# Использование capability-based security в ядрах операционных систем

Басков Евгений

Москва
2025

ИСП РАН

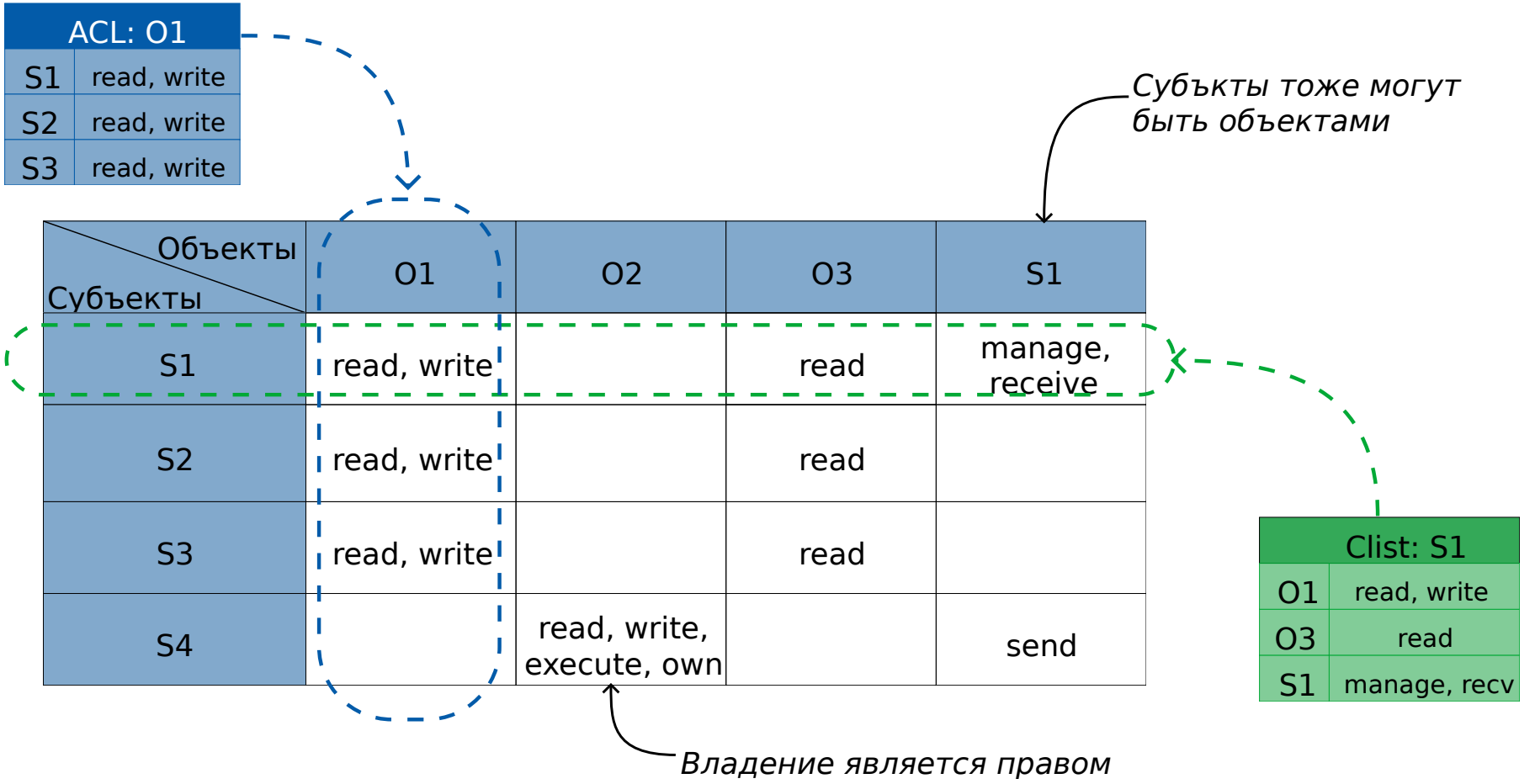
# Контроль доступа

Отвечает на вопрос: **кто** может делать **что** с **чем**?

- **Кто** — субъект.
  - Обычно является процессом.
- С **чем** — объект.
  - Файл, директория, другой процесс, сокет и т.д.
- **Что** — право доступа.
  - Read, Write, Execute, Delete, Signal, ...
  - Зависит от типа объекта.
- Должен быть корень доверия, контролирующий защиту.
  - *Например, монитор ссылок или ядро ОС.*
- DAC (Discretionary Access Control): владельцы объектов контролируют права.
- MAC (Mandatory Access Control): внешняя политика контролирует права доступа.

Рассмотрим состояние системы в какой-либо фиксированный момент времени...

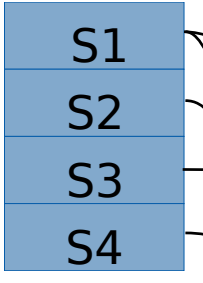
# Access Control Matrix



# UNIX ACLs

Матрица для каждой пары слишком объемна и субъекты динамически изменяются.

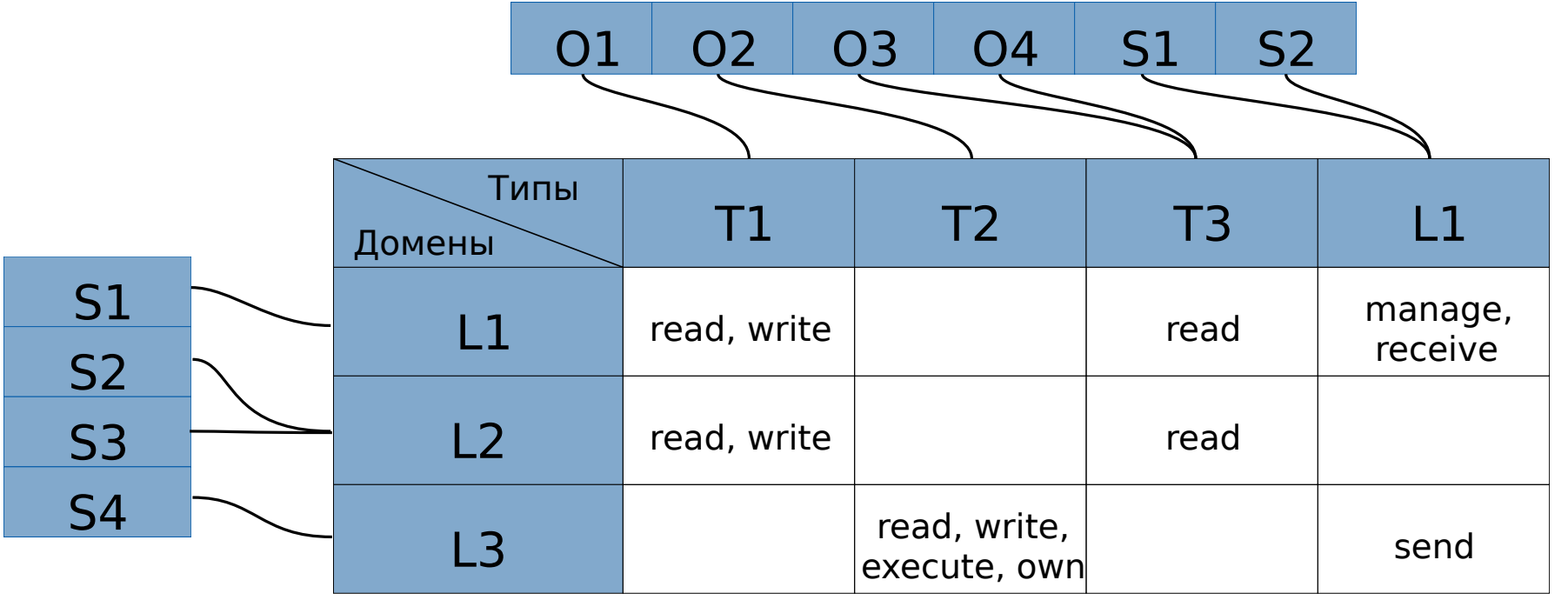
- Добавить метки (UID, GID) для субъектов.
- Задавать права для меток.
- Могут действовать отрицательные разрешения.
  - Используется наиболее специфичная метка.
- Есть неявные права на модификацию разрешений.




| Объекты / Метки | O1 | O2 | O3 | S1 |
|---|---|---|---|---|
| L1 | | | | manage, receive |
| L2 | read, write | | read | |
| L3 | | read, write, execute, own | | send |

# SELinux

- Для большей гибкости также добавить метки к объектам.
- Хранить права отдельно.



# Capability-based security

- **(Object) capability** или **жетон** — неподделываемый локальный идентификатор объекта ядра с прикрепленными к нему правами доступа и возможностью передачи через IPC.
- Владение жетоном отождествляется с правом доступа.

- **Capability list (C-list)** — локальное для субъекта пространство имен.
- Доступ предоставляется по имени жетона внутри этого пространства имен.

- *Например: Capsicum, seL4, EROS, Mach, Genode, UNIX file descriptors.*
- *Linux capabilities **не являются** objects capabilities, поскольку права, предоставляемые ими, не зависят от объекта.*

# ACL vs Capabilities: Имена

Access Control List (ACL):

- Требуют глобального пространства имен.
- Получают доступ только по имени объекта.
- Используют неявные права доступа (Ambient authority).
- **Confused Deputy problem:** *Если имя объекта было получено извне, система может получить доступ к неправильному объекту.*

Capability-based security:

- Только локальные имена.
- Права прикреплены к имени объекта.
- Нет неявных прав доступа.

# ACL vs Capabilities: Изменение прав

Access Control List (ACL):

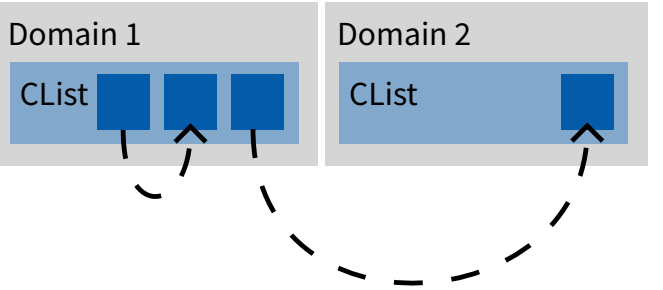
- Модификация ACL является неявным правом владельца или монитора.
- Изменение владельца — привилегированная операция.
- Дочерние процессы наследуют все права доступа.
  - *Нарушает принцип минимальных привилегий.*

Capability-based security:

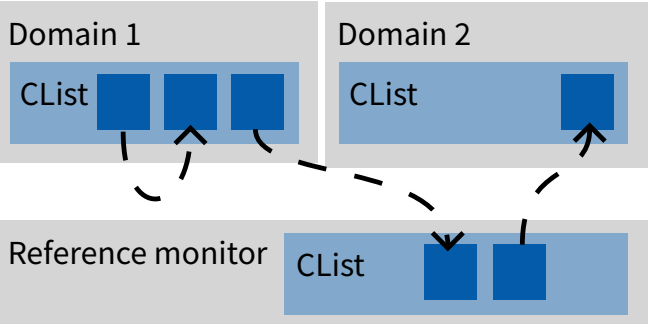
- Изменение владельца передачей жетона через IPC и отзывом (*revocation*) жетона.
  - Передача возможна только субъекту, жетоном которого владеет текущий субъект.
  - У передаваемого жетона должно быть установлено право *grant*.
  - У жетона получателя должно быть установлено право *delegate*.
- Дочерние процессы наследуют только указанные жетоны.
  - Для **UNIX fd** неверное поведение: нет `O_CLOEXEC` по умолчанию.

# Интерпозиция и MAC для жетонов

MAC реализуется посредством интерпозиции (прозрачной подмены) примитивов IPC между доменами защиты монитором ссылок и флагов доступа:




- Жетоны, через которые можно передавать данные между субъектами заменяются на промежуточные порты монитора ссылок.
- Для каждого передаваемого жетона монитор может:
  - заменить жетон;
  - понизить права;
  - запретить передачу.




- Интерпозиция также полезна для добавления дополнительной семантики,
  - *Например, ведения журнала, отладки, или ленивого создания объектов.*

# Структура C-list

**Capability list** или **таблица жетонов** — локальное для каждого домена защиты или потока пространство имен, используемое для адресации жетонов.

- Имя или путь жетона — адрес в этом пространстве имен, обычно является числовым.
- Имена могут выделяться ядром или пользователем.
- Пространство имен может иметь разную реализацию:
  - Непрерывный массив
  - Хеш-таблица
  - Дерево отрезков
  - Guarded page table

# State of the art

Среди рассмотренных реализаций актуальными являются:

- **seL4** — формально верифицированное микроядро третьего поколения.
- **EROS** — персистентная микроядерная система высокой надежности.
- **Mach** — микроядро первого поколения.
- **Capsicum** — фреймворк capability-based security для UNIX-like систем.
  - *Изначально для FreeBSD*.
- **Genode** — фреймворк для реализации компонентой ОС общего назначения поверх ряда микроядер.
- **Fuchsia** — микроядерная ОС общего назначения от Google, планировавшаяся как замена Android.

# Ограниченные жетоны

Есть два противоречащих друг другу аспекта реализации: ограниченные жетоны и уникальные имена.

- Уникальные имена/Нет ограниченных жетонов:
  - Позволяют сравнивать жетоны как числовые значения.
  - Требуют счетчиков ссылок в пространстве пользователя либо в таблице жетонов.
  - Требуют выделения имен ядром.
    - *Это усложняет реализацию и фиксирует политику.*
  - *Например: Mach, Genode*
- Дублирующиеся имена/Ограниченные жетоны:
  - Используют расширенные флаги с правами доступа.
  - Позволяют настроить возможность выполнения групп операций над объектом.
  - Требуют возможности наличия нескольких экземпляров жетона.
  - *Например: Capsicum, seL4, EROS*

# Общий интерфейс системных вызовов

Многие реализации используют общий интерфейс для объектов IPC и других объектов ядра.

- Позволяет производить интерпозицию всех объектов ядра.
- В теории может ускорить IPC.
- Операции над объектами становятся вызовами IPC.
  - Несмотря на то, что документация не называет их таковыми, также являются системными вызовами.
  - *Например, seL4 имеет около 80 методов, реализованных через IPC, но только 7 (10 для MCS) настоящих системных вызовов.*
- *Например, seL4, Mach, EROS*

# Проблемы ранних реализаций

- **Отсутствие контроля распространения**: если жетон был передан другому домену защиты, он потенциально может быть передан любому домену напрямую или косвенно достижим средствами IPC через него.
  - *Важно при наличии уязвимых или менее доверенных доменов.*

- **Невозможность возврата доступа**: если жетон был передан другому домену, невозможно аннулировать доступ к нему.
  - *Требуется перед передачей более привилегированному домену ресурса, которым раннее владел менее привилегированный.*

# Контроль распространения

- Право `delegate` для возможности передачи жетона кому либо.
  - *Например, Fuchsia `ZX_RIGHT_TRANSFER`.*
- Право `copy` для возможности создания копий.
  - *Например, Fuchsia `ZX_RIGHT_DUPLICATE`.*
- Право `grant` для возможности передачи посредством данного жетона.
  - *Например, seL4.*
- *Weak capabilities.*
  - Все преданные через них жетоны становятся read-only и weak.
  - *Например, EROS*

# Возврат доступа

- Сильное владение.
  - Все жетоны удаляются при удалении объекта владельцем.
  - *Например, Genode, Mach, Fuchsia*.
- Отзыв (revocation) ссылок.
  - Все жетоны, производные от данного, удаляются при вызове операции **revoke**.
  - *Например, EROS, seL4*.
  - EROS: Через счетчики версий и прокси-объекты.
  - seL4: Через поддержку дерева производных жетонов.
  - *Использует модель* ***take-grant***.

# EROS

Использует глобальную таблицу объектов, хранящую указатели на объекты и номера их версий, а также 16-байтные метаданные жетонов, хранящие флаги доступа, указатель на таблицу объектов, номер версии, тип.

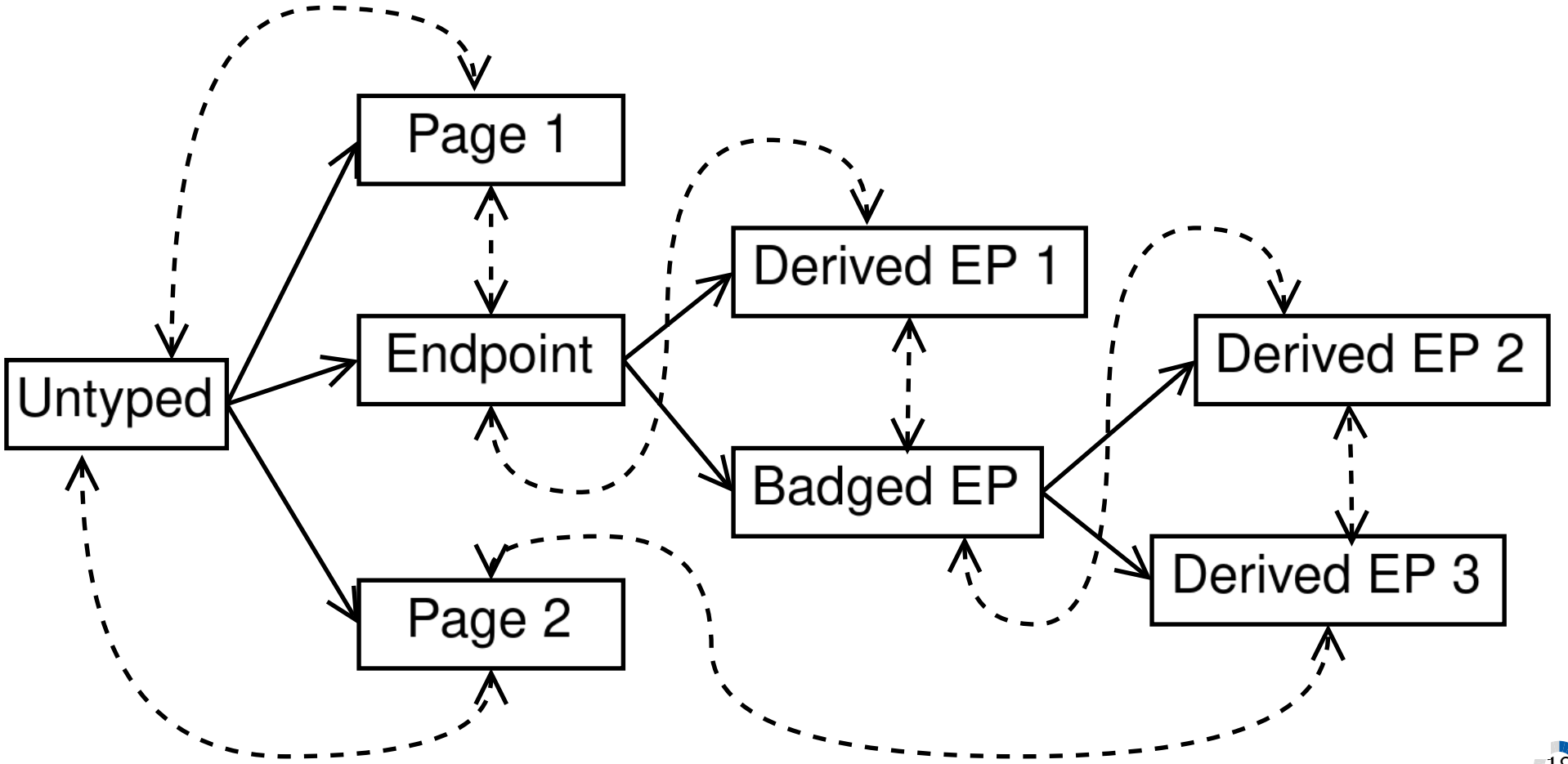
- Если версия в жетоне не совпадает с версией в таблице объектов, указатель не валидный.
- Решает проблему с загрузкой жетонов с диска для персистентности.
- Позволяет отозвать все ссылки путем увеличения номера версии ($O(1)$).
- Proxy-объекты прозрачно предоставляют доступ к родителю.
  - Позволяет отозвать только часть ссылок.

# seL4 (1)

Поддерживает дерево наследования производных жетонов (Capability Derivation Tree), представленное в виде двусвязного списка, элементами которого являются жетоны с явным указанием глубины в дереве.


- Размер данных жетона — 32 байта. Жетон содержит:
  - Элементы списка (указатель на следующий и предыдущий элемент).
  - Данные, специфичные для жетона (включая права доступа).
- Глубина дерева ограничена архитектурой.
- Позволяет отозвать все дочерние жетоны за $O(n)$.
- Поддерживает тегированные proxy-объекты.

# seL4 (2)


Page 1
Derived EP 1
Derived EP 2
Untyped
Endpoint
Badged EP
Derived EP 3
Page 2

# Требования к реализации

- Параллельное выполнение независимых операций.
- Быстрый поиск.
  - Чтения без захвата блокировок.
- Реализация не должна фиксировать политику выбора имен и отзыва.
- Пространство имен должно иметь переменный размер.
- Возможность отзыва отдельных жетонов.
- Поддержка ограниченных жетонов.
  - Без уникальных имен и с флагами доступа.
  - Поддержка ограничения передачи через порты (`grant` для IPC).
- Возможность безопасно разделять части пространства имен между доменами.

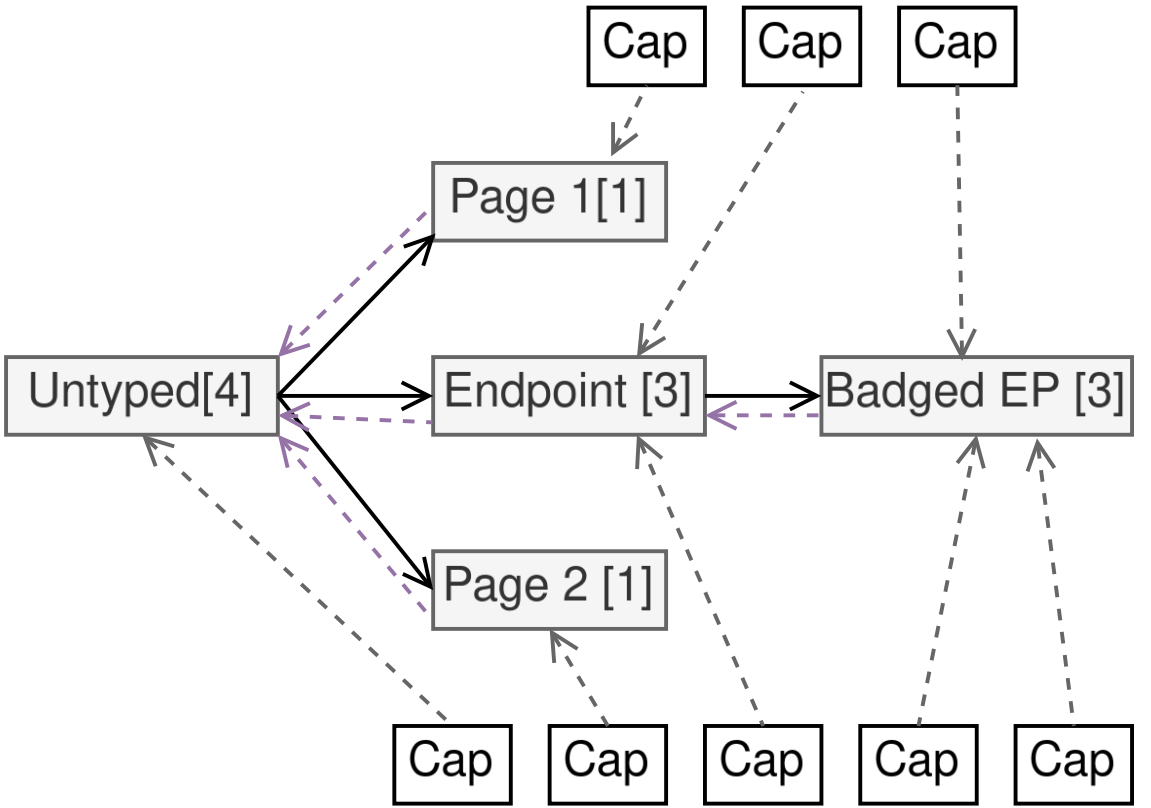
# Проблемы с реализациями EROS и seL4

- Плохая масштабируемость.
  - Big kernel lock.
  - Метаданные жетона больше машинного слова.
    - Невозможно реализовать чтение без блокировок в многопоточном ядре.
  - Использование списка в seL4.
    - Невозможно реализовать удаление и перемещение многопоточно.
- Предсказуемость реализации.
  - Использование списка в seL4 — отзыв за $O(n)$ от количества ссылок.
  - Возможность отзыва используемых объектов.
- Использование глобальной таблицы промежуточных объектов в EROS.
  - Большие требования памяти или ограниченное количество объектов.
  - Сложности с локальностью памяти.

# Реализация для микроядра общего назначения

- Объект содержит указатель на родителя и счетчик ссылок, таблица жетонов содержит только указатель *и права*.
- Объекты удаляются через RCU.
- Каждое имя и производный объект добавляет ссылку.
- Отзыв (revocation) возможен за счет прокси-объектов.
  - Отзыв требуется только для конкретных объектов (порты IPC или память).
  - Имеют тег для различия между отправителями в IPC.
  - Небольшие, 24 байта на прокси.
  - Отзыв путем удаления ссылки на родителя если есть право владения (`KPROT_OWN`).
  - Ленивое удаление ссылок при обращении.

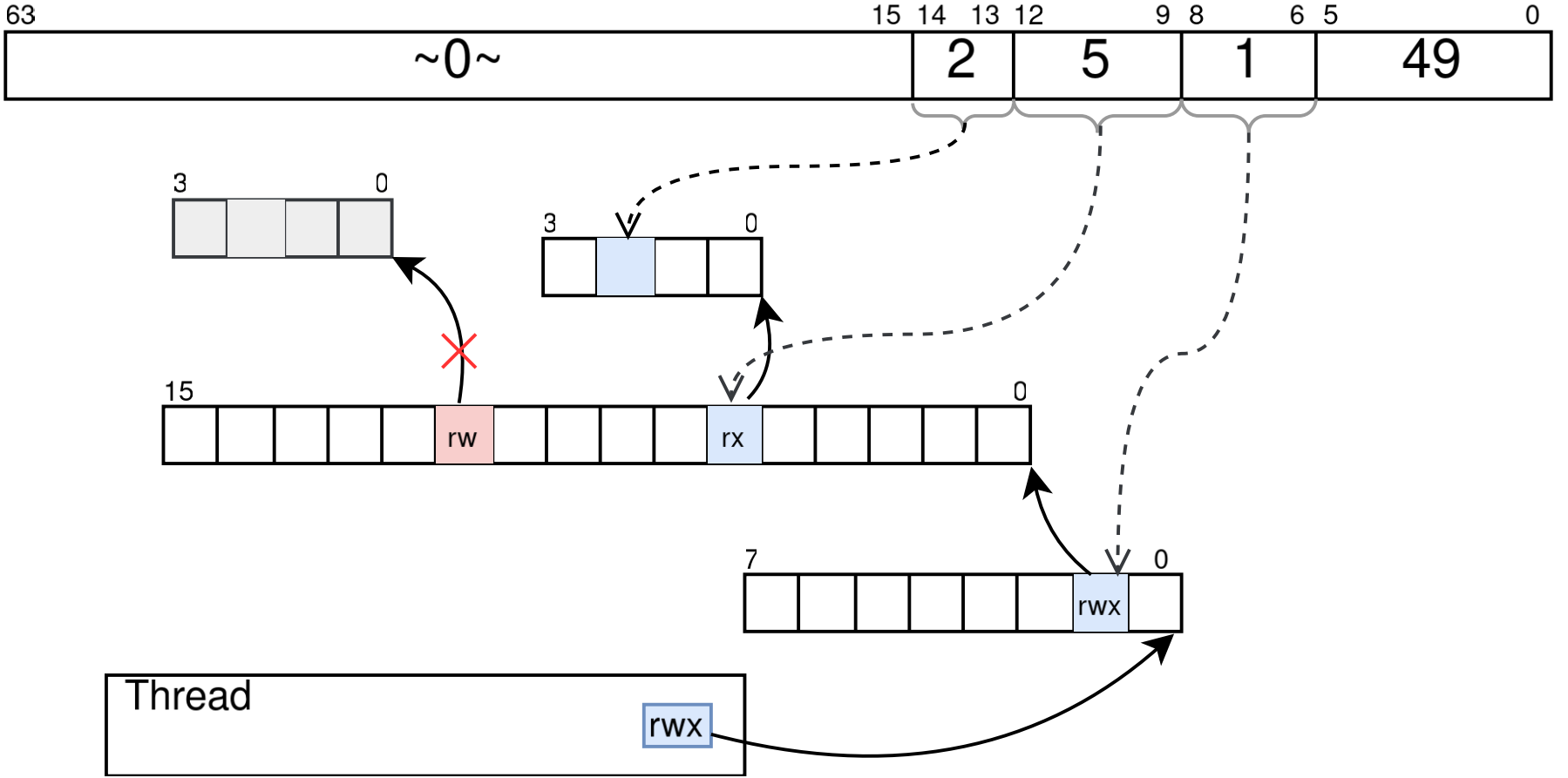
# Реализация для микроядра общего назначения



# Пространство имен

- Имена являются битовыми строками с присоединенной длиной.
- Длина кодируется как $W - \log_2(W) - L$
  - $L$ — длина машинного слова, $W$ — длина пути.
  - 0 – не валидный путь.
- Используется управляемое пользователем дерево отрезков.
- Узлы размера степени 2.
- Узлы являются объектами первого класса и выделяются пользователем.
- Права на таблице используются для поиска:
  - `KPROT_EXECUTE` — адресовать слоты.
  - `KPROT_READ` — читать значения слотов.
  - `KPROT_WRITE` — писать значения слотов.
  - `KPROT_TABLE_MOVE` — перемещать жетон из слотов.
- Путь конкретного слота является логическим И прав всех промежуточных таблиц, пройденных при поиске.

# Пространство имен



# Плюсы выбранной реализации

- Теоретически неограниченная масштабируемость и хорошая производительность.
  - Независимые операции не взаимодействуют.
  - Чтение без блокировок и барьеров памяти с помощью RCU.
  - Не требуются блокировки при копировании/перемещении.
  - Инкрементальная параллельная финализация объектов.
- Меньшее потребление ресурсов.
  - $16 + 8 * N$ байтов vs $32 * N$ байта в seL4 и $16 + 16 * N$ байтов EROS.
  - $N$ — количество жетонов-ссылок.
- Лучшая предсказумемость и более простая реализация.
  - Не требуется выделение памяти при копировании.
  - Только увеличение количества ссылок и модификация прав.
  - Отзыв и удаление за O(1). *Аналогично EROS*
  - Нет вероятности асинхронного удаления объекта в критическом коде.
    - *Пример: невозможно удалить домен во время выполнения потока.*

# Сложности в процессе реализации

- Циклические ссылки возможны, но только между потоками.
  - Для таблиц циклы запрещены с помощью строгой иерархии таблиц.
  - Для потоков вводятся итераторы групп потоков (thread sets), чтобы всегда иметь возможность получить ссылки на потерянные потоки.
- Потребовалась эффективная реализация RCU и инкрементального удаления.
  - Используются гибридные неблокирующие очереди запросов.
  - Используется иерархический RCU.

# Спасибо за внимание

Исследование и классификация подходов к управлению доступом проводилось в рамках совместных работ ИСП РАН и Лаборатории Касперского по направлению конструктивной информационной безопасности.

# Плоский массив

- Простая реализация.
- Возможен неблокирующий поиск с RCU.
- Возможно изменение размера с двумя таблицами и инкрементальным копированием.
- Требует выделения непрерывной памяти больших размеров.
  - Альтернативно: virtual mapping в ядре.
- Нет возможности разделять части пространства имен.
- *Например, таблицы fd в UNIX*

# Kernel-managed radix tree

- Дерево отрезков с фиксированным размером уровня.
- Размер уровня — степень 2.
- Возможен неблокирующий поиск с RCU.
- Нет проблем с изменением размера.
- Нет возможности контролировать выделение памяти пользователю.
- Нет возможности разделять части пространства имен.
- *Например, Mach*

# User-managed radix tree

*Аналогично предыдущему, но:*

- Узлы таблицы являются жетонами.
- Переменный размер уровня.
- Управляется пользователем.
- Длина ключа указывается явно как часть имени.
- *Текущая реализация.*

# Guarded page tables

*Аналогично предыдущему, но:*

- В каждом уровне есть guard
  - Некоторая битовая строка, которая сопоставляется с ключом.
- Позволяет иметь произвольную структуру пространства имен при небольшом количестве уровней.
- Минимум политик в ядре.
- Сложно перемещать сами таблицы — нужно указывать глубину.
- *Например, seL4, Coyotos*

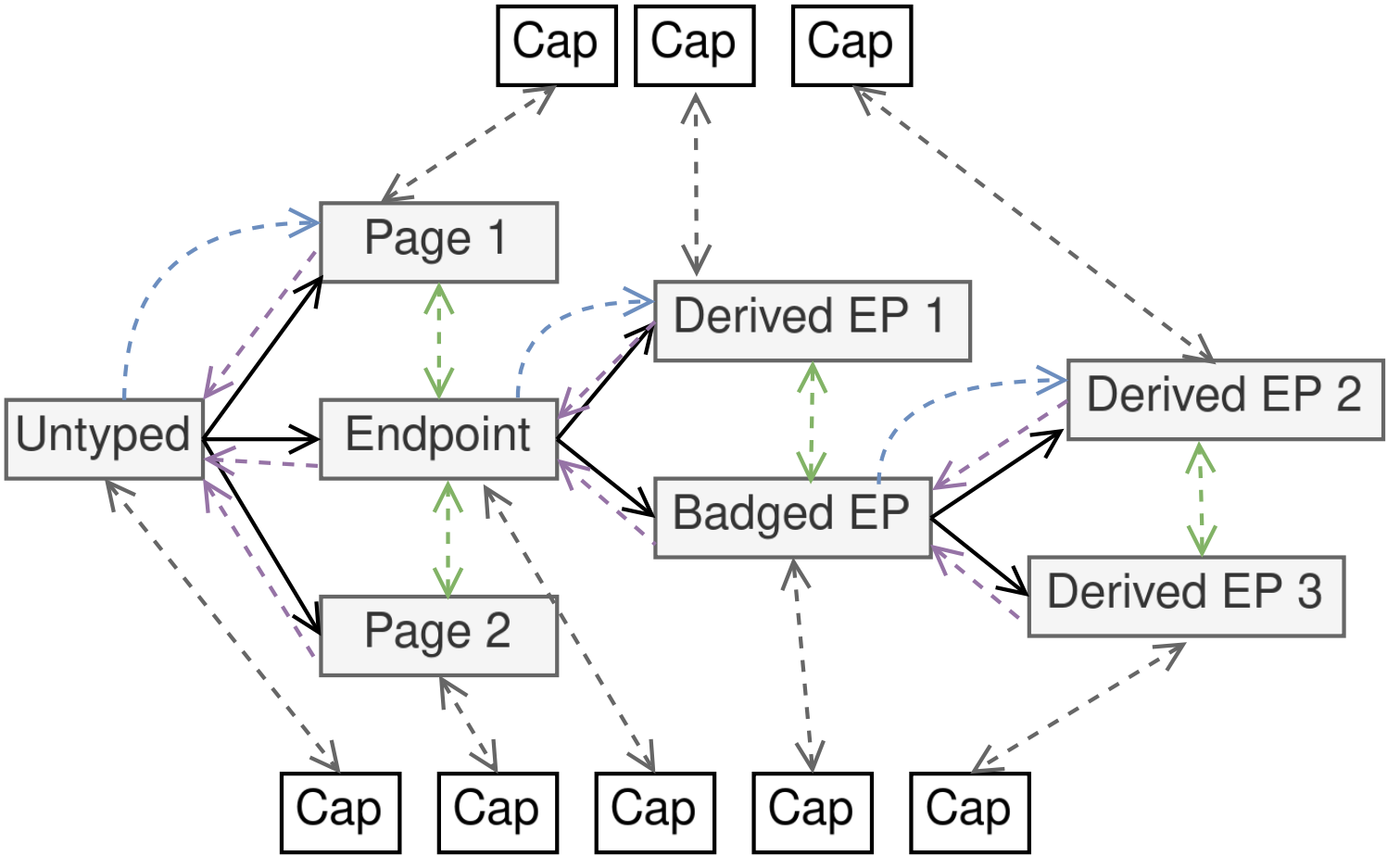
# Hash tables

- Позволяет иметь сильно разреженное пространство имен.
- Больше подходит для уникальных пространств имен.
- Возможно использование HAMT или Ctrie для получения преимуществ radix tree.
- Сложнее предыдущих.
- Требует реализации concurrent hash table.
- Невозможно контролировать локальность ссылок пользователем.
- Нет возможности контролировать выделение памяти пользователю.
- Нет возможности разделять части пространства имен.
- *Например, Managarm*

# Первый вариант

- Дерево производных, как в seL4.
- Более полное дерево.
  - Ссылки на родителя, предыдущего и следующего ребенка текущего родителя, ссылка на первого ребенка.
  - Позволяет блокировать только родителя, а не всю иерархию при модификации.
- Вся информация в отдельном блоке памяти, слот жетона содержит только указатель.
  - И тип для оптимизации.
  - Сам объект ссылается на слот жетона для поддержки отзыва.
  - Только одно имя для каждого объекта.
  - Требуется выделять отдельный объект для каждого имени.
  - RCU-поиск по дереву.
  - Удаление всех объектов через RCU.

# Первый вариант реализации



# Первый вариант (удаление)

- При удалении родителя удаляются все производные.
- Удаление в 3 прохода:
  - Финализация.
  - Ожидание RCU.
  - Освобождение памяти.
- Первый и третий:
  - используют одну очередь;
  - выполняются параллельно друг с другом;
  - выполняются параллельно и между собой для разных подграфов.
- Удаление инкрементальное.
  - Каждый проход проверяет наличие прерываний.
  - Состояние явно сохраняется и выполнение приостанавливается.
- После финализации поддерево удаляется из графа.

# Первый вариант (удаление)

Удаление всей иерархии реализуется через набор lock-free очередей.

- Очередь RCU с двумя поколениями.
- Lock-free MPMC очередь, содержащая непустые MPSC очереди.
- Одна lock-free MPSC очередь на процессор.
- Содержит поддеревья объектов.
- Используется RCU-обход поддерева.
- Возможны конфликты.
  - Начинается удаление объекта.
  - Потом удаление производного объекта.
- Конфликты обрабатываются приостановкой работы.
- При конфликте объекты добавляются в другую очередь.
  - Чтобы избежать бесконечных циклов.
  - Циклы возможны, если поддерево в другой очереди.

# Строгая иерархия таблиц

- Каждой таблице присваивается уровень.
- Только таблицы с большим уровнем могут помещаться в таблицы с меньшим.
  - Подобно таблицам страниц.
- Граф таблиц становится ациклическим.
- Возникает проблема создания таблиц верхнего уровня.
  - Решается созданием локальных для потока слотов.

# Локальные для потока слоты

- Являются таблицей верхнего уровня.
- Размер — также степень 2, указывается при создании потока.
  - Как правило — небольшой (32 слота).
- Требуется возможность доступа извне.
- Через локальные слоты происходит инициализация процесса.
- Также полезны как временные имена.
  - Для IPC требуется копирование для сокращения прав.
  - Может служить в качестве получателя при IPC (новый механизм).
  - Выделяются при помощи битовой маски — быстрое выделение и освобождение.
  - При нехватке может использовать глобальный аллокатор имен.

# Thread sets (1)

- Для потоков циклы неизбежны.
  - Поток должен ссылаться на себя и другие потоки.
- Можно не решать проблему утечки ресурсов в ядре.
- Вводится привилегированный жетон — thread set.
  - Позволяет получить жетоны на потоки в этом наборе.
  - *Подобно `zx_object_get_child()` в Zircon.*
  - Итерация без состояния.
    - `sys_threadset_next(cap_t tset, cap_t prev_thread, cap_t dst);`
    - Из ссылки на поток получается ссылка на следующий поток.
    - Требуется `KPROT_READ`.
  - При создании потока указывается tset.
    - Требуется `KPROT_EXECUTE`.
  - Возможно создавать дочерние thread sets.
    - Требуется `KPROT_WRITE`.

# Thread sets (2)

- Сборка мусора может реализоваться в пространстве пользователя.
- При удалении все ссылки перемещаются в родительский thread set.
  - Если родителя нет, ядро паникует.
- Также полезны для:
  - интроспекции;
  - отладки;
  - реализации менеджера процессов;
  - поддержки пространства имен PID POSIX.