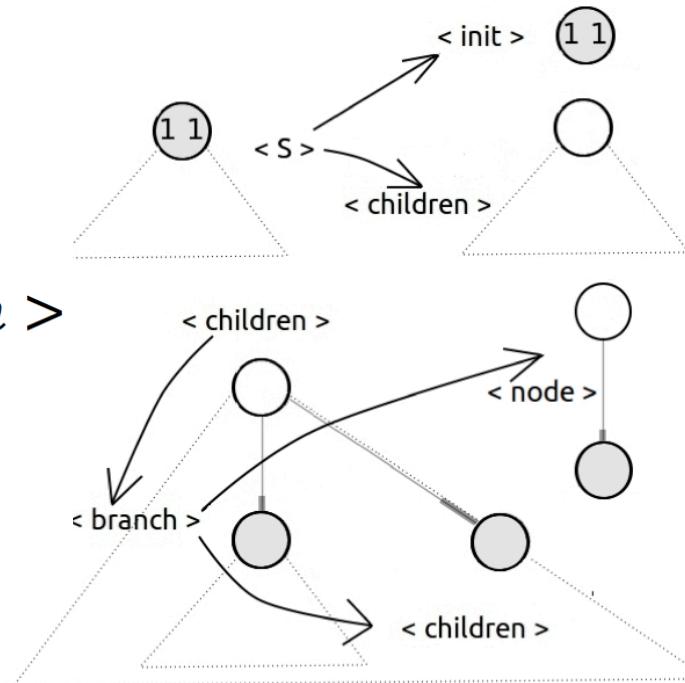
$\langle mul \rangle = \varepsilon \mid \langle mul \rangle, \langle branch \rangle$

$\langle branch \rangle = \langle node \rangle \mid \langle node \rangle, \langle children \rangle$

$\langle init \rangle = "(1..1)"$

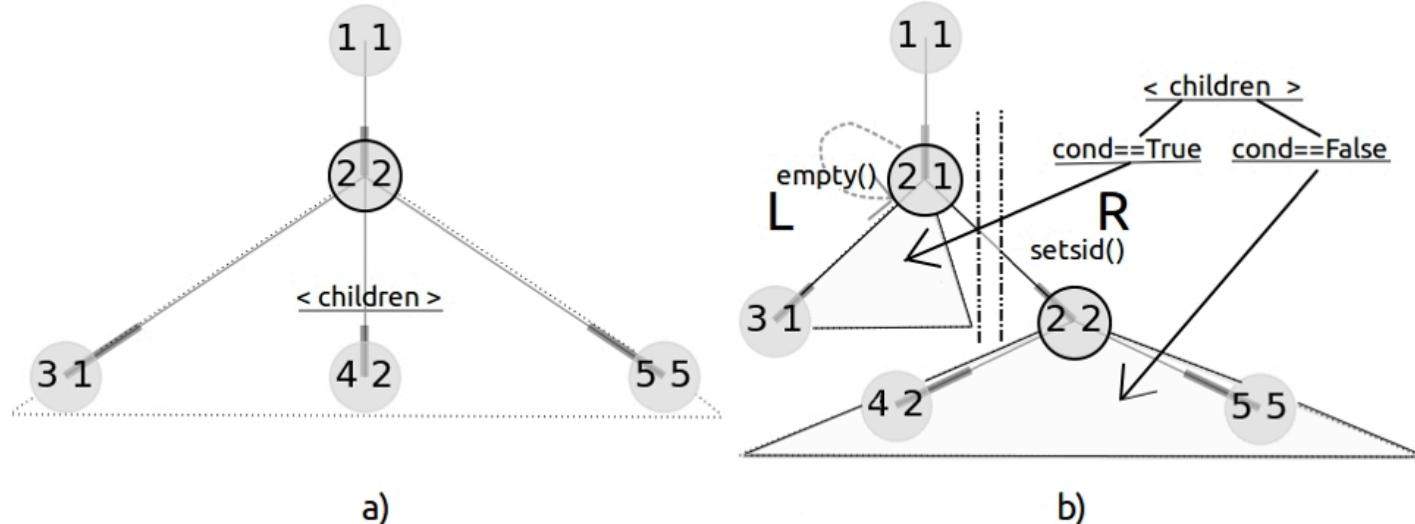
$\langle node \rangle = \langle node\_sc\_1 \rangle \mid \dots \mid \langle node\_sc\_n \rangle$

$F: \{\langle node \rangle, cond\} \rightarrow \{\langle node\_sc \rangle\}$

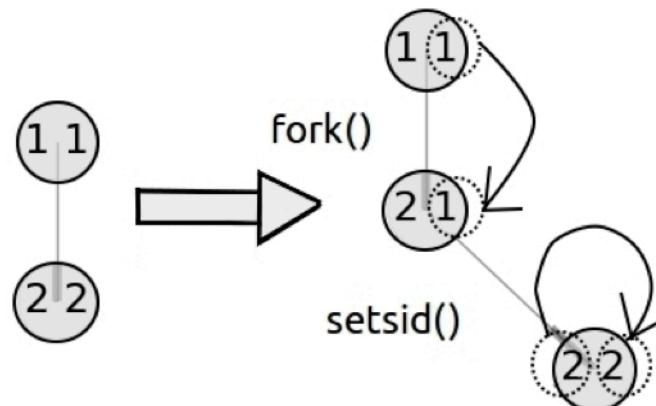


# Контекст: проверки атрибутов

- Наборы правил проверок в нетерминалах <node>:

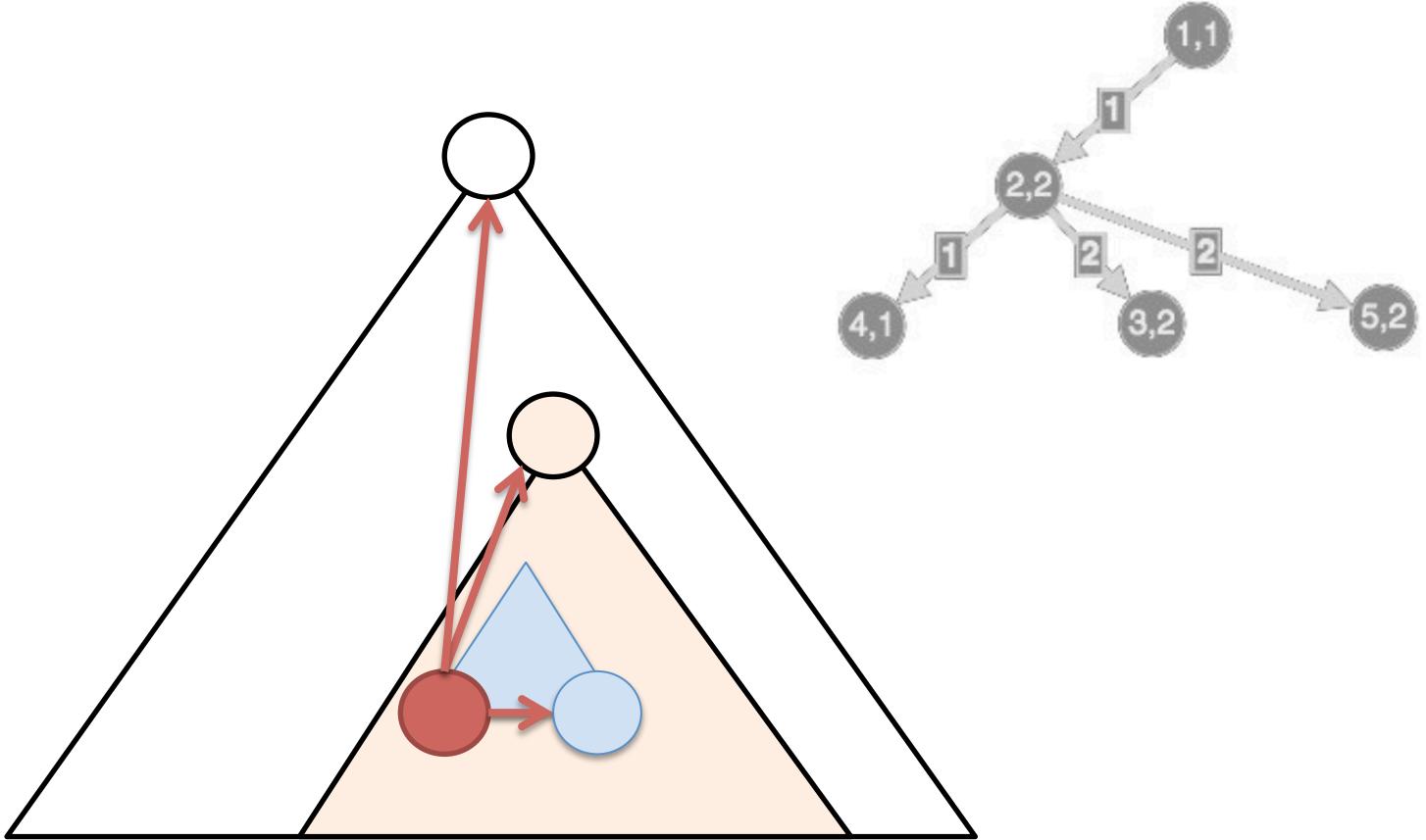


- Для большинства правил проверяемые атрибуты локализованы в одной сессии, группе и др. (наследование + неукорачивание).



# Сегментация дерева процессов

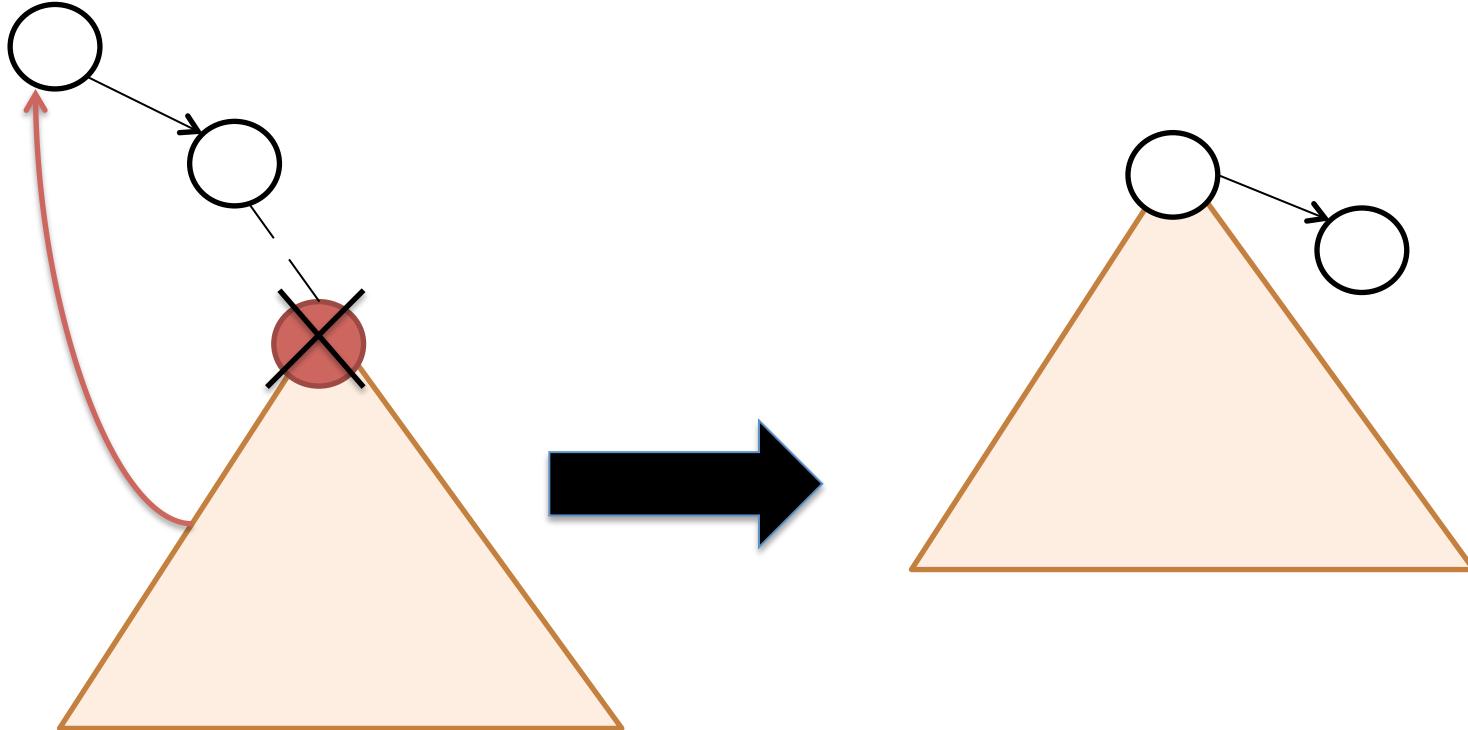
- Наследование + вложенность по поддеревьям



- Результат: для  $n$  –процессов нужно сделать  $k$  – проверок не сложнее, чем  $O(n^1)$ ,  $k \ll N \Rightarrow O(n^1)O(n^1) = \underline{O(n^2)}$

# Укорачивание

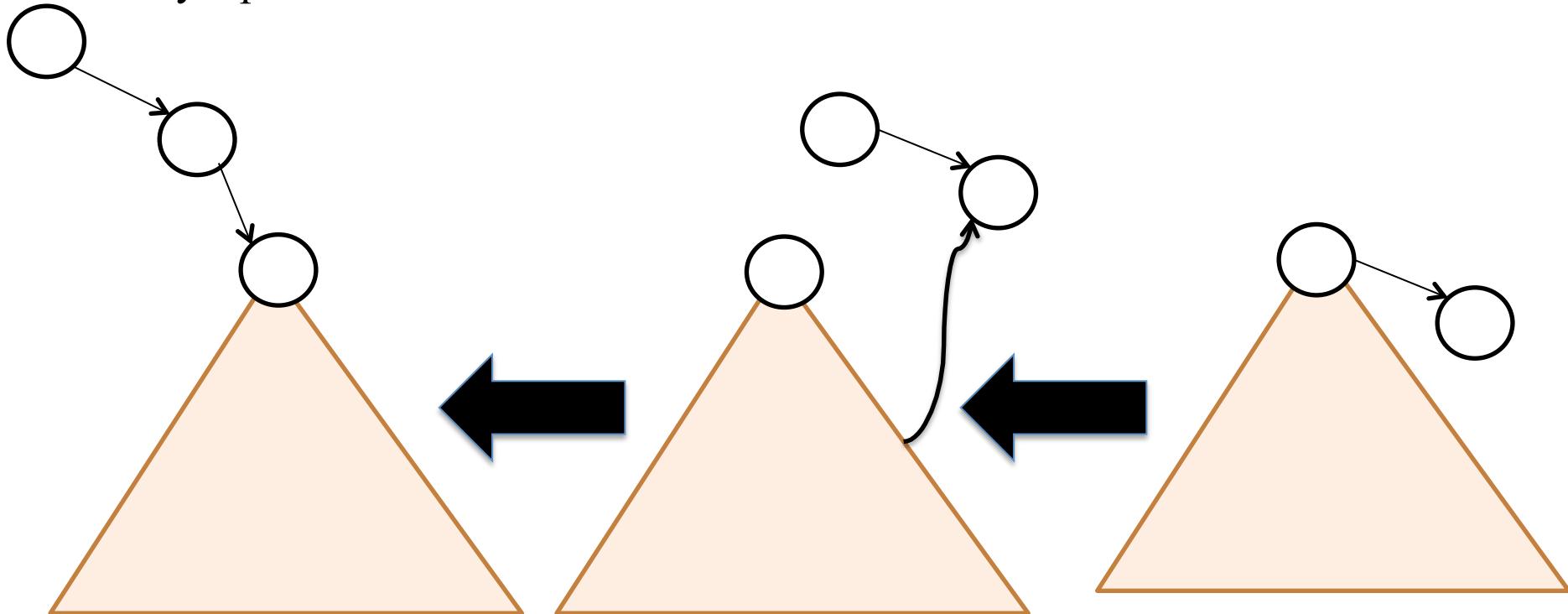
- Системный вызов exit(): завершение процесса с переупорядочиванием потомков 1-го уровня к корневому процессу.



- Проблема: грамматика Типа 0 – из дерева удаляется информация.

# Эвристический подход

- Поиск/восстановление нужного сегмента (сессии, пространства имён, любого атрибута, наследуемого сверху – см. setsid())
- Производится перед разбором в грамматике – грамматика может быть неукорачивающей.



# В заключении

- Отмеченные свойства дерева процессов с одной стороны, порождают практические трудности в практическом решении задачи восстановления цепочек системных вызовов: комбинаторный взрыв при добавлении новых типов вызовов, более слабая, но большая нелинейность роста количества деревьев при добавлении новых вершин
- С другой стороны, полученная симметрическая группа идентификаторов позволяет эффективно анализировать эквивалентные деревья процессов, например, нормализацией идентификаторов, введением хеш-функций и др.
- Ограничения на вид проверок атрибутов, а также большое число их наследований позволяют полиномиальные алгоритмы разбора таких деревьев:  $O(n^5)$ .
- Локализация атрибутов и относительно небольшое их число приводят к практически меньшему числу проверок, нежели в теоретической оценке.