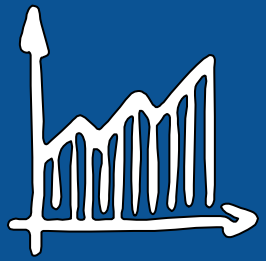
Алгоритмы и Алгоритмические Языки

Семинар #11:

1. Динамическое выделение памяти.
2. Структура данных «Стек» и структуры в языке C.
3. Анализ стратегий выделения памяти.

05.11.2024

Динамическое выделение памяти



Структура памяти программы

Стек (`RW-`) – для локальных переменных и стека вызовов, управляется компилятором.

Куча (`RW-`) – управляется программистом.

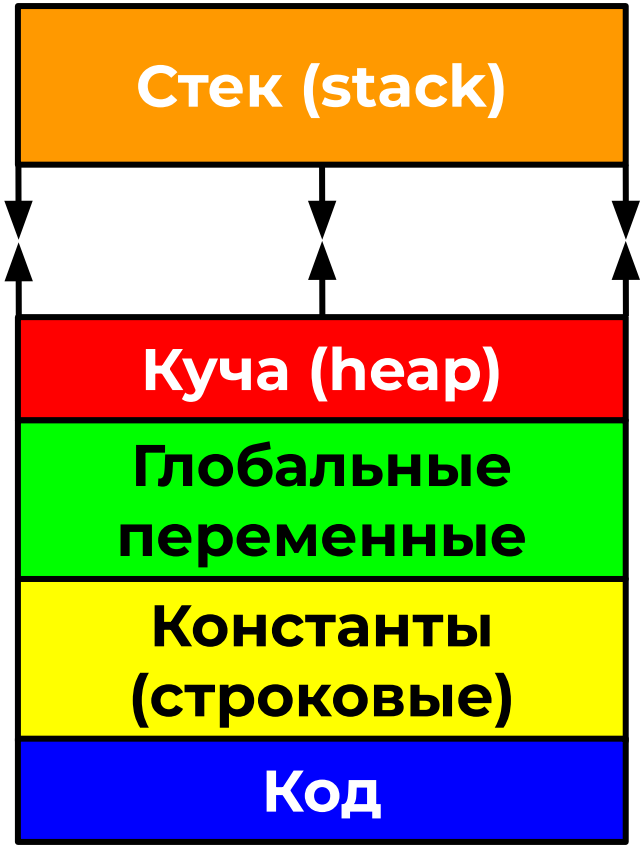
Глобальные переменные (`RW-`).

Константы и строковые литералы (`R--`).

Исполняемый код (`R-X`).

Код стандартной библиотеки.

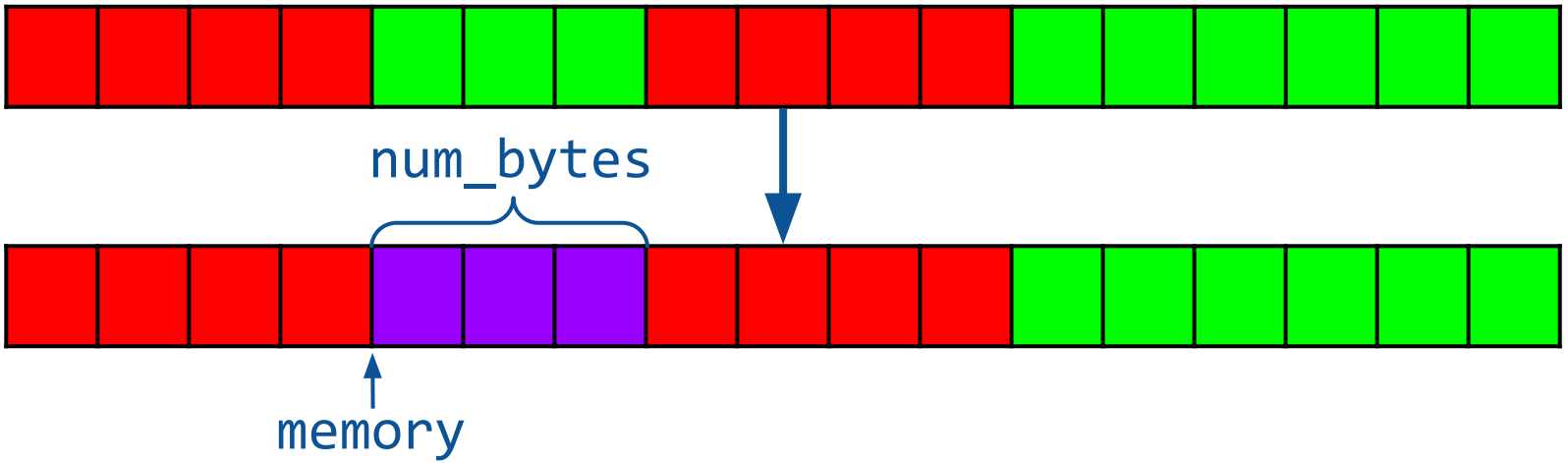
`RWX` – разрешение на запись (**`Write`**), чтение (**`Read`**), исполнение (**`eXecute`**).



Динамическое выделение памяти

Функция **malloc** выделяет свободную память на куче:

```
void* memory = malloc(num_bytes);
```

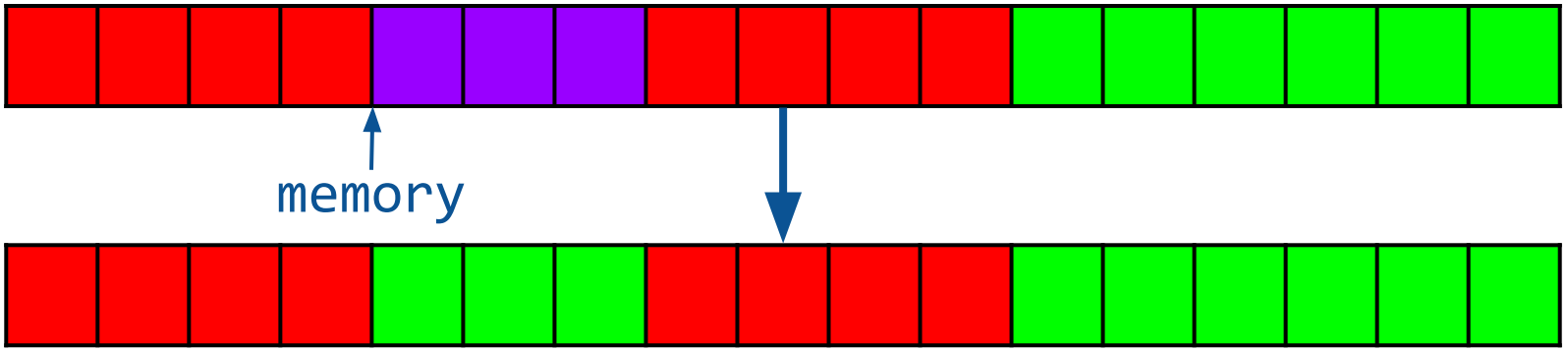


Как **malloc** находит свободную память?

Динамическое выделение памяти

Функция **free** освобождает ранее выделенную память:

```
free(memory);
```

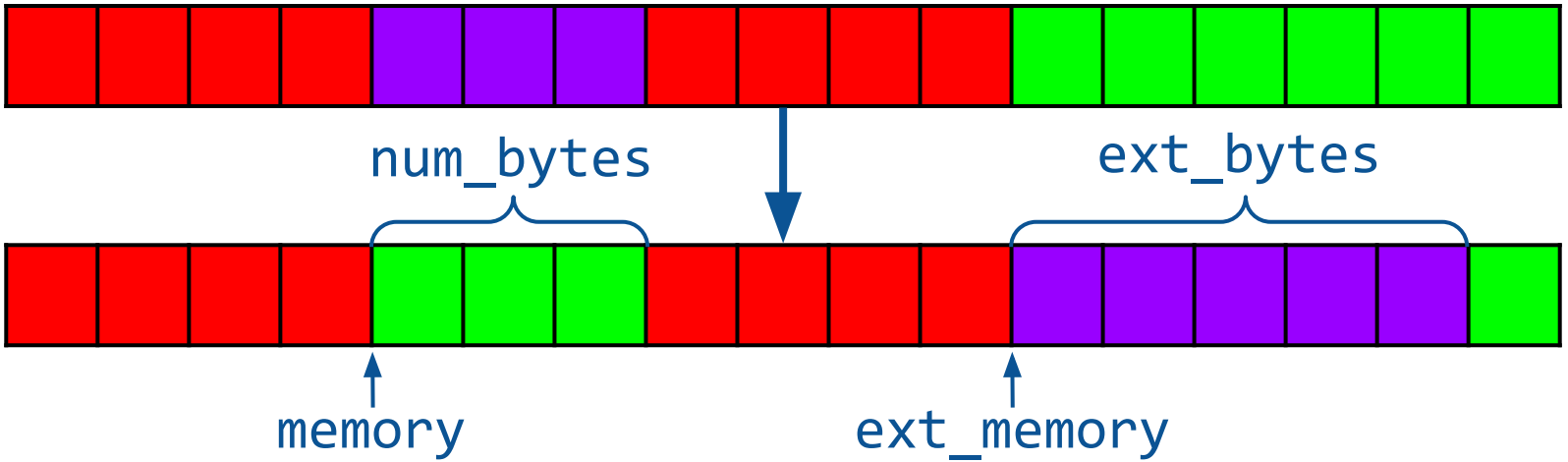


Как **free** узнаёт, сколько памяти освобождать?

Динамическое выделение памяти

Функция **realloc** изменяет размер выделенного блока памяти:

```
void* ext_memory = realloc(memory, ext_bytes);
```

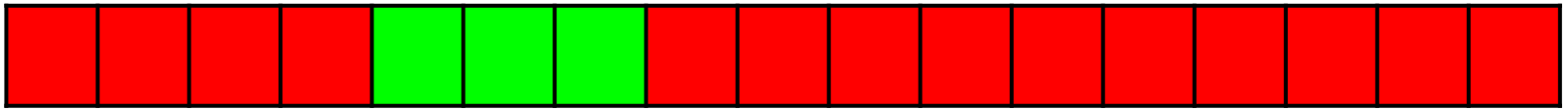


Чем **realloc** может быть лучше **malloc+memcpy+free**?

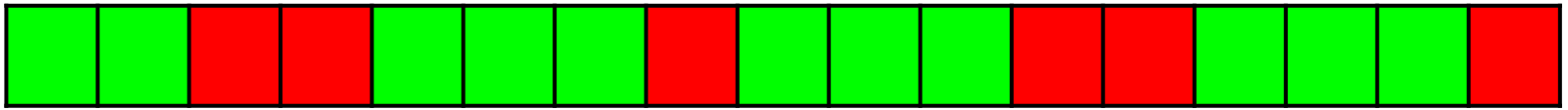
Ошибки выделения памяти

```
void* memory = malloc(num_bytes);
if (memory == NULL)
{
    printf("Unable to allocate memory\n");
    exit(EXIT_FAILURE);
}
```

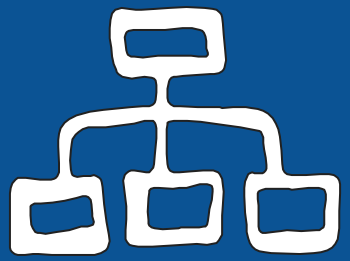
Нехватка памяти (для num_bytes = 5):



Фрагментация памяти (для num_bytes = 5):

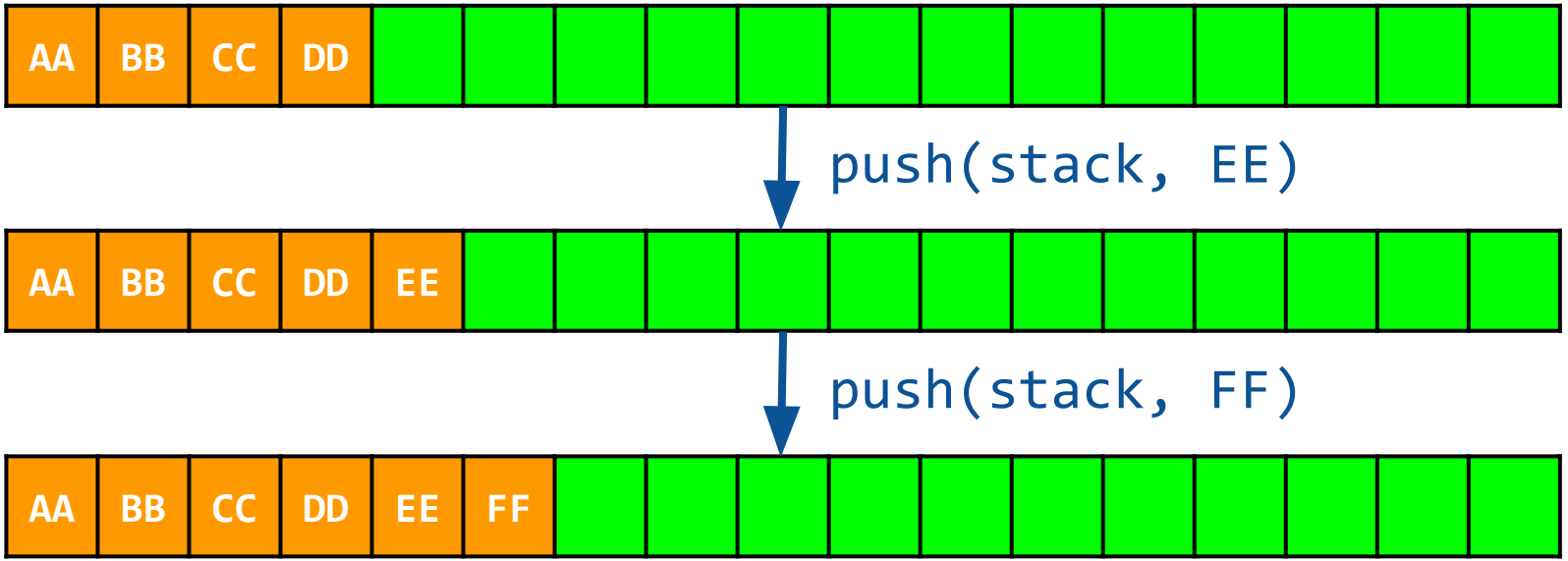


Структура данных «Стек» и структуры в языке С



Структура данных «Стек»

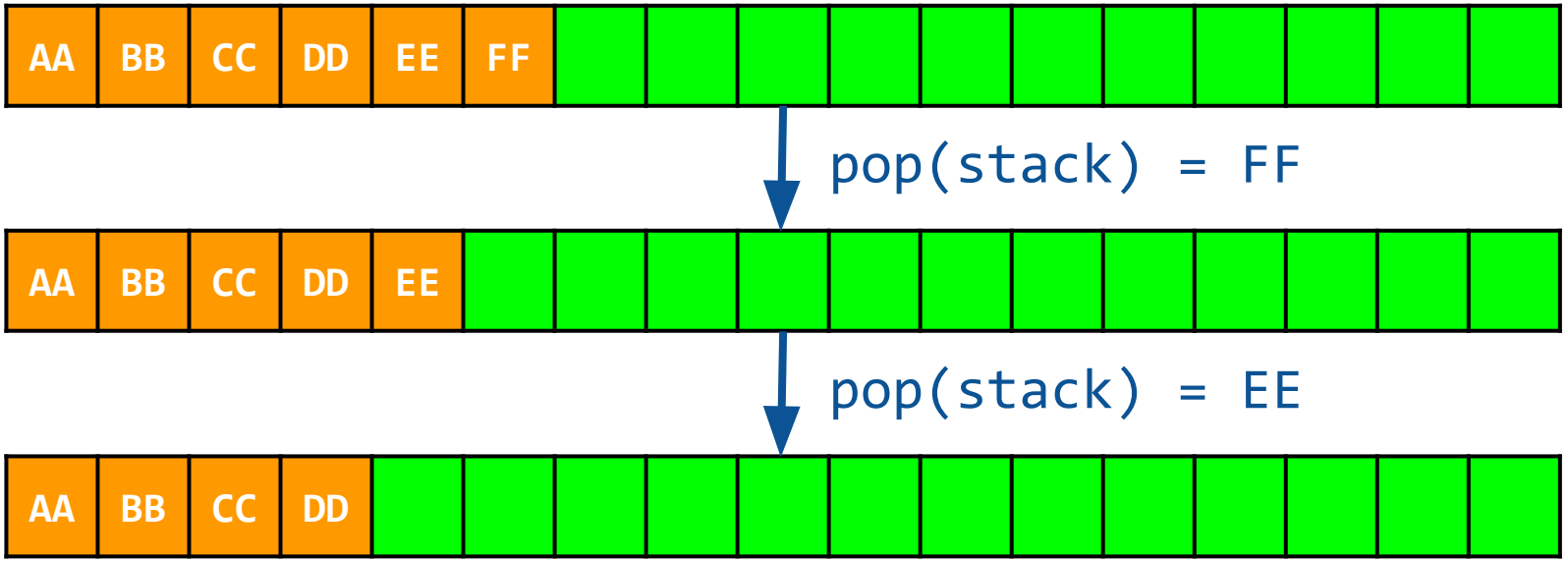
Добавление элемента на вершину стека (операция **push**):



Структура данных **LIFO (“Last In First Out”)**

Структура данных «Стек»

Удаление элемента с вершины стека (операция **pop**):



Структура данных **LIFO (“Last In First Out”)**

Синтаксис структур

Объявление структуры

```
struct Stack
{
    // Массив с элементами стека
    data_t* array;

    // Количество элементов в стеке
    size_t size;
};
```

Инициализация структуры

```
struct Stack stack = { NULL, 0 };
```

Доступ к полям структуры

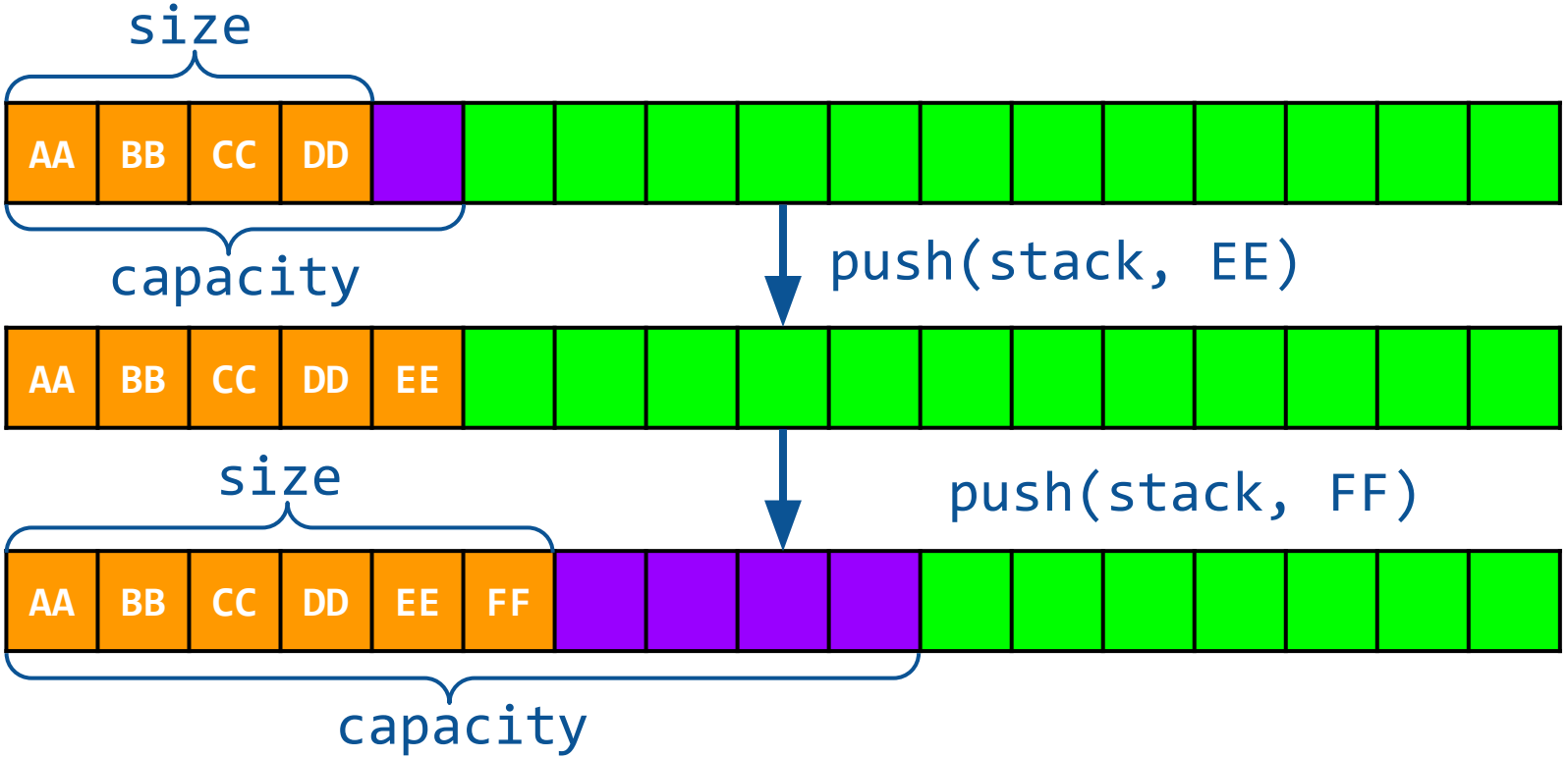
```
stack.array = malloc(10U * sizeof(data_t));
stack.size  = 10U;
```

Передача структуры в функцию по указателю и доступ к её полям

```
void stack_push(struct Stack* stack, data_t element)
{
    // Перевыделяем память под стек
    stack->array = realloc(stack->array, (stack->size + 1) * sizeof(data_t));
    // Добавляем элемент в стек
    stack->array[stack->size] = element;
    // Увеличиваем счётчик элементов в стеке
    stack->size += 1U;
}
```

Размер и ёмкость стека

В целях оптимизации можно выполнять **realloc** не каждый раз.



Коды возврата операций

```c
// Тип данных, задающий код возврата
typedef enum
{
    // Операция выполнена без ошибок
    STACK_OK    = 0,
    // Операция не выполнена, т.к. в стеке нет элементов
    STACK_EMPTY = 1,
    // Операция не выполнена, т.к. для её выполнения не хватает памяти
    STACK_NOMEM = 2,
    // Операция не выполнена, т.к. аргументы операции некорректны
    STACK_INVAL = 3,
} StackRetCode;
```

```c
StackRetCode stack_init(struct Stack* stack, StackResizePolicy policy);
StackRetCode stack_free(struct Stack* stack);
StackRetCode stack_push(struct Stack* stack, data_t  element);
StackRetCode stack_pop (struct Stack* stack, data_t* element);
```

Что может вернуть каждая из операций?

Инвариант структуры данных

```
struct Stack
{
    // Массив с элементами стека
    data_t* array;
    // Количество элементов в стеке
    size_t size;
    // Размер выделенной памяти (кол-во элементов ти
    size_t capacity;
};
```

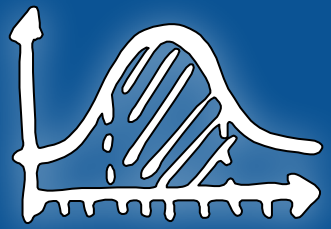
```
// Проверяем состояние стека
StackRetCode ret = stack_ok(stack);
if (ret != STACK_OK)
{
    return ret;
}
```

```
StackRetCode stack_ok(struct Stack* stack) {
    if (stack == NULL) {
        return STACK_INVAL;
    }
    if (stack->capacity == 0 || stack->array == NULL) {
        return STACK_INVAL;
    }
    if (stack->size > stack->capacity) {
        return STACK_INVAL;
    }
    return STACK_OK;
}
```

Проверка инварианта при каждой операции позволит выявить случайные записи в структуру

HSE University

Анализ стратегий выделения памяти

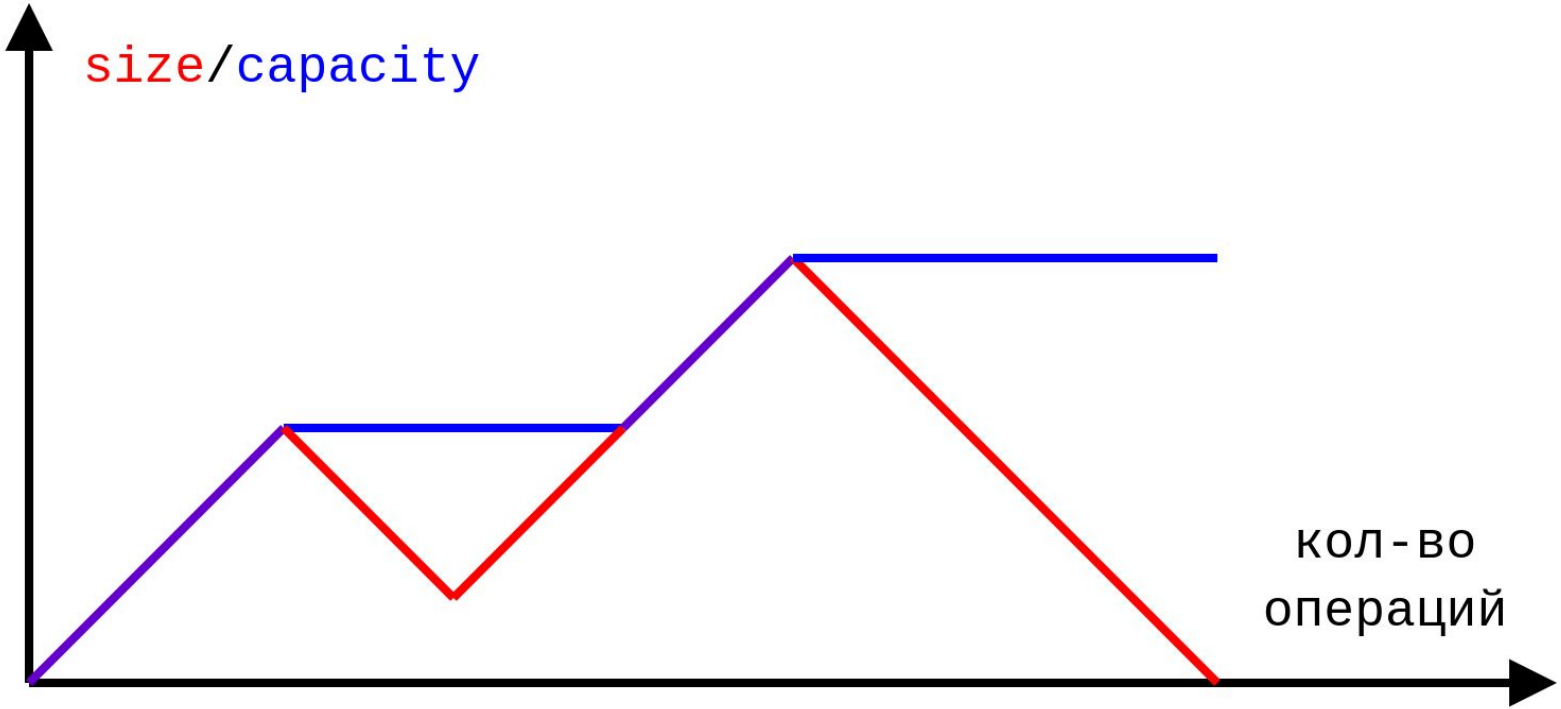


Два подхода к выделению памяти

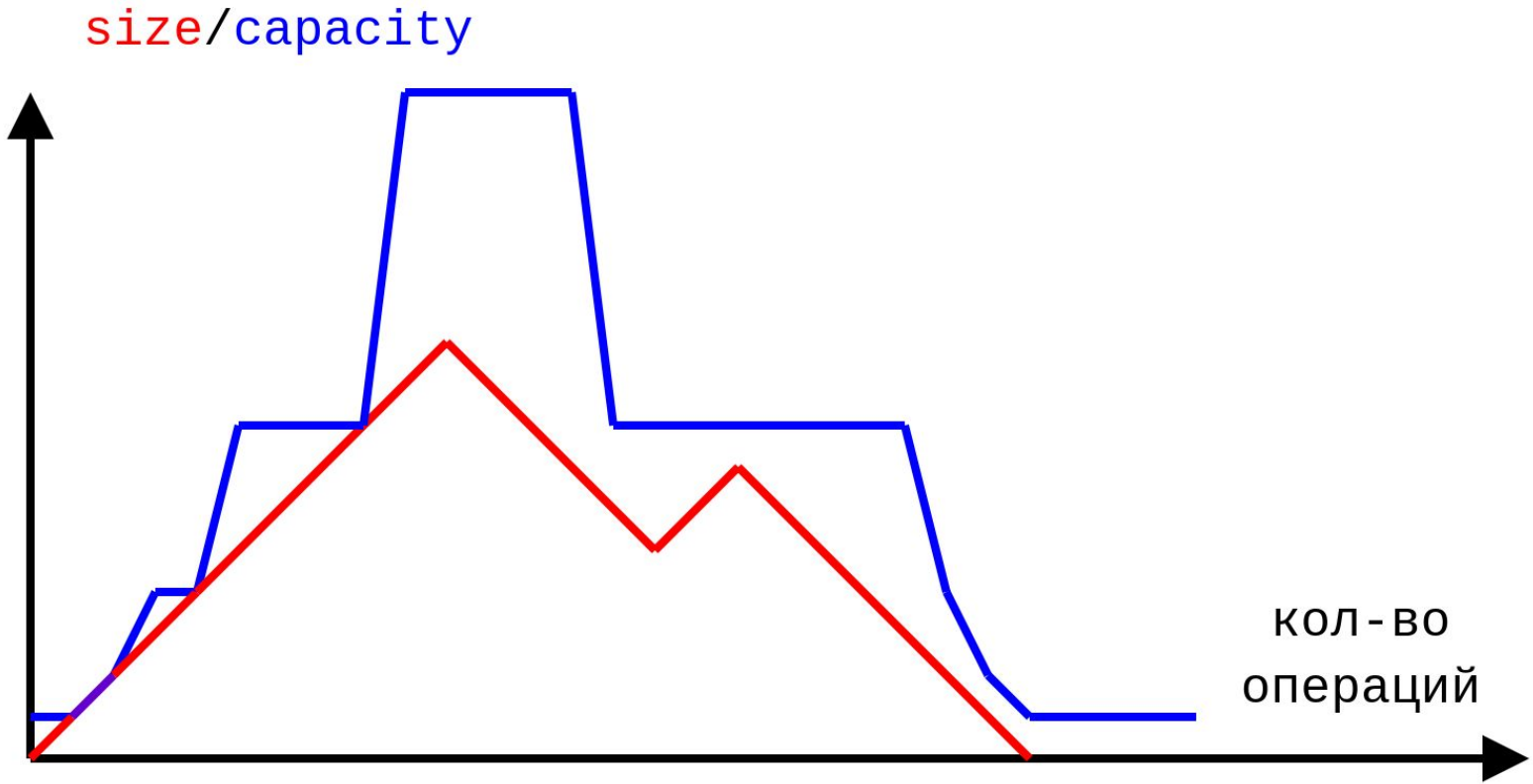
```
// Определяем необходимость увеличения ёмкости стека
if (stack->size == stack->capacity)
{
    // Ёмкость увеличивается на 1 при каждом добавлении элемента
    new_capacity = stack->capacity + 1U;
    // Производим перевыделение памяти
    stack_resize(stack, new_capacity);
}
```

```
// Определяем необходимость увеличения ёмкости стека
if (stack->size == stack->capacity)
{
    // Ёмкость стека увеличивается по степеням двойки
    new_capacity = (stack->size == 0U)? 1U : (2U * stack->capacity);
    // Производим перевыделение памяти
    stack_resize(stack, new_capacity);
}
```

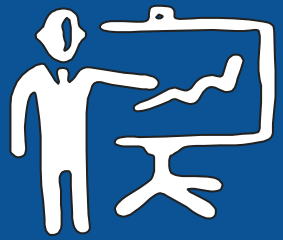
Линейное выделение памяти



Экспоненциальная реаллокация



Вопросы?



Красивые иконки взяты с сайта handdrawngoods.com