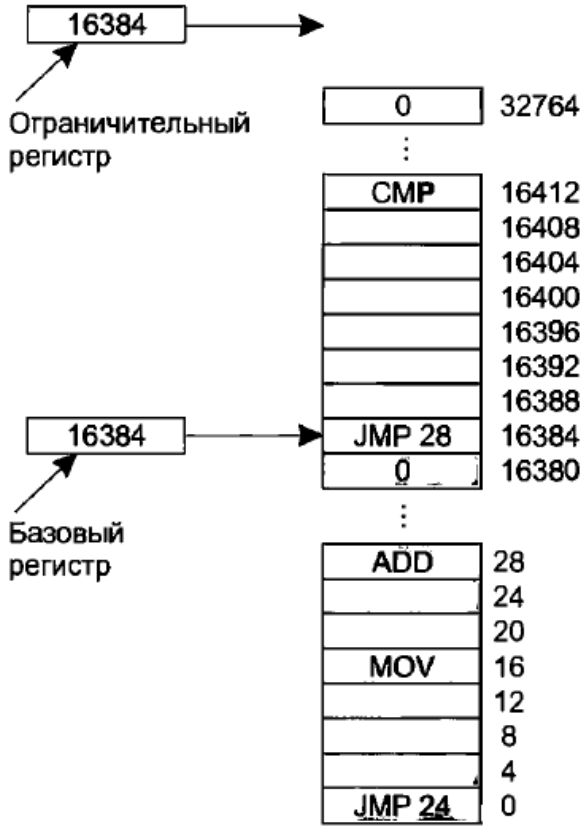
Операционные системы

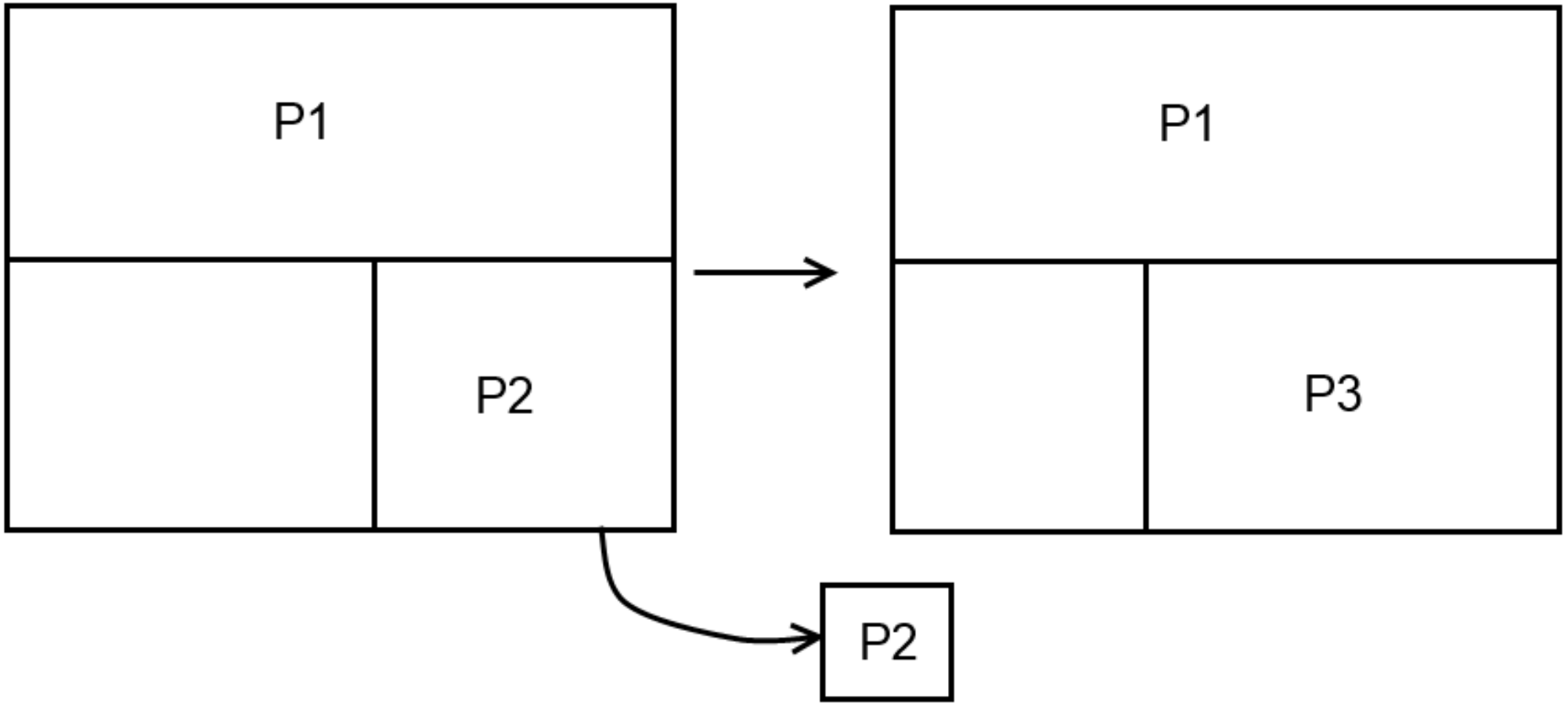
Работа с памятью. Аллокация

Память без абстракций

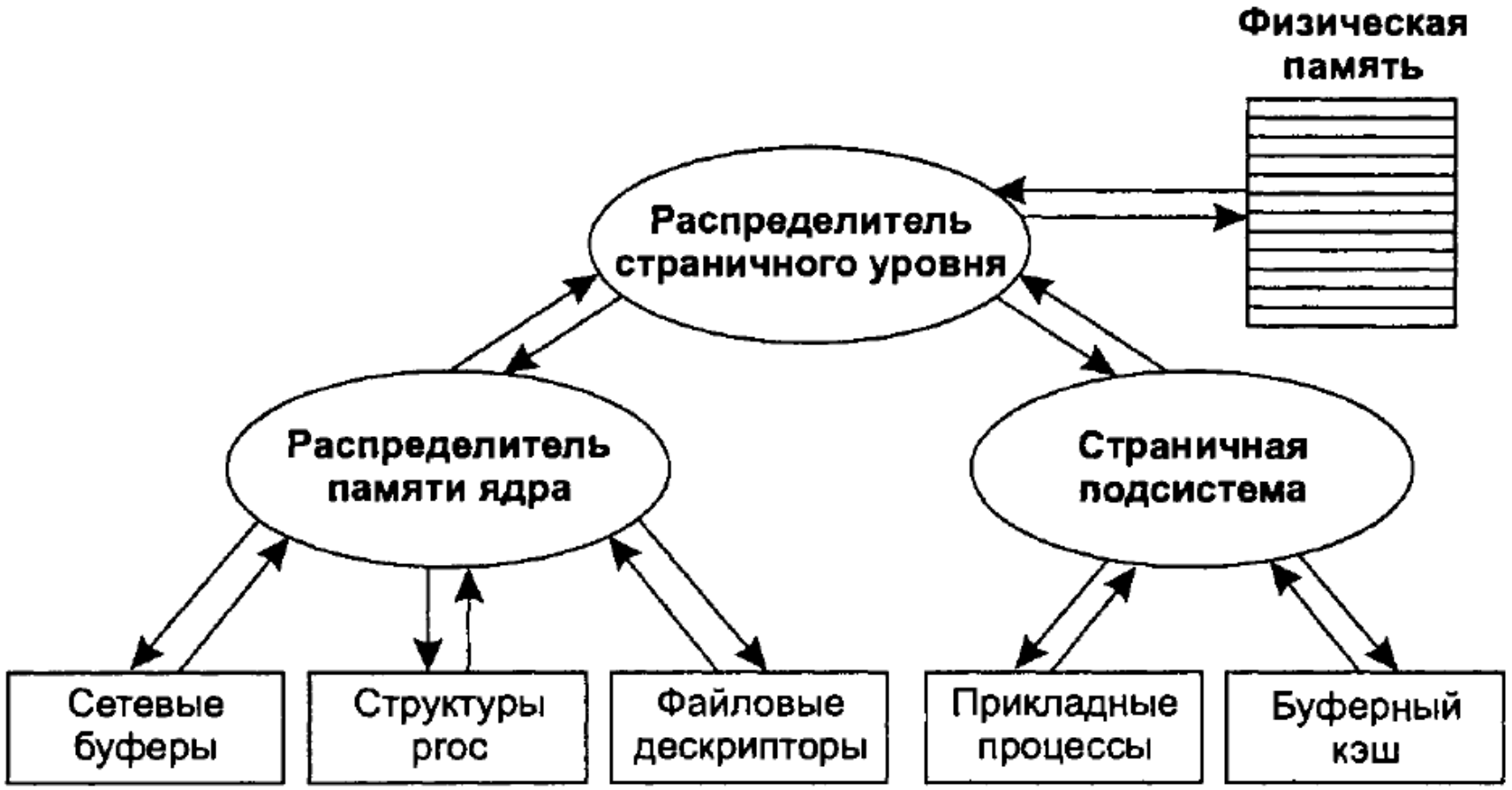
Адресные пространства (ограничительный и базовый регистры)



Свопинг



Управление памятью



Распределитель памяти ядра

1. Частое выделение/освобождение структур
2. Необходимо динамическое выделение объектов
3. Если нет памяти:
   1. Заблокировать вызывающий процесс
   2. Вернуться с ошибкой без блокировки
4. Должен отслеживать свободные части выделенного страничным аллокатором пула и уменьшать степень фрагментации

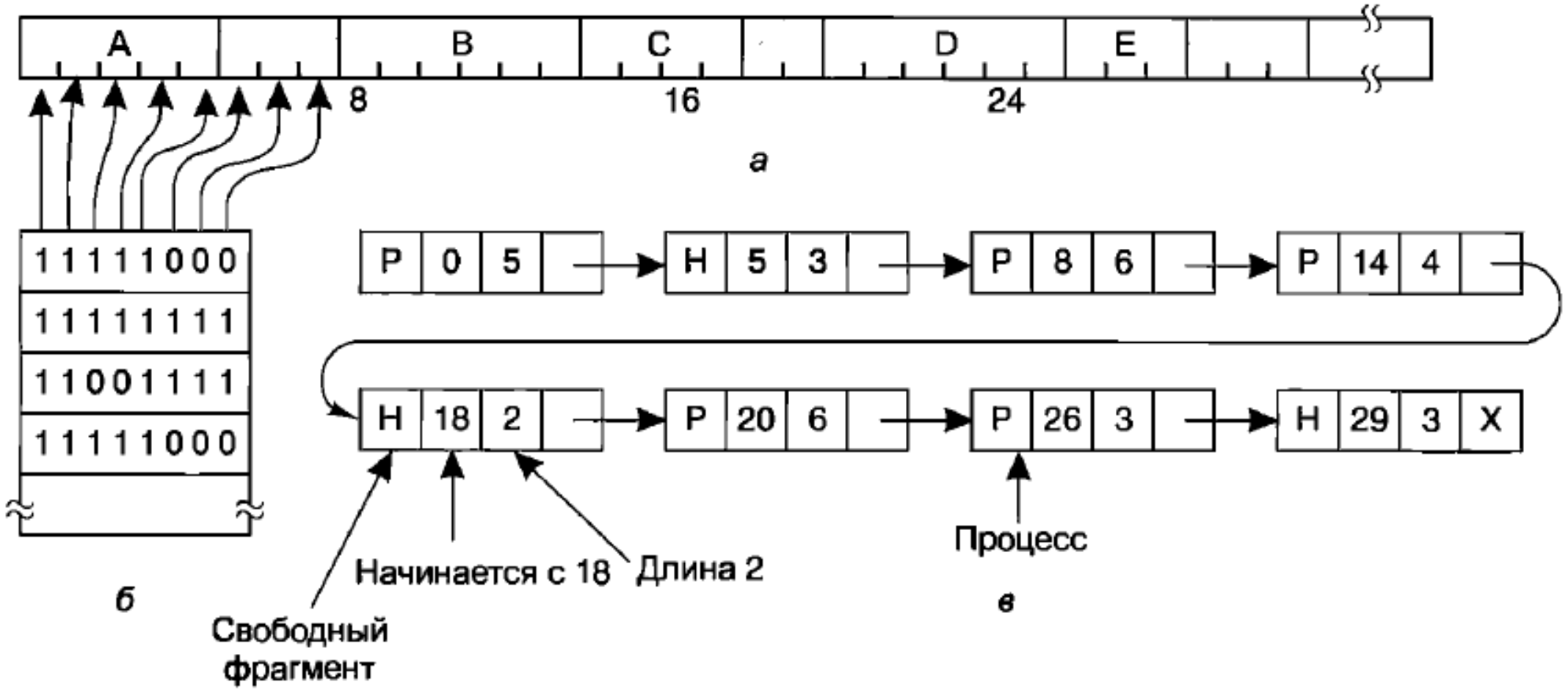
Оценка аллокаторов

1. Фактор использования
   $\frac{V_{req}}{V_{tot}}$, где V_{req} – запрашиваемая память, V_{tot} – требуемая память
2. Скорость выделения/освобождения памяти
3. Простота использования аллокатора

Способы отслеживания свободной памяти

1. Битовая матрица
2. Связанные списки

Способы отслеживания свободной памяти



Виды аллокаторов

1. Список свободных элементов
2. Выделение памяти по степени 2
3. Алгоритм Мак-Кьюзи-Кэрелса
4. Алгоритм двойников
5. Слябовый аллокатор

Список свободных элементов. Основные положения

1. Свободные блоки организуем в список
2. Блок хранит размер и ссылку на следующий свободный блок
3. Стратегии выбора свободного блока:
   a) Первое подходящее
   b) Следующее подходящее
   c) Наиболее подходящее
   d) Наименее подходящее

Алгоритм выделения/освобождения памяти

1. Выделение
   a) Находим нужный блок памяти
   b) Уменьшаем его размер на N
2. Освобождение
   a) Находим место для вставки освобожденного блока
   b) При необходимости производим слияние соседних блоков
   c) С односвязным списком сложность O(n)

Улучшения

1. Хранение размера занятого блока
   a) Если есть свободный блок длиной N+1 нельзя выделить из него блок длиной N
2. Использование двусвязного списка
   a) В занятом блоке храним: размер, тег «занят» вначале и конце блока
   b) В свободном блоке: размер, ссылки на следующий и предыдущий свободные блоки, тег «свободен» в начале и конце блока

Анализ аллокатора: «список свободных элементов»

Плюсы

- Простота
- Можно выделять ровно запрошенное количество байт
- Можно освобождать меньше, чем размер занятого блока
- Есть слияние свободных блоков

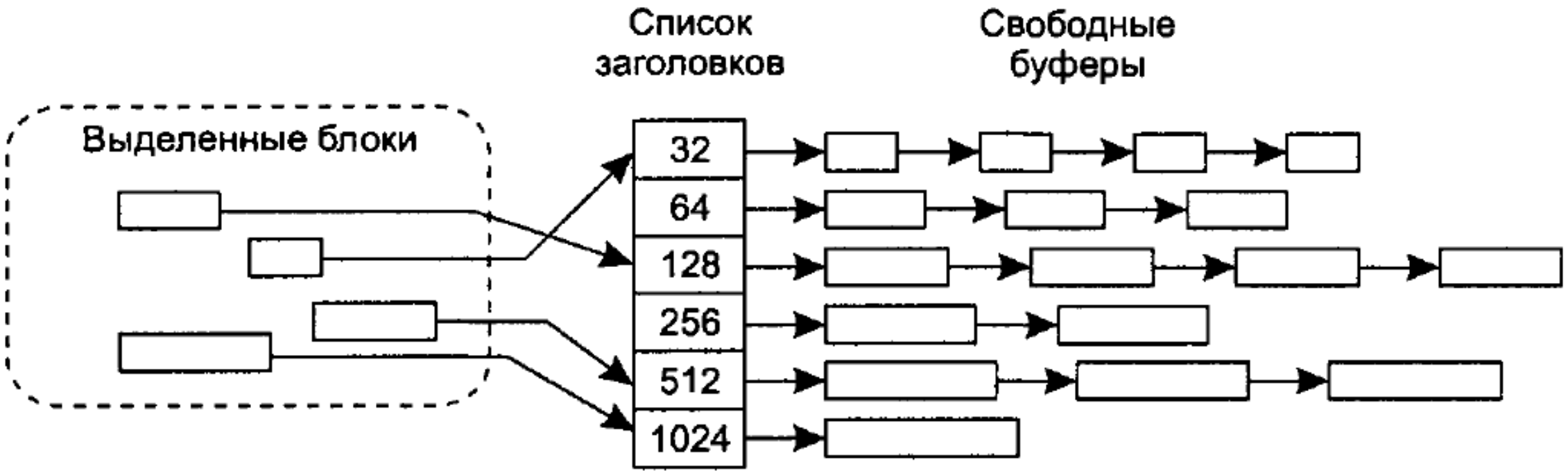
Минусы

- Большая фрагментация при длительной работе
- Для ускорения требуется значительное количество дополнительной памяти

Выделение памяти по степени 2

1. Вся память делится на списки свободных элементов равного размера
2. Если элемент свободен, то он хранит ссылку на следующий свободный элемент
3. Если элемент занят, то хранит на ссылка на голову списка, откуда он был взят
4. При освобождение нужно просто добавить элемент к голове указанного списка
5. При выделении выбирается список $K = [\log_2(N)]$

Блоки по 2^n: пример



Блоки по 2^n: анализ

Плюсы

- Удобный интерфейс для free()
- Быстрый поиск ресурса для выделения

Минусы

- Использование лишней памяти
- В занятой памяти хранится указатель на голову списка
- Нельзя освобождать буферы частично
- Нет слияния буфера и возвращение памяти страничному аллокатору

Алгоритм Мак-Кьюзика-Кэрелса

1. Является улучшенной версией алгоритма «Блоки по 2^n»
2. Храним массив страниц
   a) Страница может быть свободной и хранить ссылку на следующую свободную страницу
   b) Может быть разбита на блоки - в массиве содержится размер блока, на которые разбита страница
   c) Указатель на то, что страница является частью крупного блока памяти
3. Храним массив с указателями на списки свободных блоков
   - Блоки имеют размер равный степени 2
   - Блоки внутри одной страницы имеют одинаковый размер

Выделение/освобождение памяти

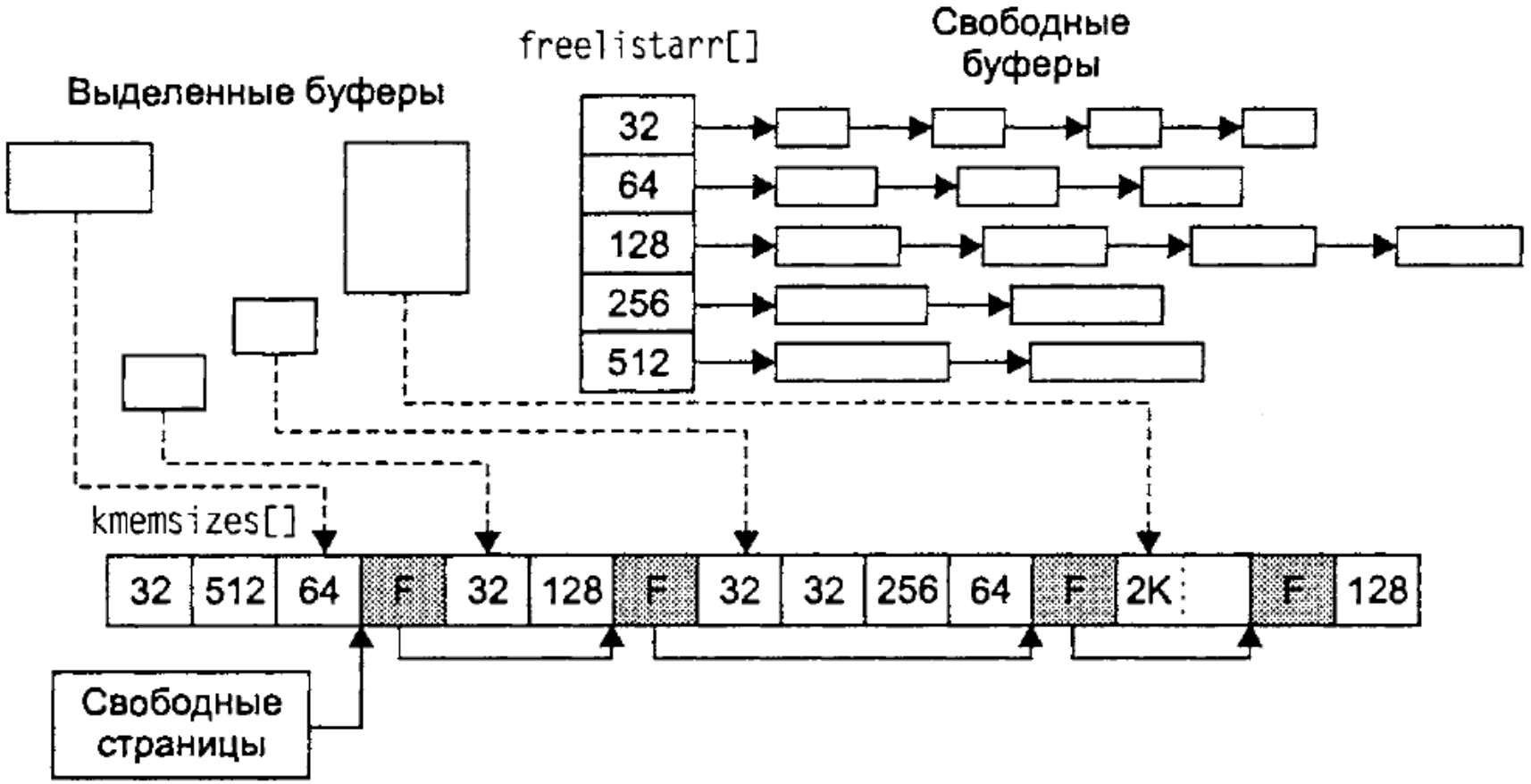
Выделение

- При запросе памяти округляем запрос до степени 2
- При выделении блока в блоке не требуется хранить дополнительную информацию
- Если нет блока нужного размера, то выделяем новую страницу, которую разбиваем на блоки данного размера

Освобождение

- Для освобождения не требуется указывать размер блока

Илюстрация


freelistarr[]
Свободные буферы
Выделенные буферы
32
64
128
256
512
kmemsizes[]
32 512 64 F 32 128 F 32 32 256 64 F 2K F 128
Свободные страницы

Анализ

Плюсы

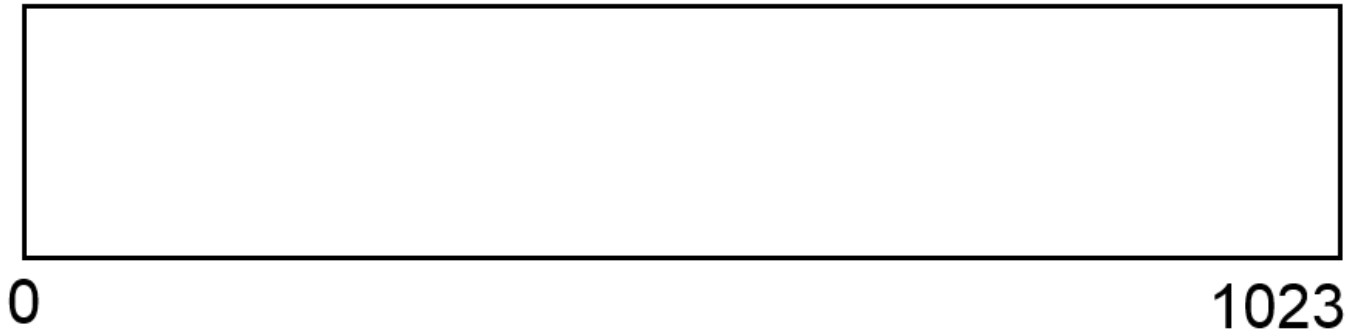
- Блоки можно выделять нужного размера
- Блоки формируются по требованиям системы
- Функция free не требует размера освобождаемого блока

Минусы

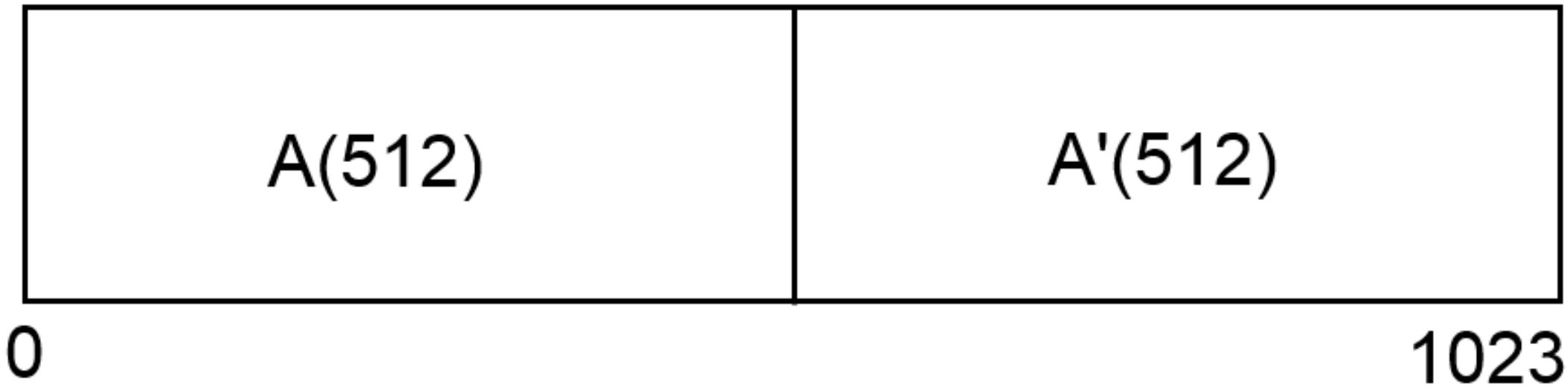
- Нет возможности возвращать страницы обратно страничному аллокатору
- Нет возможности слияния блоков
- Возможны потери при неравномерном распределении запрашиваемых размеров памяти

Алгоритм двойников. Пример.

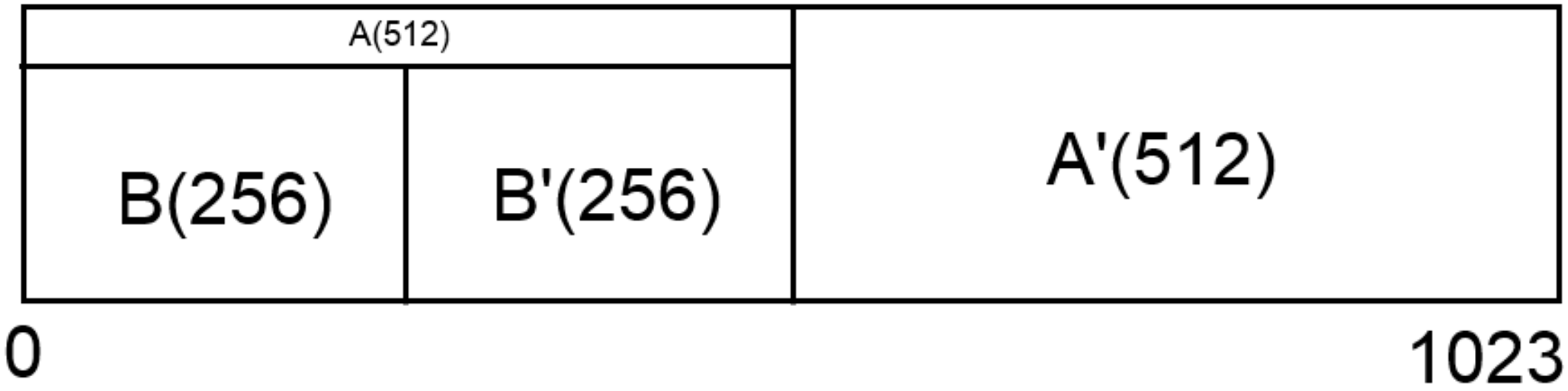
1. Запрос на 70 байт. Округляем до 128
2. Есть свободная страница размера 2^m



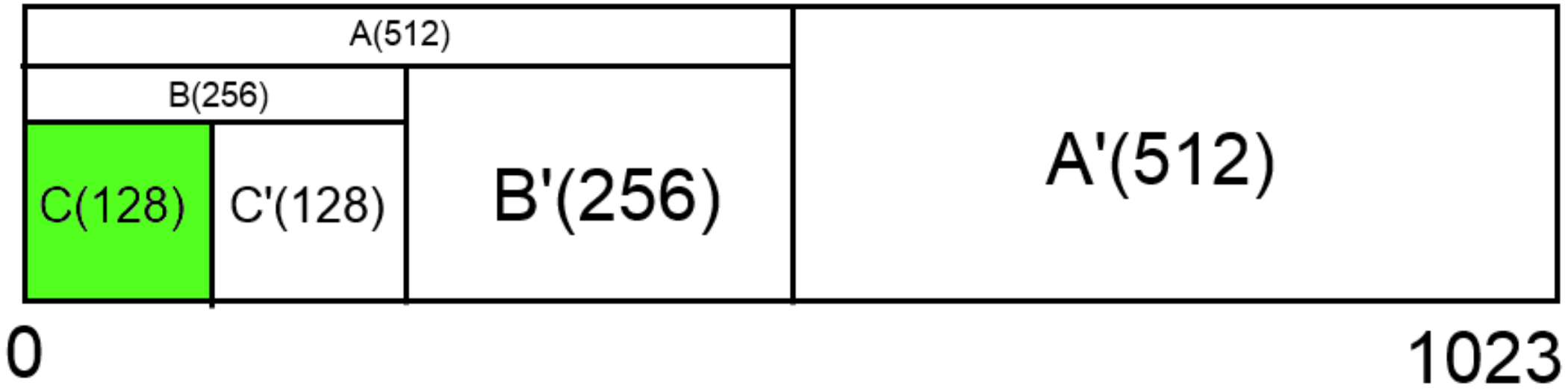
Алгоритм двойников. Пример. Шаг 1



Алгоритм двойников. Пример. Шаг 2



Алгоритм двойников. Пример. Шаг 3



Основные положения алгоритма

1. Бьем свободный пул до тех пор пока не получаем блок нужного размера
2. Части разделенного блока называют двойниками
3. В каждом блоке надо хранить тег «свободен»/ «занят»
4. Если известен адрес блока и его размер, то известен адрес двойника
   Адрес: 10110000, размер: 16 -> адрес двойника: 10100000
5. Если освобождается блок, и его двойник оказывается свободен, то двойников сливают. Полученный блок пытаются слить с его двойником. Блок, который не удалось слить добавляют в список свободных блоков
6. Свободные блоки хранятся в двусвязном списке

Анализ и улучшения

Плюсы

- Объединение смежных блоков
- Легко обмениваться со страничным аллокатором

Минусы

- При частом выделении/освобождение падает производительность из-за слияния
- При удалении необходимо указывать размер блока
- Можно выделить только блок кратный степени 2

Улучшения

- N = A + L + G
- slack = N – 2*L – G - отложенное слияние, только если slack < 2

Слябовый аллокатор

1. Цикл жизни объекта: создание – использование – уничтожение – освобождение
2. Возможно использовать повторно объекты без инициализации
3. Объекты ядра делятся на группы-кэши, кэш делится на слябы (слябы представляют собой одну или последовательность нескольких страниц), которые содержат объекты. При создании пытаемся найти свободный объект в слябе. При удалении помечаем объект в слябе, как свободный.

Иллюстрация