Лабораторна робота №2
Алокатор пам’яті загального призначення з використанням розподілу slab

Ця лабораторна робота не модель алокатору пам’яті, а справжній алокатор пам’яті

Характеристики алокатору №1

Алокатор пам’яті загального призначення побудований в лабораторній роботі №1 мав бути апаратнонезалежним і системнонезалежним. Це означає, що алокатор не знає як представлений покажчик в системі та в компіляторі. Через це була необхідність використовувати заголовок для кожного блоку. Інша умова це можливість працювати з невеликими адресними просторами. Алокатор пам’яті в лабораторній роботі №1 розподіляв пам’ять в аренах випадково (випадково вибиралась арена та випадково вибирався блок для розподілу).

Ці дві умови призводили до внутрішньої та зовнішньої фрагментацій. Внутрішня фрагментація виникає коли розмір payload (корисне навантаження) менший за розмір блоку. Причини внутрішньої фрагментації наступні: існування заголовку у блока (також вирівняний), вирівнювання самого payload, виділення блоку більшого розміру для запитаного меншого розміру. Зовнішня фрагментація виникає коли існує сумарно достатньо пам’яті в арені, але немає достатньо великого блоку для виконання запиту.

Ідея алокатору №2

Якщо взяти до уваги особливості деяких комп’ютерів, операційних систем та компіляторів, то можливо розробити більш ефективний алокатор пам’яті загального призначення. Такі особливості є наступними: сторінкова пам’ять, плоска адресація (або імітація плоскої адресації), компілятор для покажчика використовує представлення рівне адресі пам’яті на яку вказує покажчик.

В цій лабораторній роботі буде використовуватись розподіл slab (кусок, плита), який є більш ефективним способом розподілу пам’яті, який запобігає значній фрагментації та має алгоритми з константною складністю.

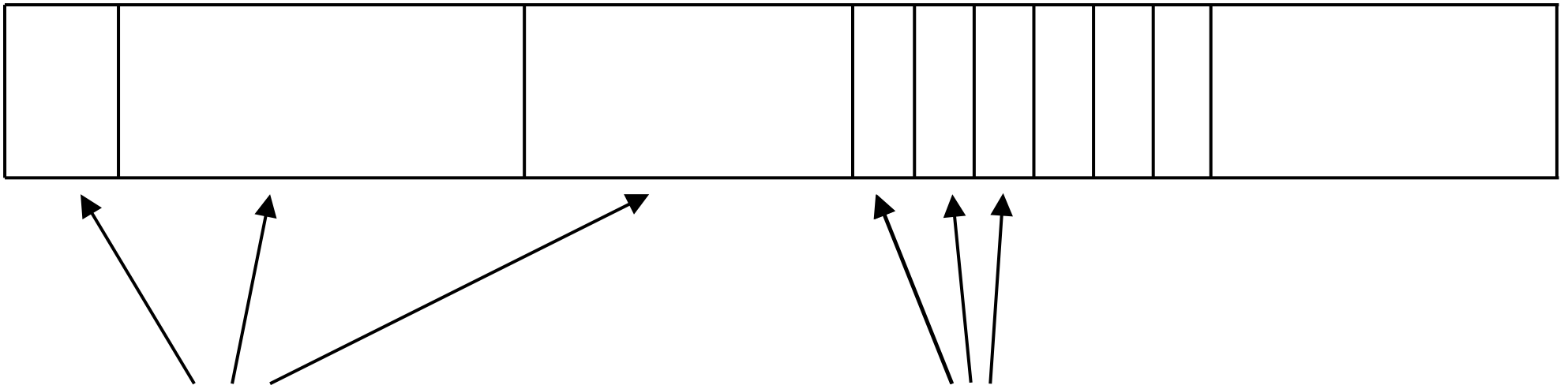
Slab

Основна ідея цього механізму полягає в тому, що об’єкти одного розміру зберігаються в відповідному slab’і, звідси алокатор бере вільні об’єкти і після звільнення об’єкти повертаються в slab, звідки вони були отримані. В пам’яті можуть бути декілька slab’ів, як для об’єктів одного розміру, так для об’єктів різних розмірів, кількість розмірів обмежена через введення регулярності розмірів.

Термінологія

- Арена – це пам’ять отримана від ядра в адресний простір процесу. Арена це неперервна частина пам’яті, що складається з цілої кількості сторінок і початок арени вирівняний на початок сторінки.
- Extent (обсяг, протяжність) – це пам’ять отримана від ядра в адресний простір процесу або частина арени, що складається з цілої кількості сторінок і початок extent’у вирівняний на початок сторінки. Extent може бути використаний для створення slab’у або може бути повернений користувачу відповідно до його запиту на виділення пам’яті.
- Slab – це extent з арени в якому містяться об’єкти однакового розміру. В slab’і міститься масив об’єктів однакового розміру.
- Об’єкт – це пам’ять певного розміру, що є частиною slab’у.
- Кеш – це список вільних об’єктів або extent’ів однакового розміру. Замість списку об’єктів однакового розміру може бути список slab’ів з вільними об’єктами однакового розміру.

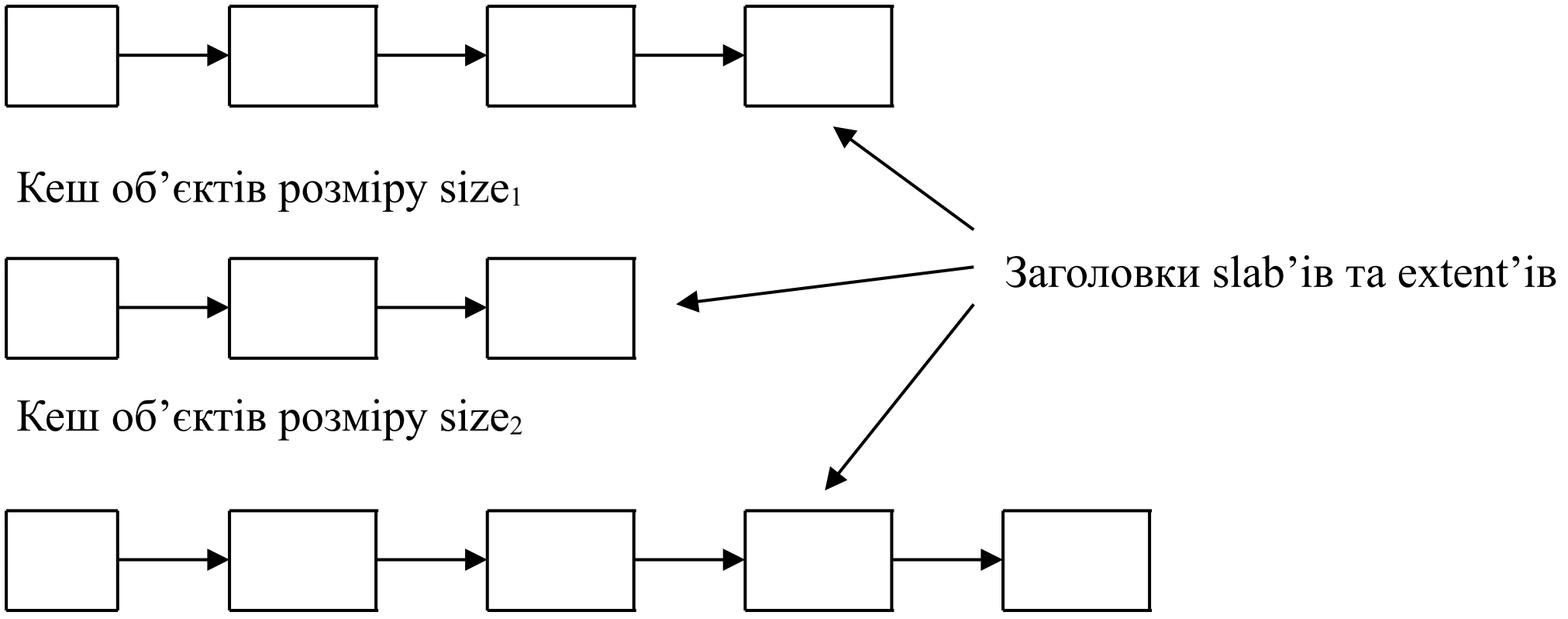
Термінологія



Extent’и в арені,
розмір кожного extent’у
ціла кількість сторінок

Об’єкти в slab’і, що створений
в extent’і в арені

Термінологія



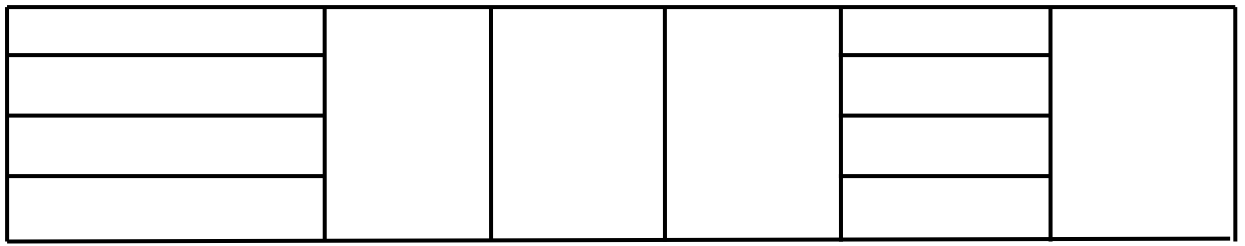
Отримання об’єкту зі slab’у

Для отримання об’єкту зі slab’у достатньо перевірити чи є відповідний кеш об’єктів не порожнім. Якщо кеш об’єктів не порожній, то об’єкт береться з кешу, якщо його slab став заповненим (всі об’єкти slab’у зайняті), то slab забирається з свого кешу. При звільнені об’єкту, він повертається як вільний в свій slab. Якщо slab був до цього заповнений, то він додається до свого кешу. Якщо slab стає порожнім (всі об’єкти slab’у вільні), то extent slab’у повертається в свою арену. Якщо кеш об’єктів порожній, то алокатор отримує вільний extent відповідного розміру і створює з нього slab і додає цей slab в його кеш. Ці дії можливо реалізувати за константний час.

Облік об’єктів

Кожний slab має вести облік своїх об’єктів та має мати покажчики для зв’язування slab’а в його кеші. Цей облік ведеться в заголовку slab’а. Об’єкти поділяють на маленькі та великі. Маленькі об’єкти – це об’єкти розмір яких менший або порівняний з розміром заголовку slab’а, об’єкти інших розмірів називаються великими. Заголовок slab’а з маленькими об’єктами зберігається в пам’яті самого slab’а. Заголовок slab’а з великими об’єктами зберігається в об’єкті іншого slab’а. Заголовки extent’ів, що не є slab’ами зберігаються в об’єктах slab’ів. Якщо алокатору треба якась пам’ять, то алокатор може запитати цю пам’ять у самого себе.

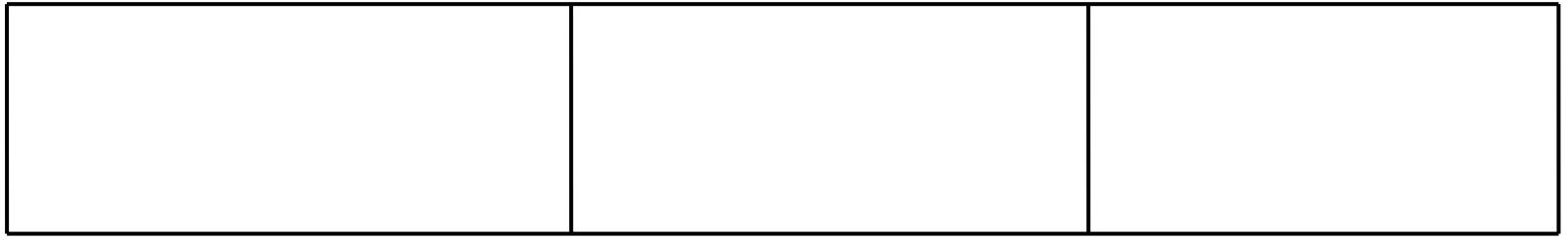
Облік об’єктів



Slab з маленькими об’єктами

Заголовок slab’у розташований в самому slab’і

Заголовок іншого slab’у розташований в об’єкті



Slab з великими об’єктами

Використання технологій

Для реалізації ефективної роботи з пам’яттю арен, extent’ів, slab’ів та кешей використовують наступні технології. Бітові мапи дозволяють компактно представити інформацію про існування чи не існування чогось, швидкість роботи з бітовою мапою залежить від кількості бітів в мапі і може бути невеликою константою. Двох зв’язані списки дозволяють додавати чи забирати елемент за константний час. Radix tree дозволяє за константний час працювати з елементами дерева, якщо ключ має фіксований розмір. Хеш таблиці дозволяють в середньому за константний час працювати з елементами таблиці. Дерево, що самостійно балансує, дозволяє за логарифмічний час працювати з елементами.

Використання технологій

При звільнені пам’яті алокатор має з’ясувати що це за пам’ять і куди її треба повернути. Для реалізації ефективного пошуку використовують наступну ідею. Адреса пам’яті, що звільнюється, конвертується в номер сторінки (це адреса початку сторінки). Потім номер сторінки використовується для пошуку заголовку і з’ясовування чи є ця пам’ять об’єктом в slab’і, extent’ом в арені чи extent’ом не в арені. Якщо адреси всіх арен та extent’ів не з арен вирівнювати на розмір арени, то по адресі пам’яті, що звільнюється, можливо визначити чи ця адреса належить арені та визначити адресу самої арени.

Варіант реалізації: розміри об’єктів

Визначимо розміри об’єктів для slab’ів. Визначаємо максимальну вимогу до вирівнювання align байт у компілятора. Перші n розмірів об’єктів беремо з кроком align (align, 2 * align, 3 * align, …, n * align байт). Нехай $size_n = n * align$. Наступні розміри об’єктів будуть визначатися групами розмірів починаючи з розміру $size_n$ (для першої групи) або максимального розміру об’єкта попередньої групи з кроком $2^p * align$, таких розмірів буде m, де $p \geq 1$ (p це номер групи). Всього таких груп буде k.

Більші розміри округлюються до цілої кількості сторінок.

Коли користувач запитує пам’ять, то за запитаним розміром size визначається чи є це розмір: з перших n розмірів, розмір з k груп, розмір extent’а з арени або розмір extent’а не з арени.

Варіант реалізації: клас slab’а

Конвертування запитуваного розміру в клас slab’у можливо виконати за допомогою таблиці, але в залежності від значень n, m та k розмір цієї таблиці може бути великим. Замість таблиці клас slab’у можливо вираховувати.

Конвертування запитуваного розміру в клас slab’у з перших n розмірів виконується шляхом округлення ділення розміру на align:

$\lceil size / align \rceil$

Конвертування запитуваного розміру в клас slab’у з k груп виконується наступним чином. Максимальний розмір об’єкта в групі p дорівнює:

$size_n + 2^1 * m * align + 2^2 * m * align + ... + 2^p * m * align$
$size_n + m * align * (2^1 + 2^2 + ... + 2^p)$
$size_n + m * align * 2 * (2^0 + 2^1 + ... + 2^{p-1})$
$size_n + m * align * 2 * (2^p - 1)$

Варіант реалізації: клас slab’a

Запитуваний розмір має бути менший або дорівнюватись цьому максимальному розміру. В такому випадку група буде мати номер p. Для того, щоб не використовувати умову в алгоритмі, запишемо це відношення так:

$$size - 1 < size_n + m * align * 2 * (2^p - 1)$$

З цього відношення визначимо p:

$$size - size_n - 1 < m * align * 2 * (2^p - 1)$$
$$\lfloor (size - size_n - 1) / (m * align * 2) \rfloor < 2^p - 1$$
$$\lfloor (size - size_n - 1) / (m * align * 2) \rfloor + 1 < 2^p$$

Значення p це номер біту після старшого встановленого біта в значенні в лівій частині відношення. Якщо align та m це ступені двійки, то множення та ділення виконуються ефективно.

Варіант реалізації: клас slab’a

Номер розміру в групі визначається наступною формулою:

$m - \lfloor (size_n + m * align * 2 * (2^p - 1) - size) / (2^p * align) \rfloor$

Якщо align = 16, n = 8, m = 4 та k = 7 отримуємо наступні розміри: 16, 32, 48, 64, 80, 96, 112, 128 (перші 8 розмірів), 160, 192, 224, 256 (перша група з кроком 32), 320, 384, 448, 512 (друга група з кроком 64), 640, 768, 896, 1024 (128), 1280, 1536, 1792, 2048 (256), 2560, 3072, 3584, 4096 (512), 5 Кб, 6 Кб, 7 Кб, 8 Кб (1 Кб), 10 Кб, 12 Кб, 14 Кб, 16 Кб (сьома група з кроком 2 Кб).

Варіант реалізації: сторінки в slab’і

Оптимальна кількість сторінок для об’єктів кожного розміру визначається наступним відношенням (x це кількість об’єктів в slab’і, y це кількість сторінок в extent’і slab’а):

obj_size * x = page_size * y

Це відношення має сенс для x = page_size та y = obj_size, тому найменша кількість сторінок буде (gcd – найбільший спільний дільник):

obj_size / gcd(obj_size, page_size)

В алокаторі треба задати мінімальну кількість об’єктів у slab’і і максимальну кількість сторінок в extent’і з якого створюється slab. Виходячи з розрахованого значення кількості сторінок і цих значень визначаємо кількість сторінок для створення slab’у.

Варіант реалізації: розмір арени

Розмір арени обираємо як 4096 сторінок пам’яті (якщо розмір сторінки 4 Кб, то розмір арени буде 16 Мб). Всього в аренах можливо створити 4096 extent’ів різних розмірів. Зверніть увагу, що extent розміром в одну сторінку і об’єкт розміром в одну сторінку в slab’і це не те саме. Якщо користувач запитує одну сторінку, то вона шукається у slab’і. Якщо алокатор запитує одну сторінку, то вона шукається серед extent’ів в арені.

Варіант реалізації: кеші

Покажчики на списки slab’ів в кеші групуються в масив. Покажчики на списки вільних extent’ів однакових розмірів в арені також групуються в масив, але в масив групується тільки частина покажчиків. Наступна частина покажчиків зберігається в хеш таблиці (наприклад, метод ланцюжків), де ключ пошуку це розмір extent’а. Пам’ять для масивів та хеш таблиці запитується у ядра. Існування вільних extent’ів певних розмірів кодується бітовою мапою. При пошуку в аренах extent’а необхідного розміру використовуємо стратегію best fit.

Варіант реалізації: облік об’єктів

Для обліку об’єктів в slab’і використовуємо бітову мапу, якщо об’єкт вільний, то його біт дорівнює 1. При запиті об’єкту повертаємо покажчик на перший за порядком вільний об’єкт. Можливо не використовувати бітову мапу, а зв’язати об’єкти у список використовуючи сам об’єкт для вказування індексу наступного вільного об’єкту. Але такий підхід означає, що вільний об’єкт використовується алокатором для обліку цього об’єкту, тому не завжди буде можливим повідомити ядро, що якісь сторінки з вільними об’єктами є вільними.

Варіант реалізації: бітові мапи

Бітова мапа це масив з даних типу цілий. Кількість цих даних може бути великою і шукати не нульові дані в масиві може займати багато часу. Якщо в даних можливо закодувати bitmap_bits біт, то кожні bitmap_bits даних кодуються бітом бітової мапи верхнього рівня. Якщо дані не нульові (тобто є хоча б один встановлений біт), то такі дані мають встановлений біт в даних бітової мапи верхнього рівня. В свою чергу для бітової мапи верхнього рівня використовується такий самий алгоритм рекурсивно. Наприклад якщо бітова мапа має 4096 біта і цілий тип має 32 біта, то $\lceil 4096 / 32 \rceil = 128$, $\lceil 128 / 32 \rceil = 4$, $\lceil 4 / 32 \rceil = 1$, тобто треба створити три рівня для цієї бітової мапи і зробити три кроки для знаходження необхідного біту. Якщо кількість даних в бітовій мапі не велика, то можливо використовувати лінійний пошук.

Для пошуку з стратегією best fit (для пошуку extent'у) шукається не конкретний біт в бітовій мапі, а декілька бітів одночасно. Для цього на кожному рівні треба створювати маску для визначення які біти перевіряти в даних бітової мапи і вибирати перший встановлений біт.

Варіант реалізації: бітові мапи

11 10 | 00 00 | 00 00 | 11 00 | 00 00 | 00 00 | 10 10 | ... нижній рівень бітової мапи

11 00 | 00 10 | 00 00 | 11 00 | ... наступний рівень

10 01 | 00 10 | ... наступний рівень

11 01 | ... наступний рівень

11 | верхній рівень

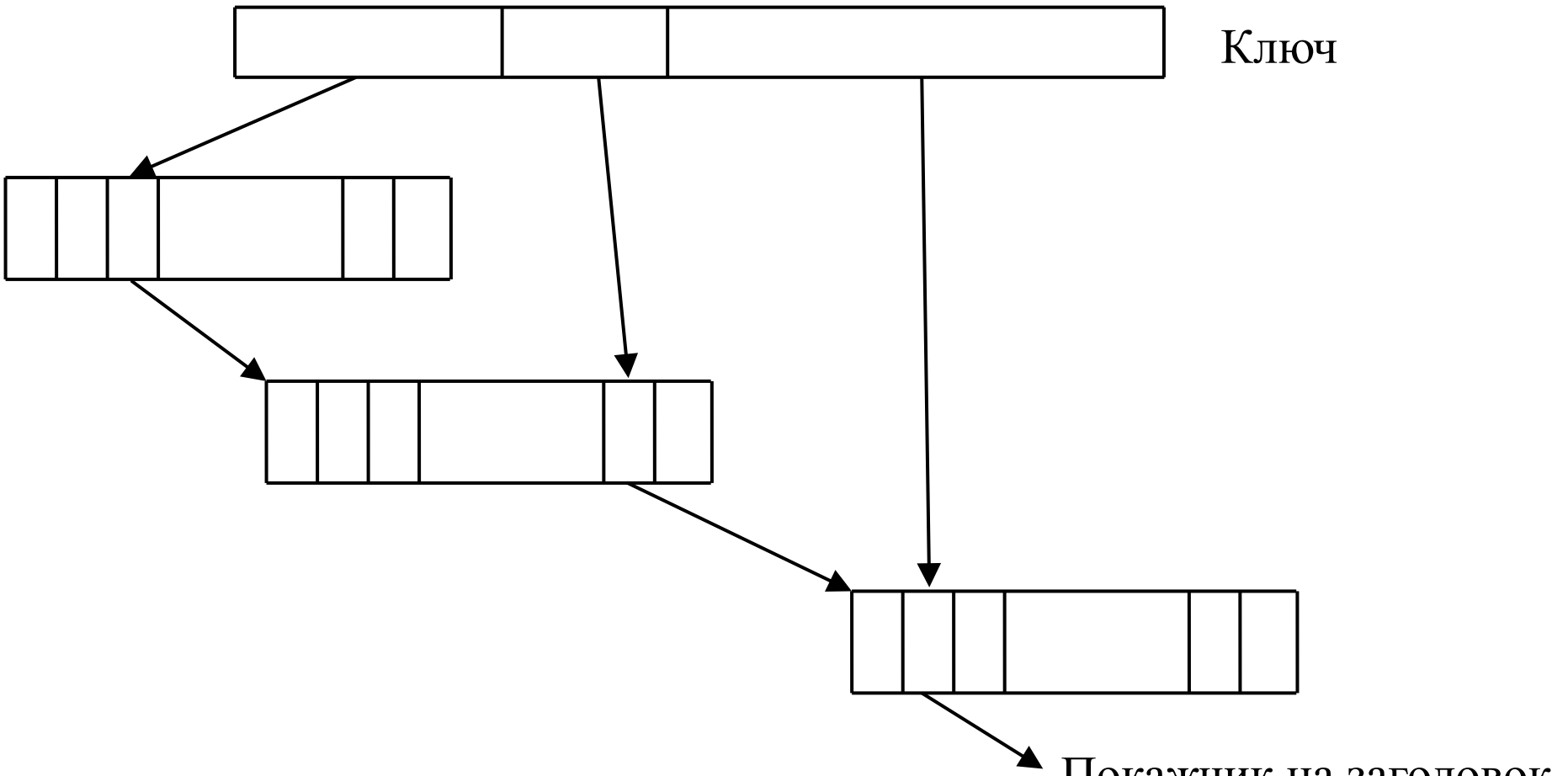
Варіант реалізації: пошук заголовків

Для пошуку заголовків slab’ів та extent’ів використовуємо Radix tree. Пам’ять для побудови дерева запитується у ядра. Адреса пам’яті, що звільнюється, конвертується в номер сторінки (це адреса початку сторінки). Далі біти в номері сторінки діляться на групи, кожна група кодує елемент в відповідному рівні дерева (деякі процесори використовують не всі біти в адресі, наприклад x86_64). Останній рівень дерева має покажчик на заголовок slab’а або extent’а. Групи бітів не обов’язково мають мати однаковий розмір. Всі сторінки кожного slab’у мають покажчики в дереві на заголовок slab’а. Кожний extent має в дереві покажчики для першої та останньої своїх сторінок на заголовок extent’а. Якщо extent належить арені і він є останнім, то покажчик для останньої сторінки не потрібен. Якщо extent не з арени, достатньо запам’ятовувати покажчик на його заголовок тільки для першої його сторінки.

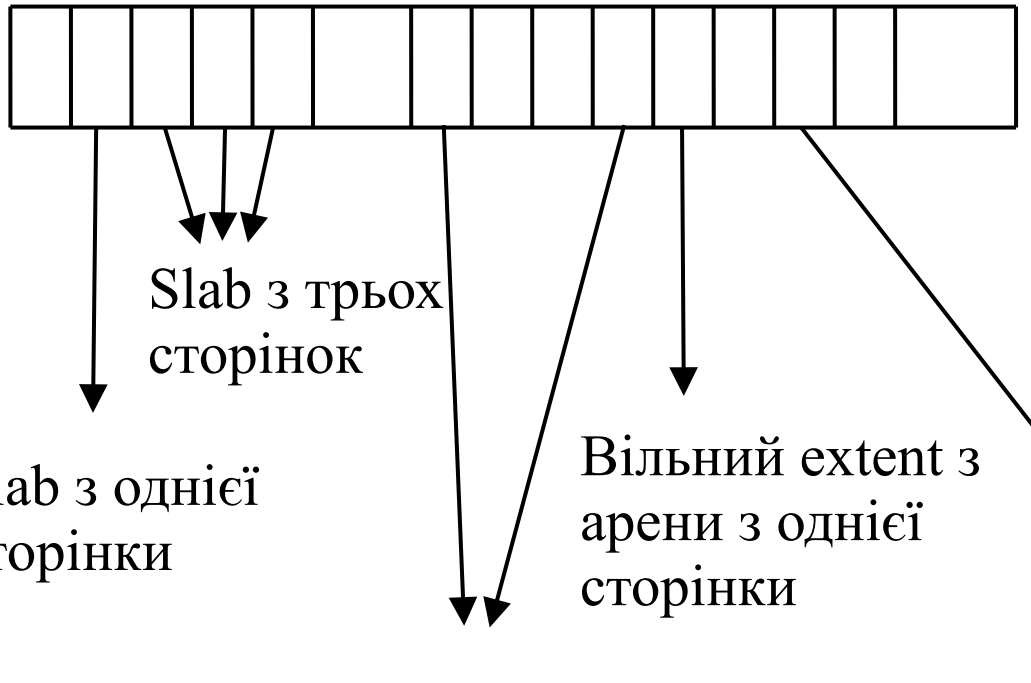
Варіант реалізації: пошук заголовків

Ці покажчики дозволяють визначати сусідні slab’и та extent’и в арені і якщо якийсь з сусідів вільний, то виконати об’єднання в більший extent. Ці покажчики також дозволяють визначити чи використовується сторінка в останньому рівні дерева, якщо сторінка не використовується, то ядро повідомляється про це або сторінка звільнюється. Можливо визначати чи використовуються сторінки на верхніх рівнях дерева, якщо сторінки не використовуються, то ядро повідомляється про це або сторінки звільнюються.

Варіант реалізації: radix tree



Варіант реалізації: radix tree



Нижній рівень дерева

Slab з трьох сторінок

Slab з однієї сторінки

Вільний extent з арени з однієї сторінки

Покажчики вказують на заголовки slab’ів або extent’ів

Зайнятий extent з арени з чотирьох сторінок

Зайнятий extent не з арени (запам’ятовується тільки початок extent’у)