Яндекс

Память –
Идеальная Абстракция

Короткий Фёдор

Что такое «Хорошая» абстракция?

«Хорошая» абстракция

- Имеет минимальный интерфейс
- Широко используется
- Имеет множество различных реализаций
- Пользователь не думает о существовании абстракции

new и delete

- Имеет минимальный интерфейс
- Широко используется
- Имеет множество различных реализаций
- Пользователь не думает о существовании абстракции

Что делает этот код?

```
int* ptr = new int;
*ptr = 42;
delete ptr;
```

- Выделяет на куче 4 байта
- Пишет 0x0000002a по адресу ptr
- Освобождает объект по адресу ptr

Что делает этот код?

```
int* ptr = (int*) malloc(4);
*ptr = 42;
free(ptr);
```

- Выделяет на куче 4 байта
- Пишет 0x0000002a по адресу ptr
- Освобождает объект по адресу ptr

Disclaimer

- Я буду говорить про конкретную реализацию (jemalloc, linux, x86_64)
- Реализация на вашей системе может отличаться
- Принципы работы везде одинаковые

Что делает этот код?

```
int* ptr = (int*) malloc(4);
*ptr = 42;
free(ptr);
```

- Мы заглянем внутрь аллокатора jemalloc
- Посмотрим как linux управляет памятью
- Узнаем какие механизмы внутри «железа» позволяют все это реализовать

Что делает этот код?

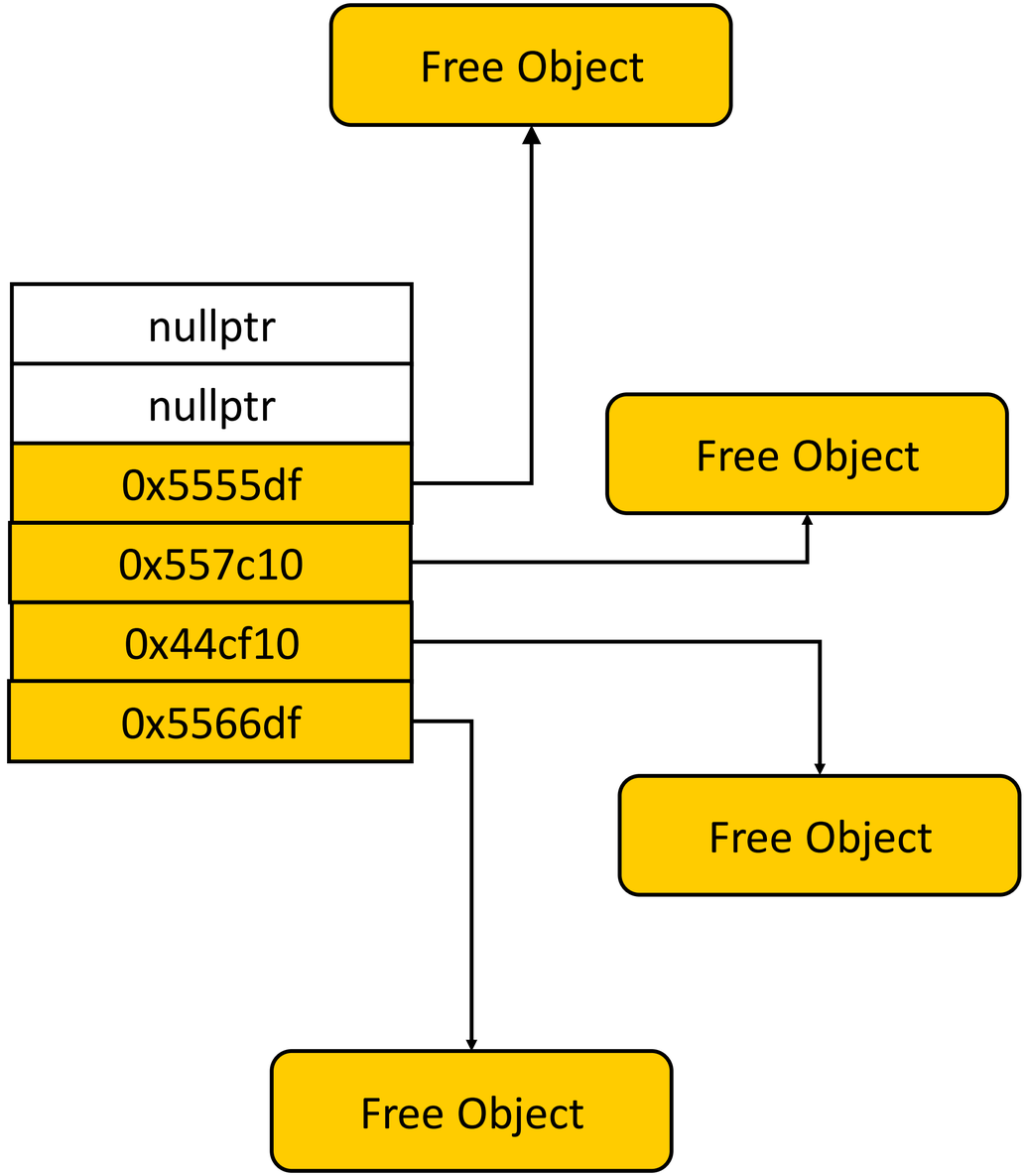
```
int* ptr = (int*) malloc(4);
*ptr = 42;
free(ptr);
```

Классы размеров

- Small – меньше 16KiB
  - 16, 32, 48, 64, … 160, 192, … 14KiB
- Large – больше 16KiB
  - 16KiB, 20KiB, … 48MiB, 56MiB, 64MiB …

tcache

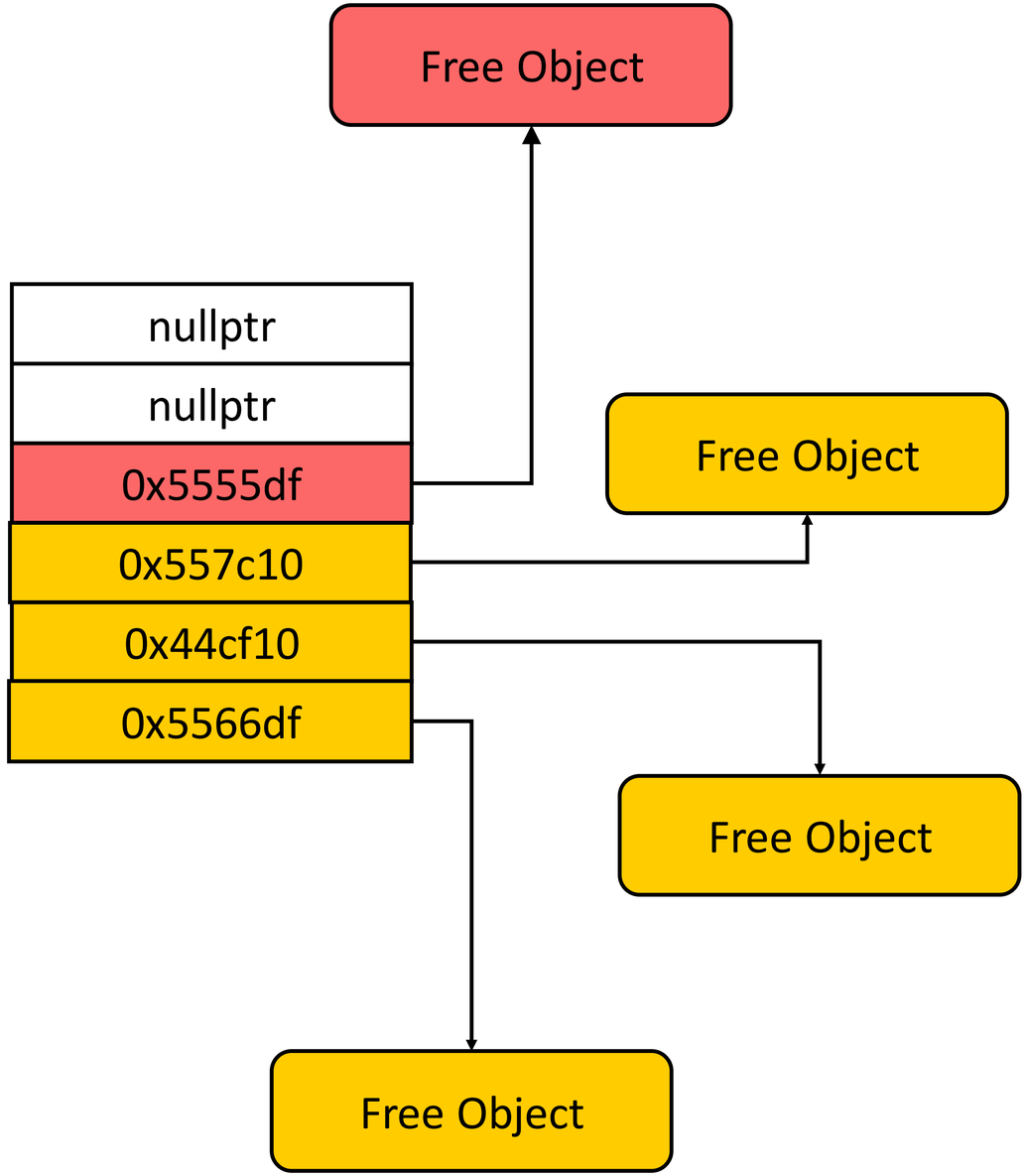
- Локальный для потока кэш свободных объектов
- Очень быстрый и очень глупый
- void* clip[16];



tcache → arena → slab → extent → vma → pte → tlb → rtree → purge

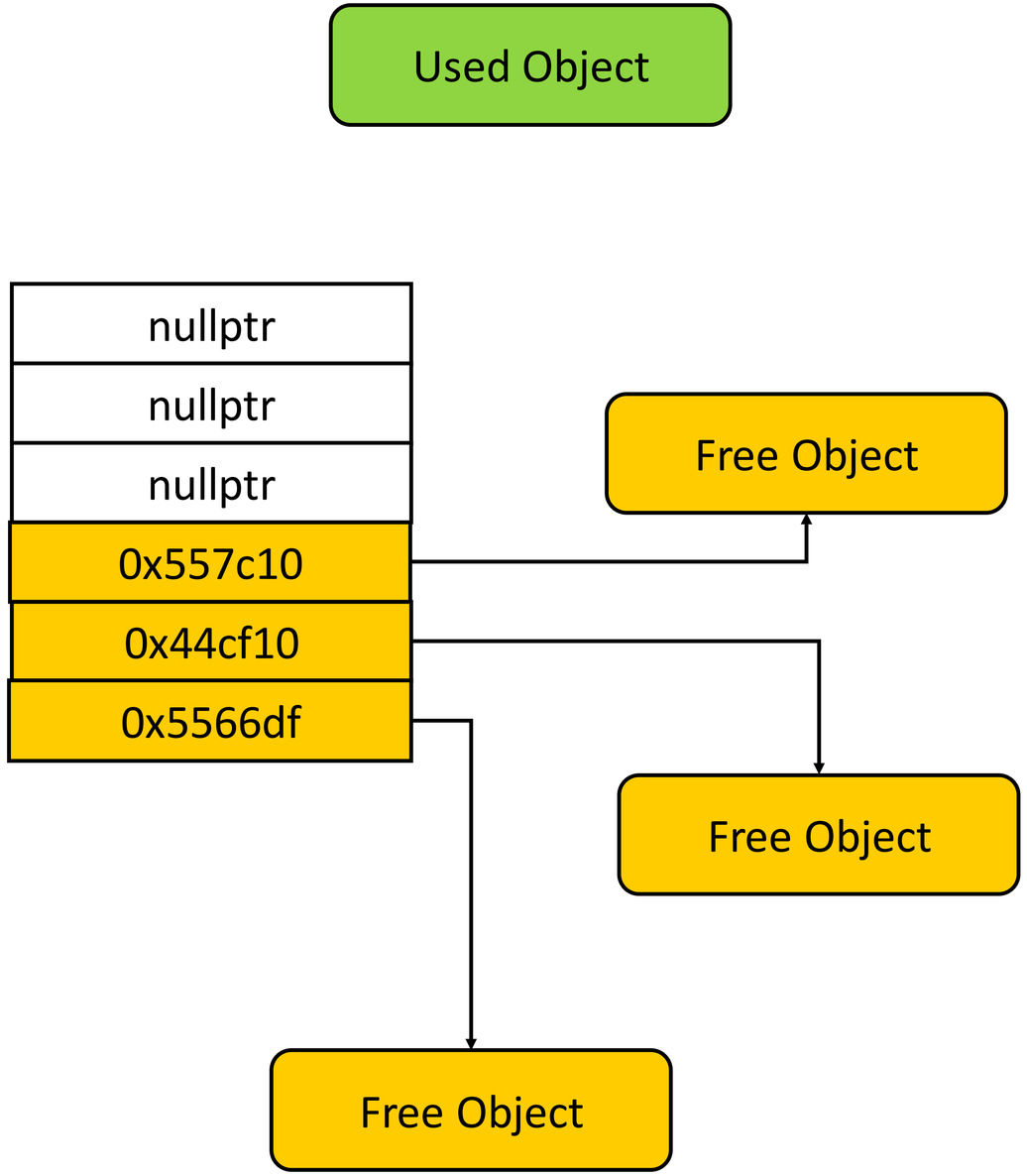
tcache

- Локальный для потока кэш свободных объектов
- Очень быстрый и очень глупый
- void* clip[16];



tcache → arena → slab → extent → vma → pte → tlb → rtree → purge

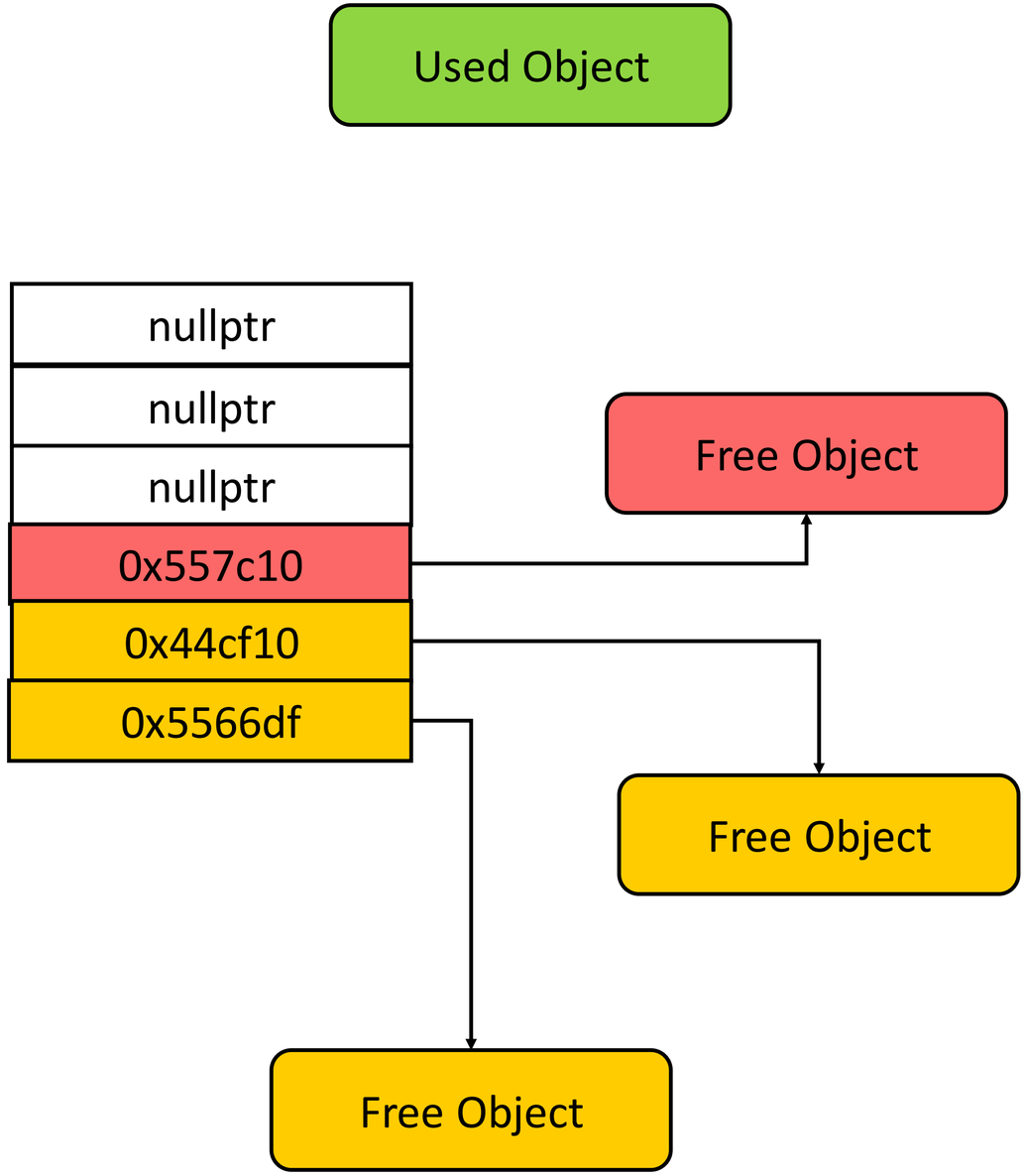
tcache



- Локальный для потока кэш свободных объектов
- Очень быстрый и очень глупый
- void* clip[16];

tcache → arena → slab → extent → vma → pte → tlb → rtree → purge

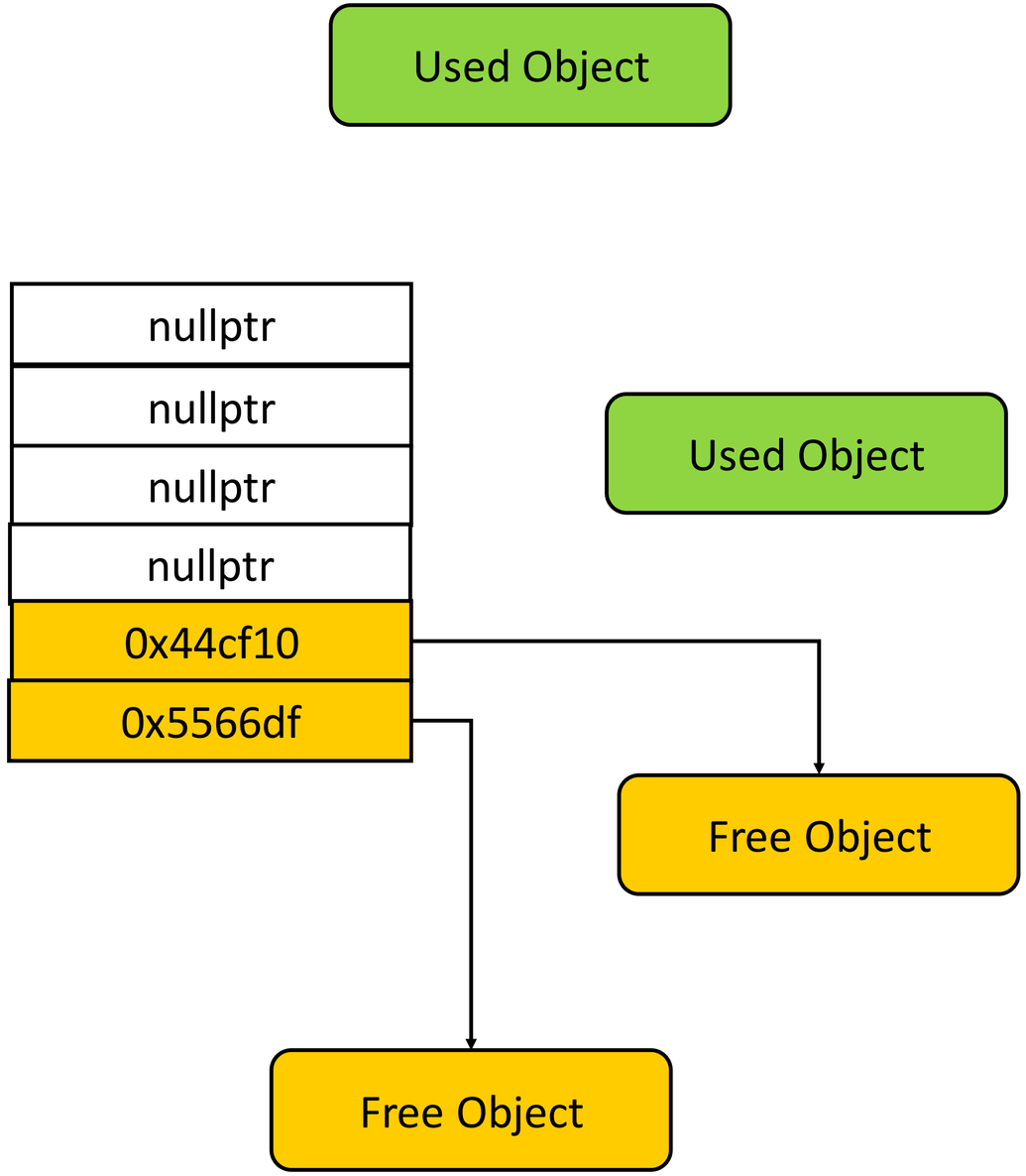
tcache



- Локальный для потока кэш свободных объектов
- Очень быстрый и очень глупый
- void* clip[16];

tcache → arena → slab → extent → vma → pte → tlb → rtree → purge

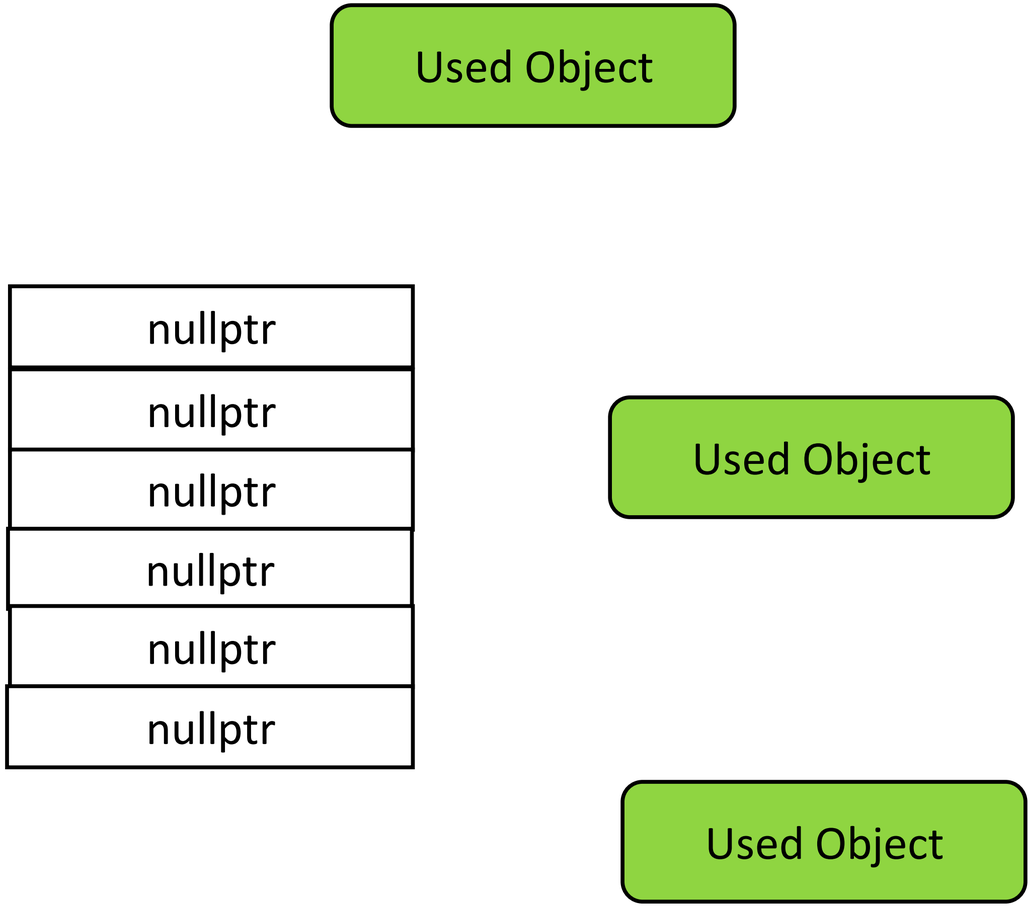
tcache



- Локальный для потока кэш свободных объектов
- Очень быстрый и очень глупый
- void* clip[16];

tcache → arena → slab → extent → vma → pte → tlb → rtree → purge

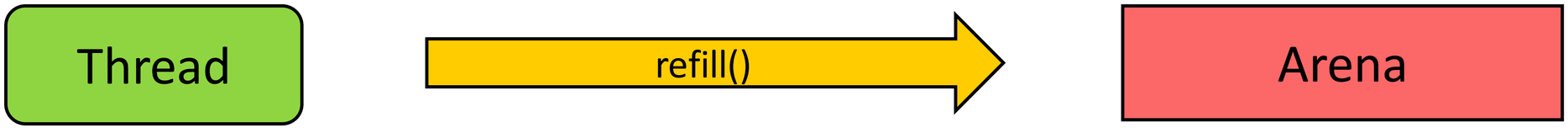
tcache



- Локальный для потока кэш свободных объектов
- Очень быстрый и очень глупый
- void* clip[16];

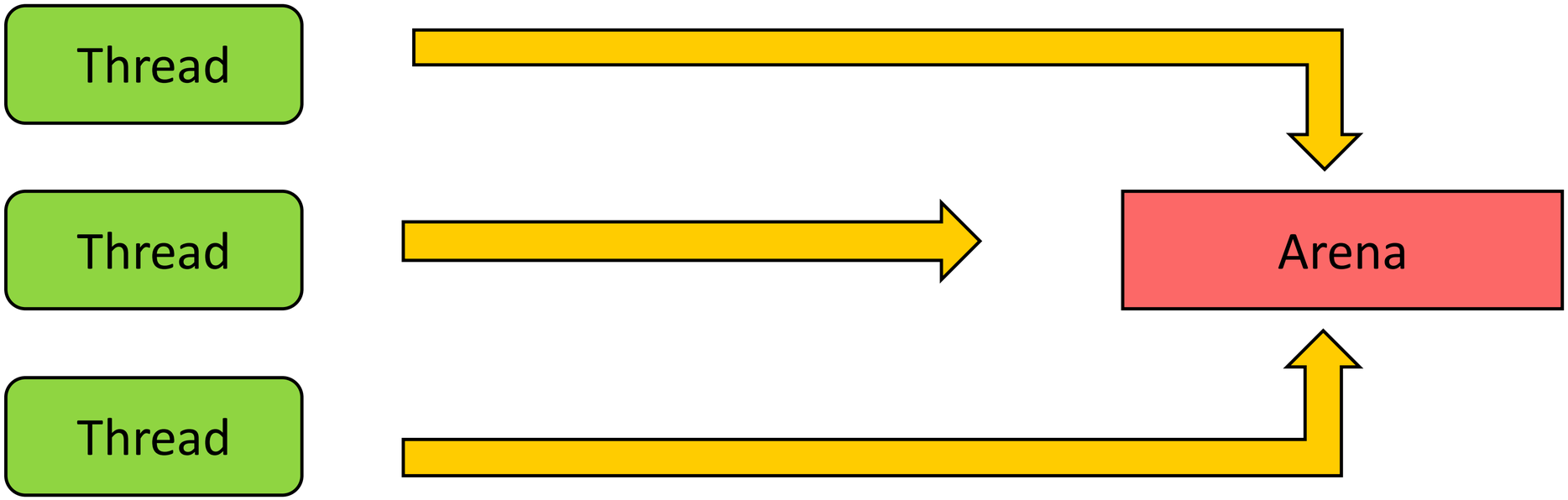
arena

- Если локальный кэш опустел, его нужно наполнить



arena

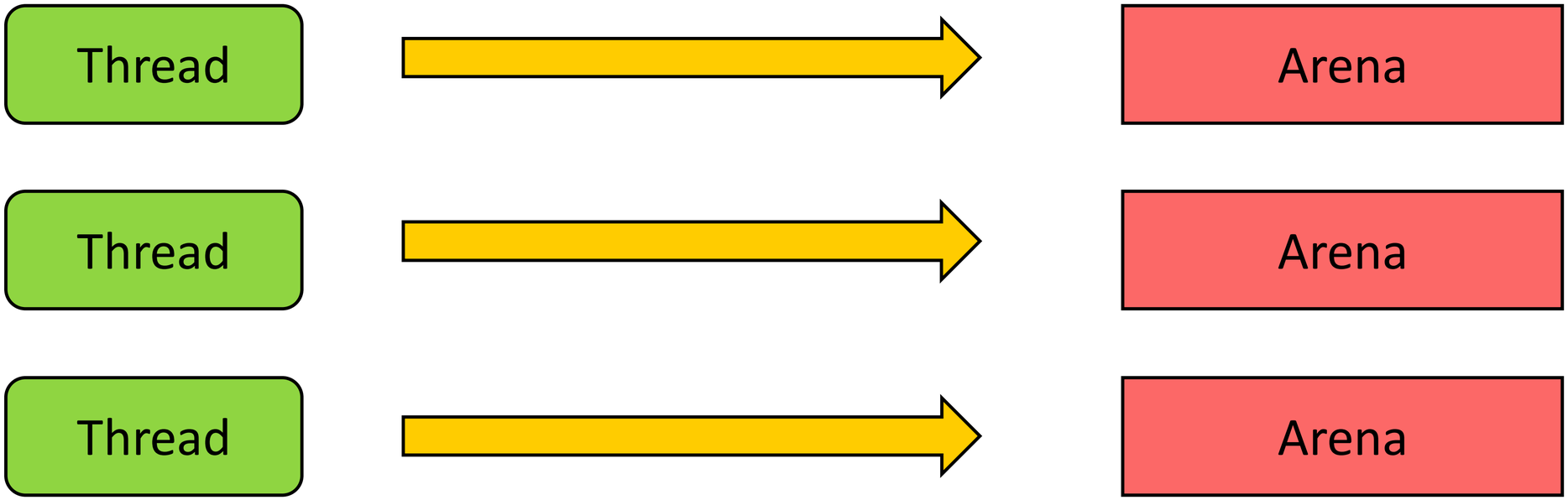
- Если локальный кэш опустел, его нужно наполнить



tcache | **arena** | slab | extent | vma | pte | tlb | rtree | purge

arena

- Чтобы бороться с contention, глобальная куча разделена на независимые арены
- Потоки выделяют память из разных арен



tcache | **arena** | slab | extent | vma | pte | tlb | rtree | purge

Управление маленькими объектами в арене

- Мы пришли в арену и запросом
  - Дай мне 8 объектов размера 16 байт
- Мы просим много маленьких объектов
  - Важно держать накладные расходы на один объект небольшими
  - Важно бороться с фрагментацией
  - Не нужно освобождать объекты сразу

slab

Slab – массив объектов одинакового размера + bitmap

Чтобы найти свободные объекты – сканируем биты

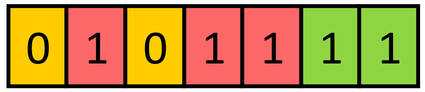
Бонус – объекты скорее всего будут лежать рядом в памяти

slab

Slab – массив объектов одинакового размера + bitmap

Чтобы найти свободные объекты – сканируем биты

Бонус – объекты скорее всего будут лежать рядом в памяти

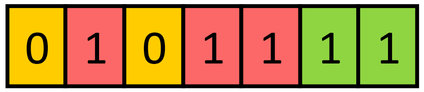


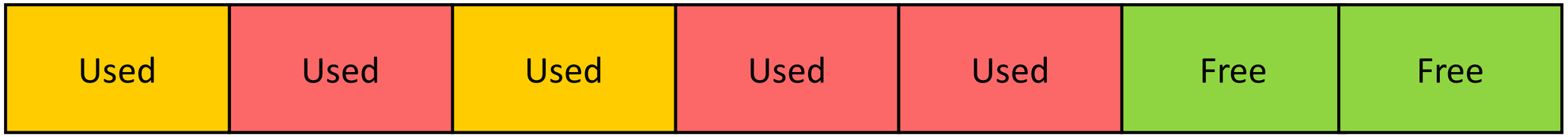
slab

Slab – массив объектов одинакового размера + bitmap

Чтобы найти свободные объекты – сканируем биты

Бонус – объекты скорее всего будут лежать рядом в памяти





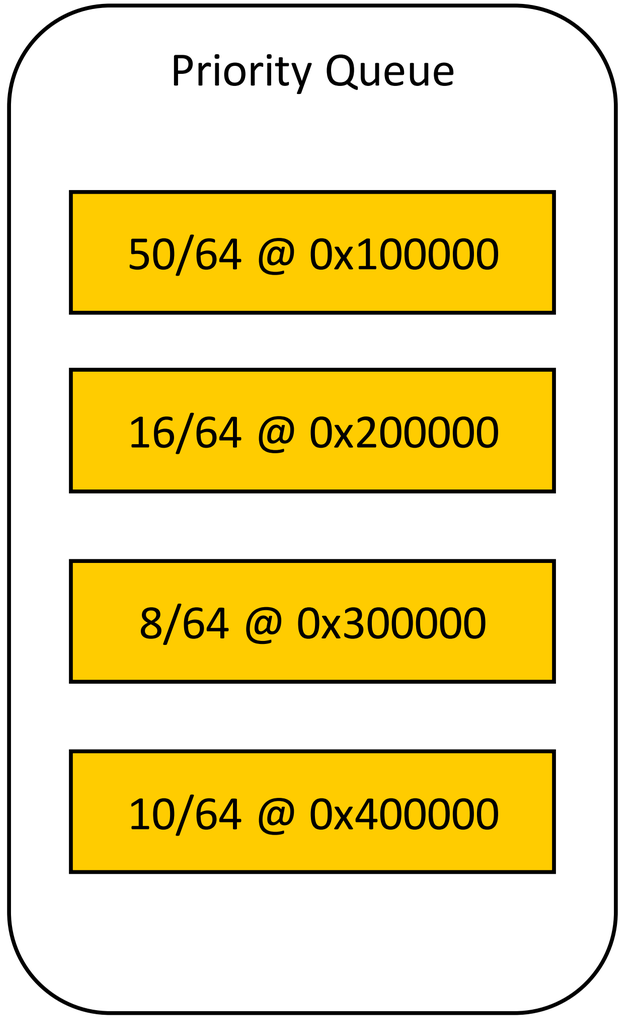
slab

Что делать если slab полностью заполнен?

- Вариант А: Поискать свободный slab
- Вариант Б: Создать новый slab

Ищем свободный slab

- Все незаполненные slab-ы хранятся в очереди
- Ключом в очереди является адрес slab-а
- Такой подход всегда выделяет новые объекты из более старого slab-а



Создаем новый slab

- Чтобы выделить блок под новый slab – вызовем malloc() рекурсивно :)
- Внутри slab хранились маленькие объекты – но сам slab является большим объектом
- Выделение новых slab-ов обсуживается наравне с пользовательскими аллокациями больших размеров

Что делать с большими объектами?

Два наблюдения

- Больших объектов мало
- Большие объекты - большие :)

Важно эффективно переиспользовать память больших объектов

Неправильно

Правильно

Что делать с большими объектами?

Два наблюдения

- Больших объектов мало
- Большие объекты - большие :)

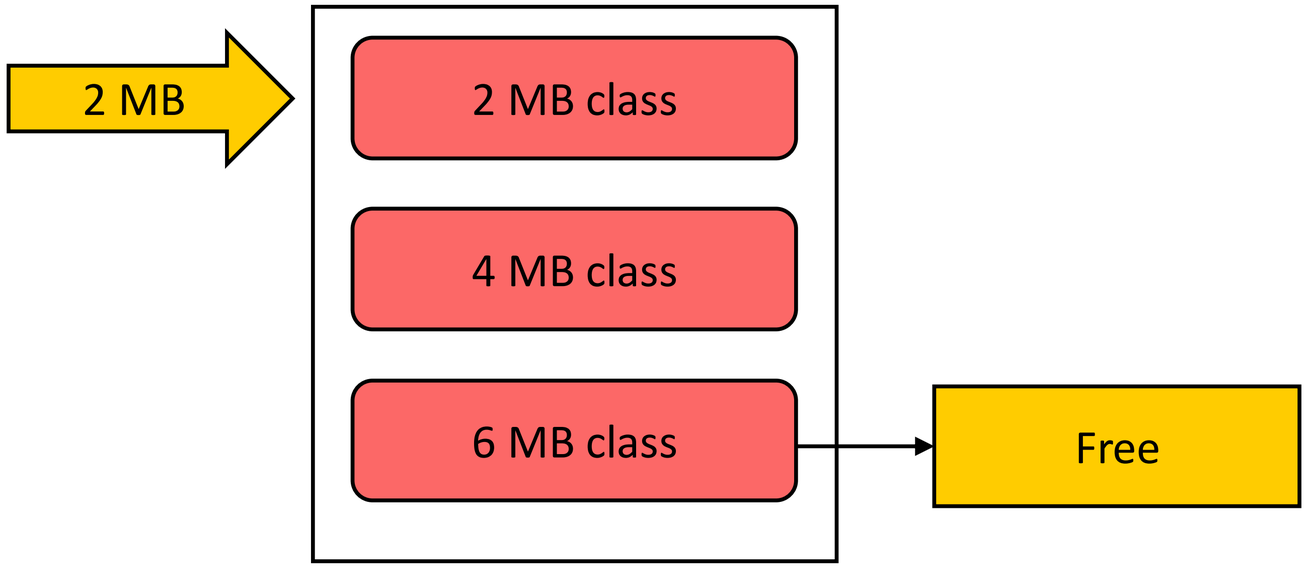
Важно эффективно переиспользовать память больших объектов

Неправильно

Правильно

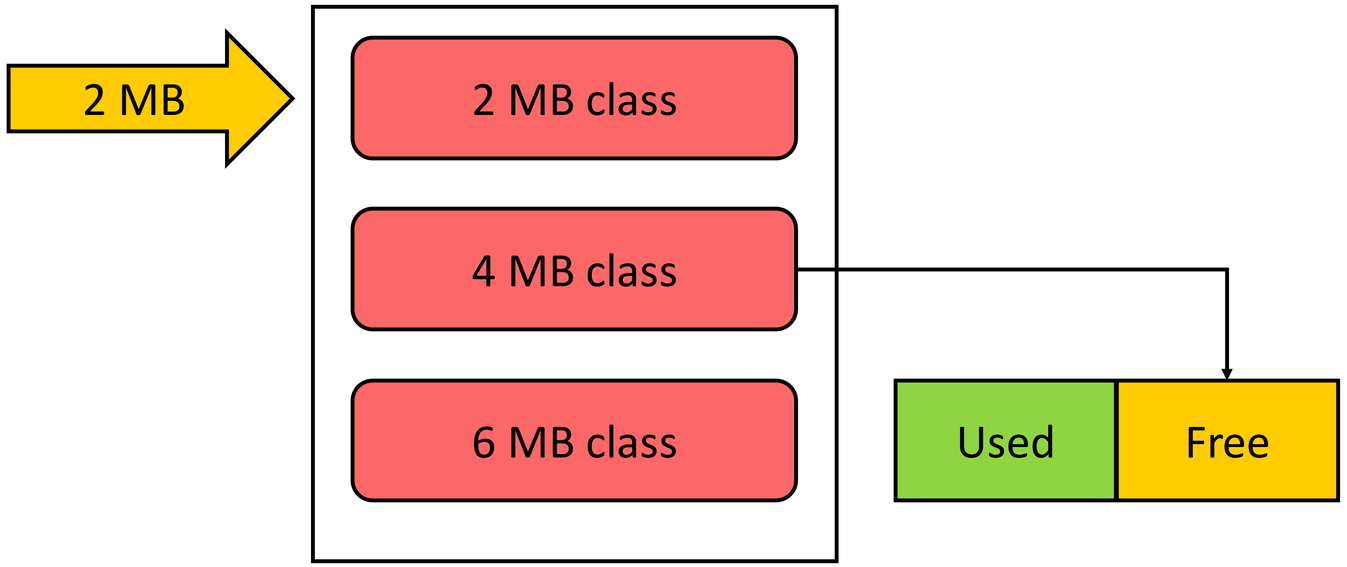
extents

- extent – большой объект
- extent-ы тоже делятся на классы по размеру
- Внутри класса мы храним объекты в очереди с приоритетами



extents

- extent - большой объект
- extent-ы тоже делятся на классы по размеру
- Внутри класса мы храним объекты в очереди с приоритетами



tcache | arena | slab | **extent** | vma | pte | tlb | rtree | purge

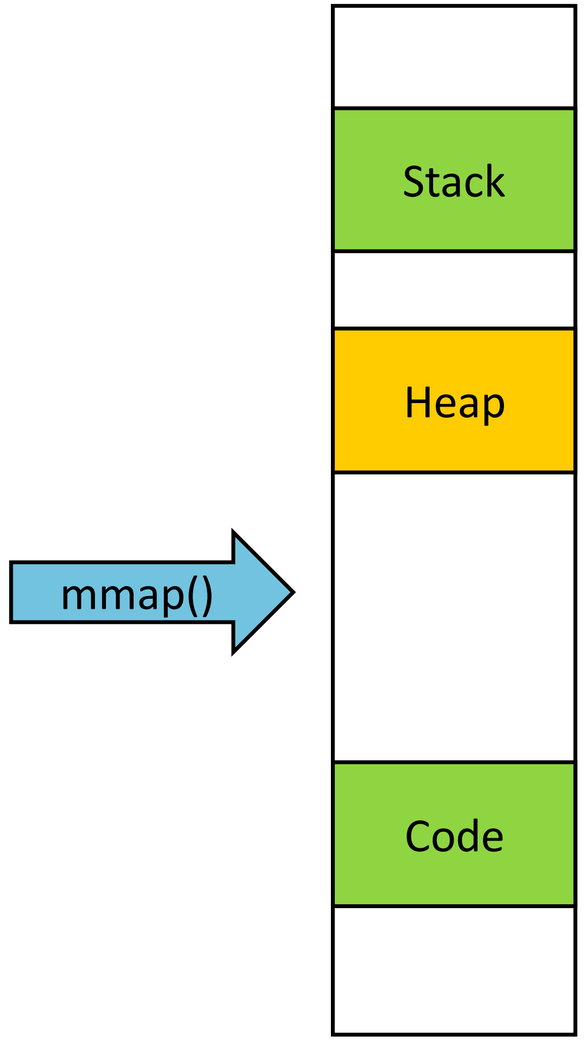
Что делать если нет свободного extent?

Что делать если нет свободного extent?

- Нужно создать новый extent
- Просим памяти у операционной системы
- mmap(3) – системный вызов

Что делает mmap?

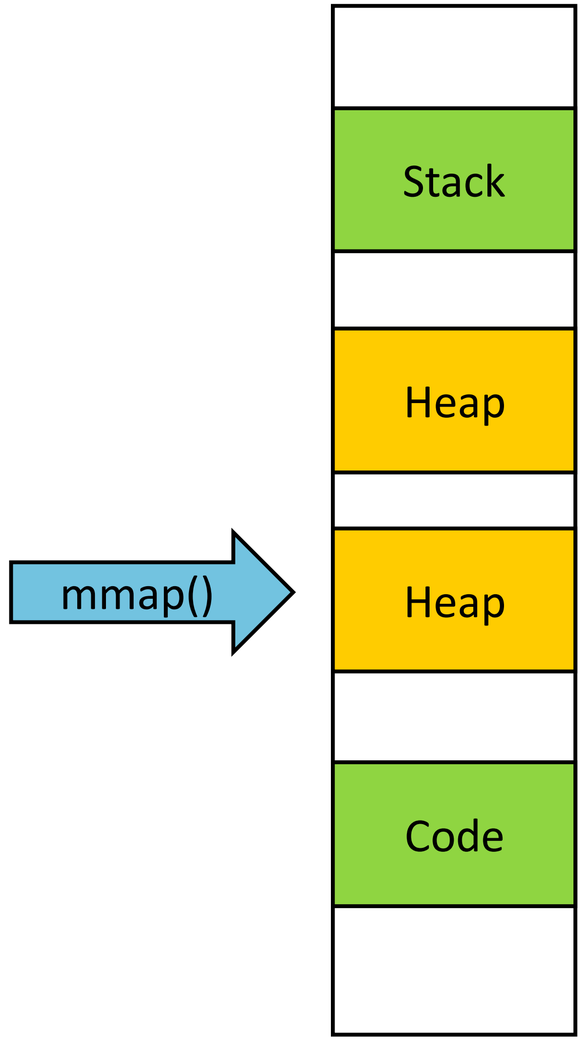
- Вы просите у ОС еще 4 Mb
- ОС хранит множество всех непрерывных участков виртуальной памяти
- ОС находит «дырку» и создаёт там новый участок



tcache | arena | slab | extent | **vma** | pte | tlb | rtree | purge

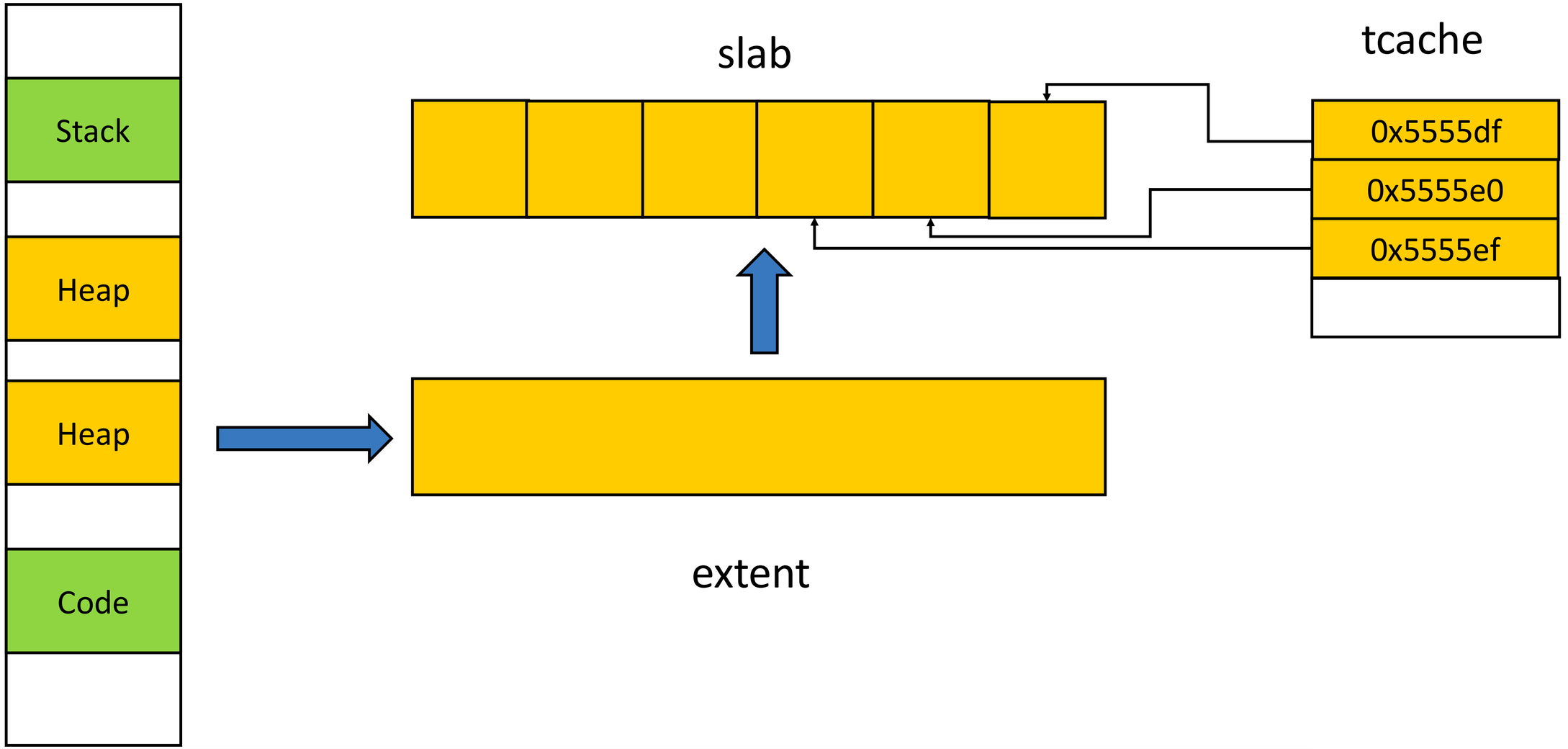
Что делает mmap?

- Вы просите у ОС еще 4 Mb
- ОС хранит множество всех непрерывных участков виртуальной памяти
- ОС находит «дырку» и создаёт там новый участок



tcache | arena | slab | extent | **vma** | pte | tlb | rtree | purge

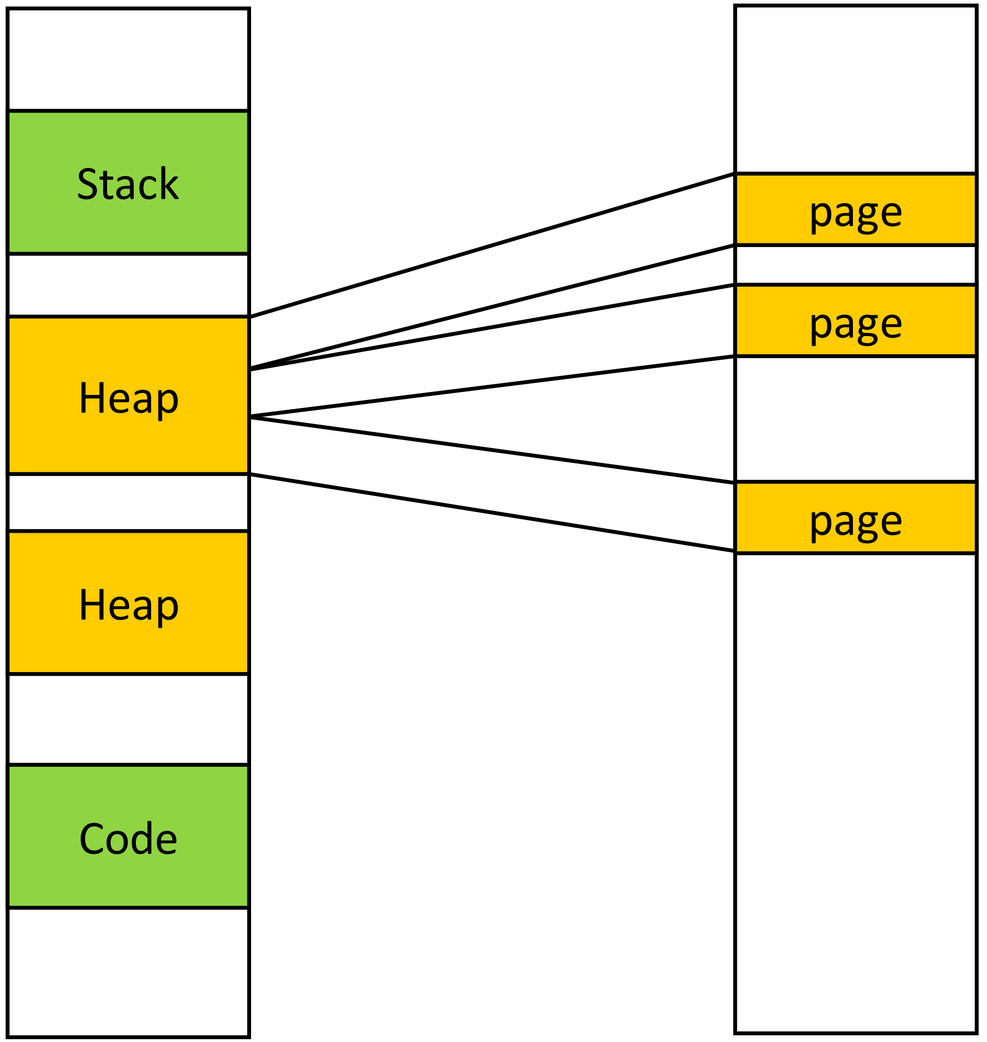
Возвращаемся наверх



tcache → arena → slab → extent → **vma** → pte → tlb → rtree → purge

Мы выделили память?

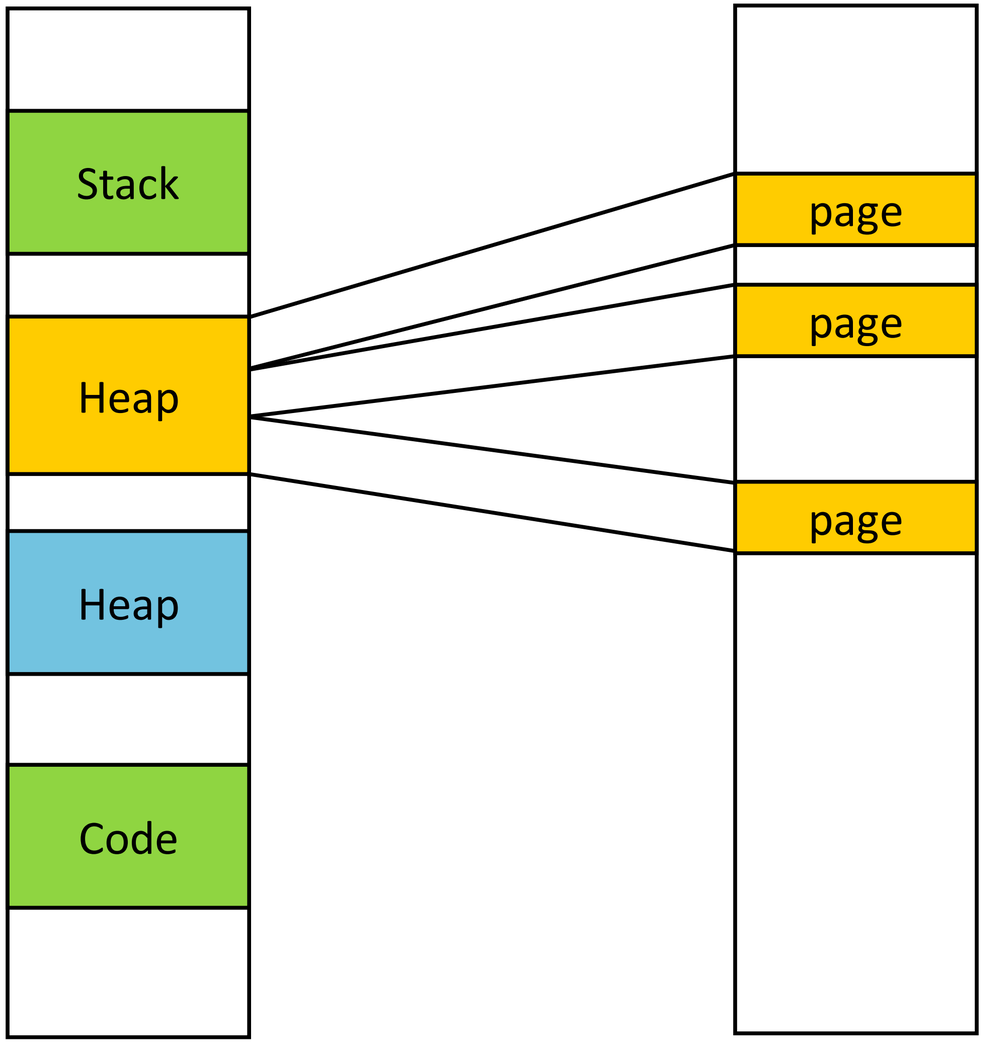
- Не совсем…
- Вызов mmap() создаёт новую запись VMA
- Но эта вируальная память не обеспечена физической памятью



tcache → arena → slab → extent → vma → **pte** → tlb → rtree → purge

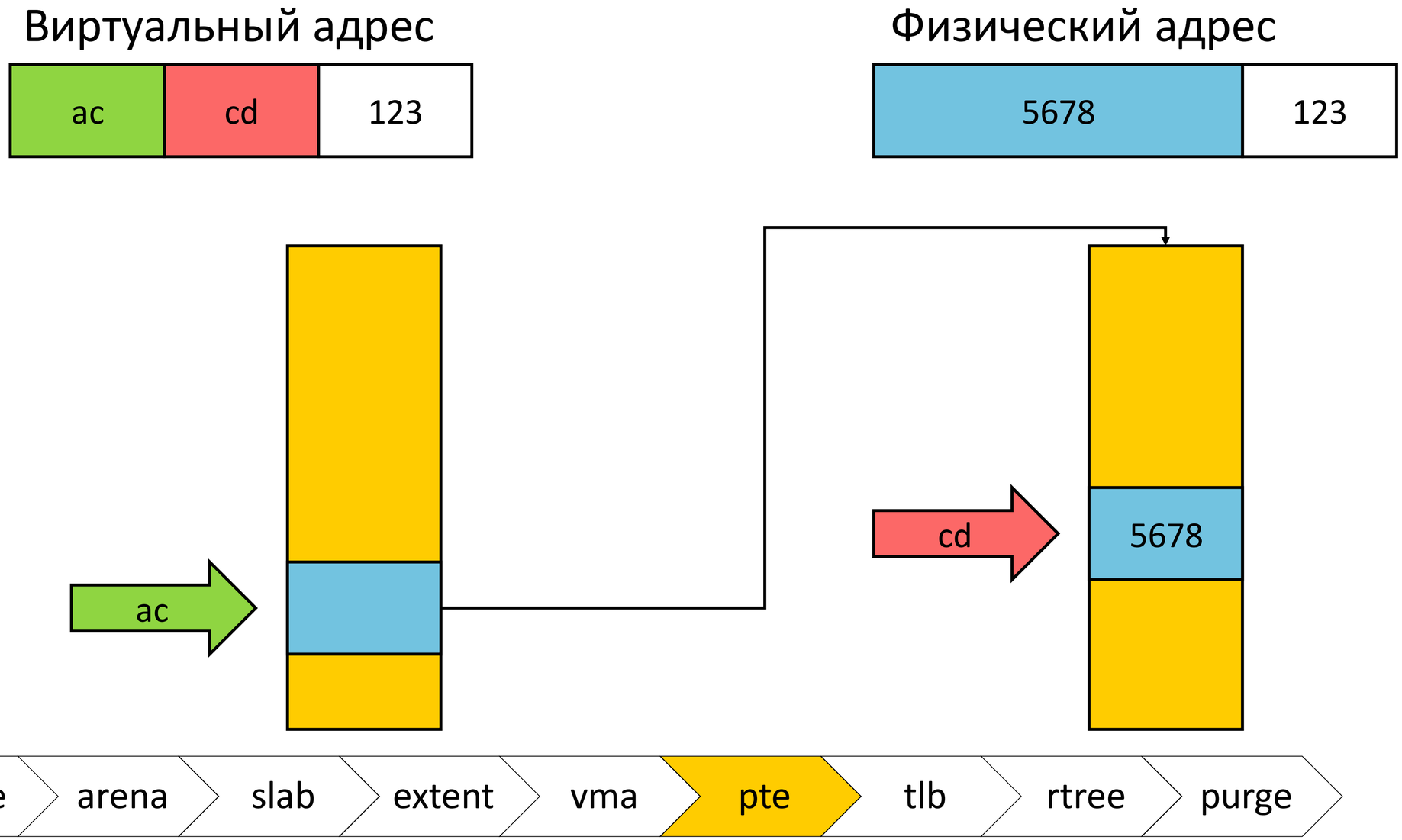
Трансляция адресов

- Трансляция адресов преобразует виртуальный адрес в физический
- Оперирует над страницами
- Реализована в «железе»
- Выполняется на **каждое** обращение к памяти



tcache → arena → slab → extent → vma → **pte** → tlb → rtree → purge

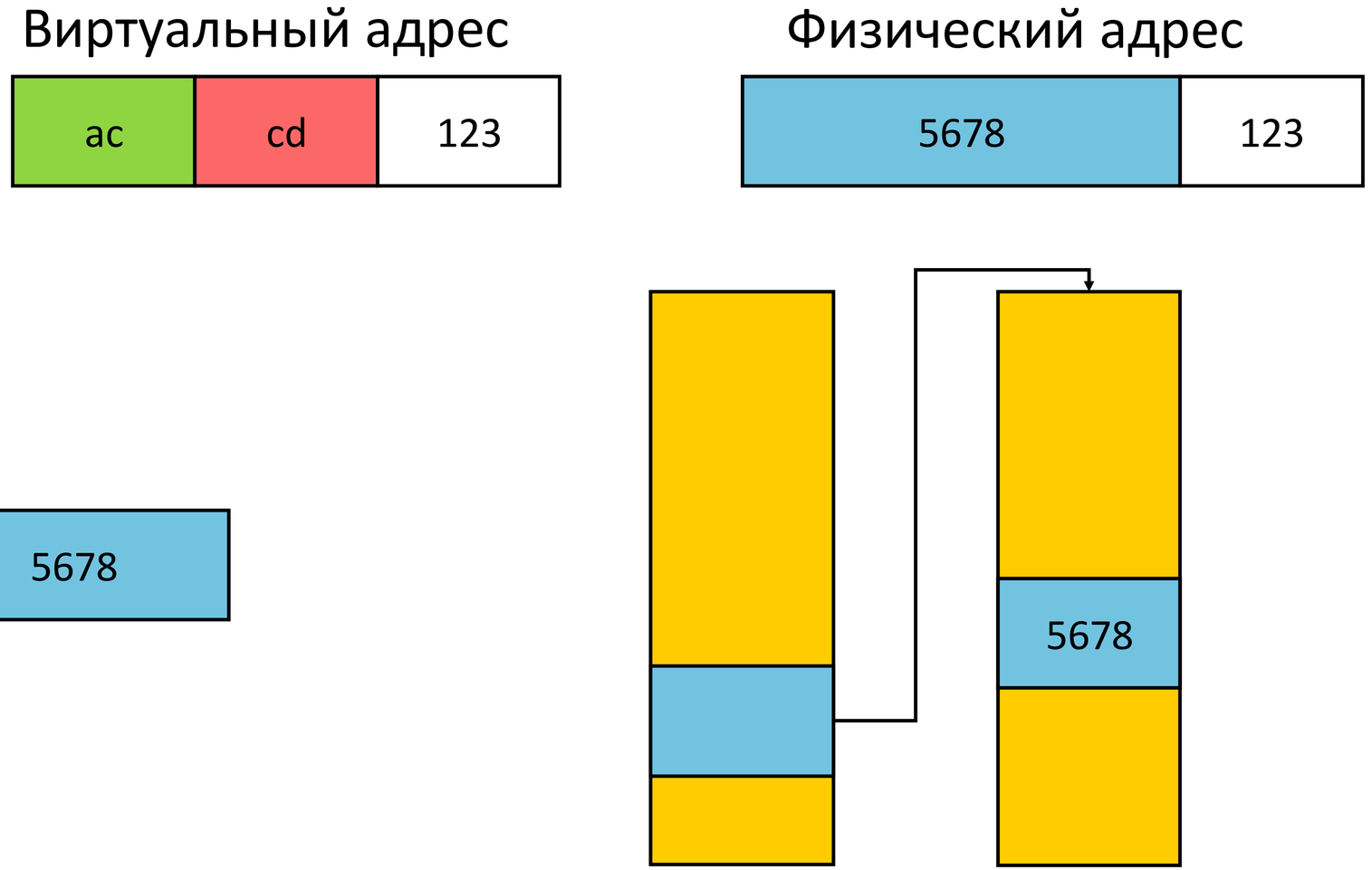
Структура данных Page Table



Ленивое выделение физической памяти

- *ptr = 42;
- CPU ищет ptr в page table
- Не находит и генерирует page fault
- ОС обрабатывает page fault и смотри в VMA
- ОС изменяет page table и продолжает выполнение пользовательского процесса

TLB – кэш над page table



Освобождаем память

- free(ptr);
- А какой был размер?
- А что это за extent?

Освобождаем память

- free(ptr);
- А какой был размер?
- Какие объект лежат в памяти рядом?
- Нужно хранить отображение из ptr в extent

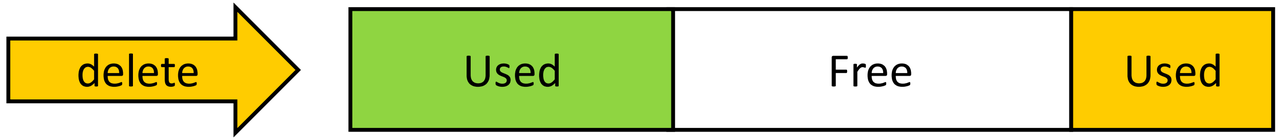
Освобождаем память - rtree

- rtree – глобальный radix tree
- Хранит отображение address -> extent metadata

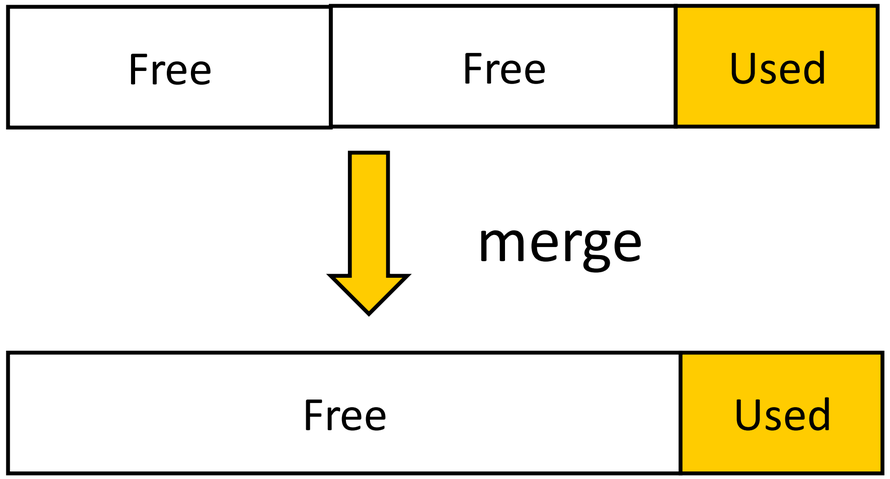
Освобождаем память

- rtree – глобальный radix tree
- Хранит отображение address -> extent metadata
- void free(void* ptr, size_t size_class);

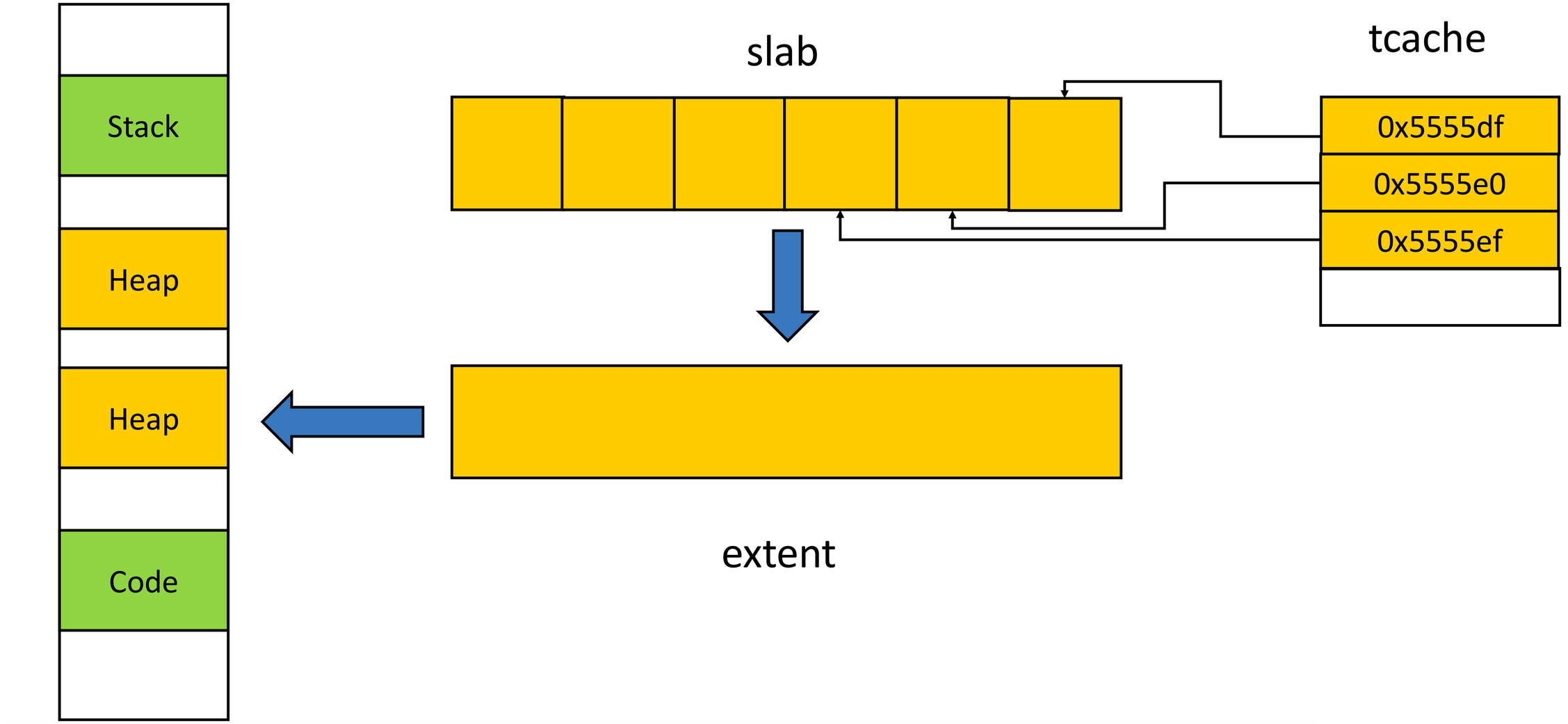
Освобождаем память – слияние extent



- address -> extent metadata
- rtree – используется чтобы найти соседей



Освобождаем память

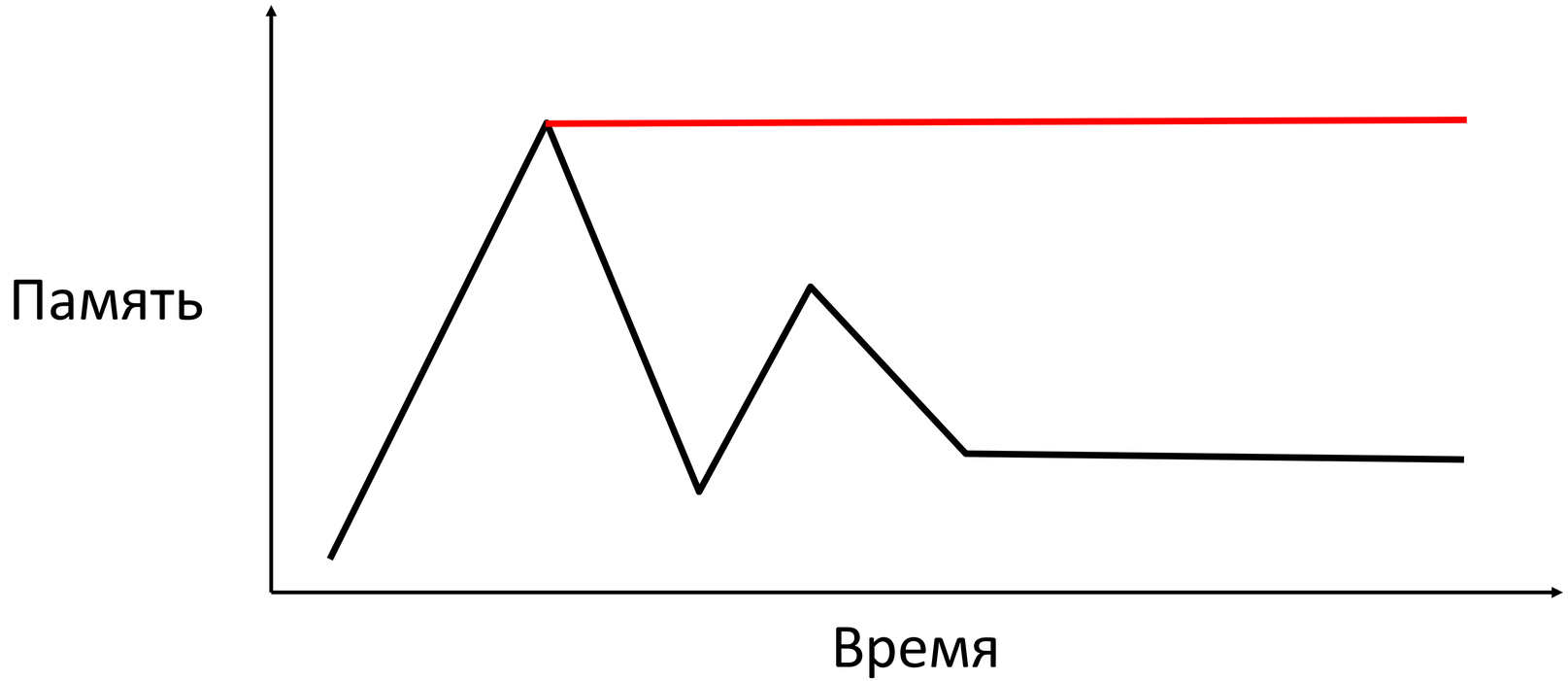


tcache → arena → slab → extent → vma → pte → tlb → **rtree** → purge

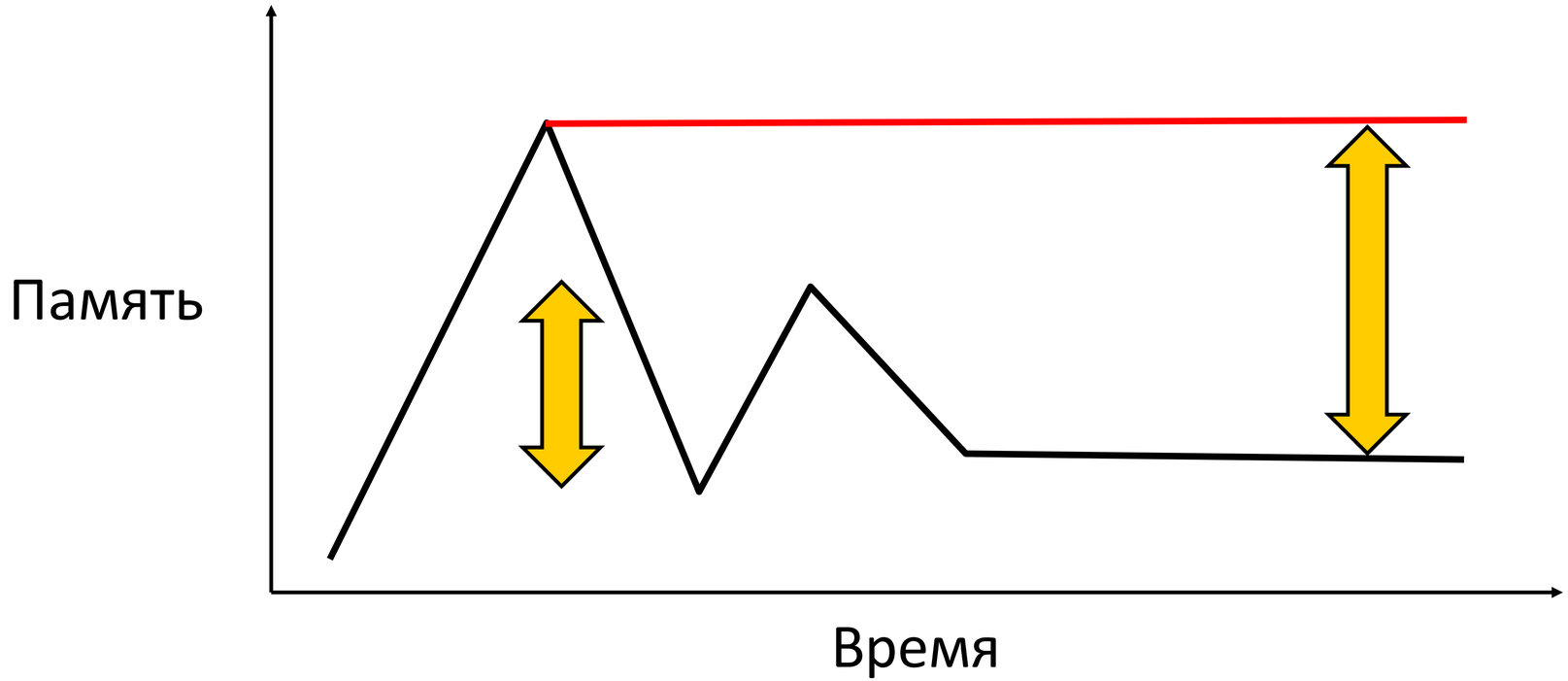
Отдаём память в систему - purging

- Хоти отдать неиспользуемую память
- Но что если она нам снова понадобится?
- Будем освобождать память асинхронно с задержкой в 5 секунд

Отдаём память в систему - purging

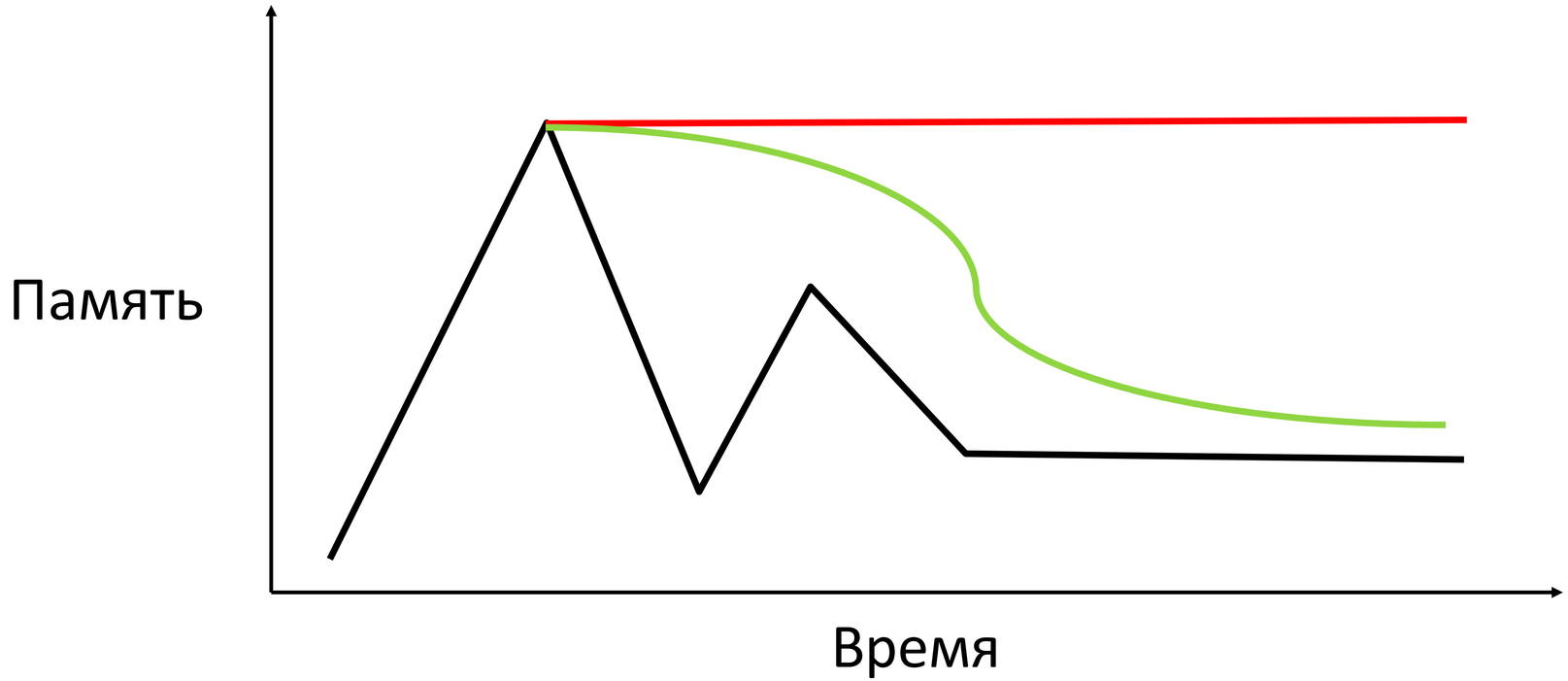


Отдаём память в систему - purging



- Стратегия «возвращать сразу» делает лишнюю работу
- Стратегия «не возвращать никогда» потребляет лишнюю память

Отдаём память в систему - purging



- Будем освобождать память асинхронно с задержкой в 5 секунд (зелёная прямая)

tcache → arena → slab → extent → vma → pte → tlb → rtree → purge

Что на самом делает этот код?

```
int* ptr = (int*) malloc(4);
*ptr = 42;
free(ptr);
```

1. tcache
2. slab
3. extent
4. vma
5. page table
6. tlb
7. rtree
8. purging

Спасибо

Короткий Фёдор

Группа разработки Map-Reduce, Яндекс

 prime@yandex-team.ru