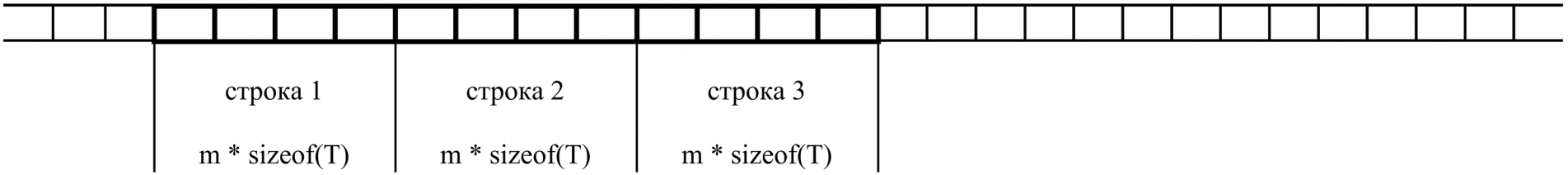
Динамические матрицы

Матрица как одномерный массив

n = 3 – количество строк

m = 4 – количество столбцов

T – тип элементов матрицы



k	0	1	2	3	4	5	6	7	8	9	10	11
i;j	0;0	0;1	0;2	0;3	1;0	1;1	1;2	1;3	2;0	2;1	2;2	2;3

a[i][j] ⇔ a[k], k = i * m + j

Матрица как одномерный массив

```
double *data;
int n = 3, m = 2;

data = malloc(n * m * sizeof(double));
if (data)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            // Обращение к элементу i, j
            data[i*m+j] = 0.0;

    free(data);
}
```

Матрица как одномерный массив

- Преимущества:
  - Простота выделения и освобождения памяти.
  - Возможность использовать как одномерный массив.

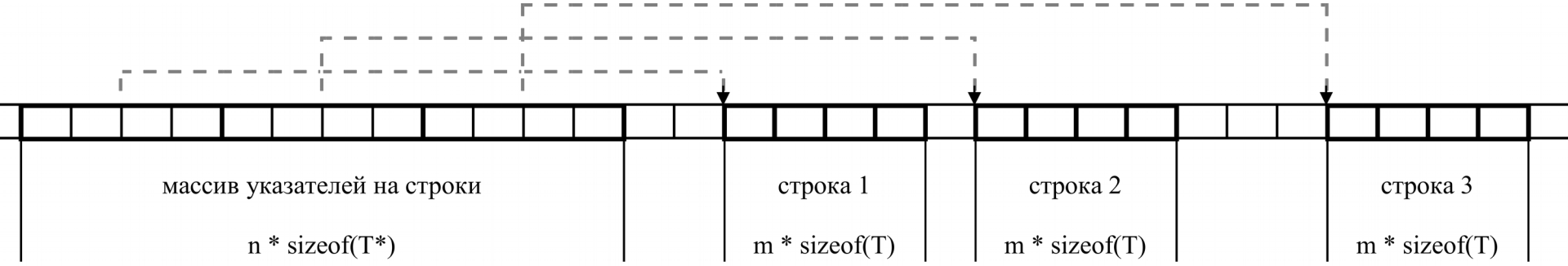
- Недостатки:
  - Средство проверки работы с памятью (СПРП), например Doctor Memory, не может отследить выход за пределы строки.
  - Нужно писать i * m + j, где m – число столбцов.

Матрица как массив указателей

n = 3 – количество строк

m = 4 – количество столбцов

T – тип элементов матрицы



Матрица как массив указателей

Алгоритм выделения памяти

Вход: количество строк (n) и количество столбцов (m)

Выход: указатель на массив строк матрицы (p)

- Выделить память под массив указателей (p)
- Обработать ошибку выделения памяти
- В цикле по количеству строк матрицы (0 <= i < n)
  - Выделить память под i-ую строку матрицы (q)
  - Обработать ошибку выделения памяти
  - p[i]=q

Матрица как массив указателей

Алгоритм освобождения памяти

Вход: указатель на массив строк матрицы (p) и количество строк (n)

- В цикле по количеству строк матрицы (0 <= i < n)
  - Освободить память из-под i-ой строки матрицы
- Освободить память из-под массива указателей (p)

Матрица как массив указателей

```
void free_matrix_rows(double **data, int n);

double** allocate_matrix_rows(int n, int m)
{
    double **data = calloc(n, sizeof(double*));
    if (!data)
        return NULL;

    for (int i = 0; i < n; i++)
    {
        data[i] = malloc(m * sizeof(double));
```

Матрица как массив указателей

```
            if (!data[i])
            {
                free_matrix_rows(data, n);

                return NULL;
            }
        }

        return data;
    }
```

Матрица как массив указателей

```
void free_matrix_rows(double **data, int n)
{
    for (int i = 0; i < n; i++)
       // free можно передать NULL
       free(data[i]);

    free(data);
}
```

Матрица как массив указателей

- Преимущества:
  - Возможность обмена строки через обмен указателей.
  - СПРП может отследить выход за пределы строки.

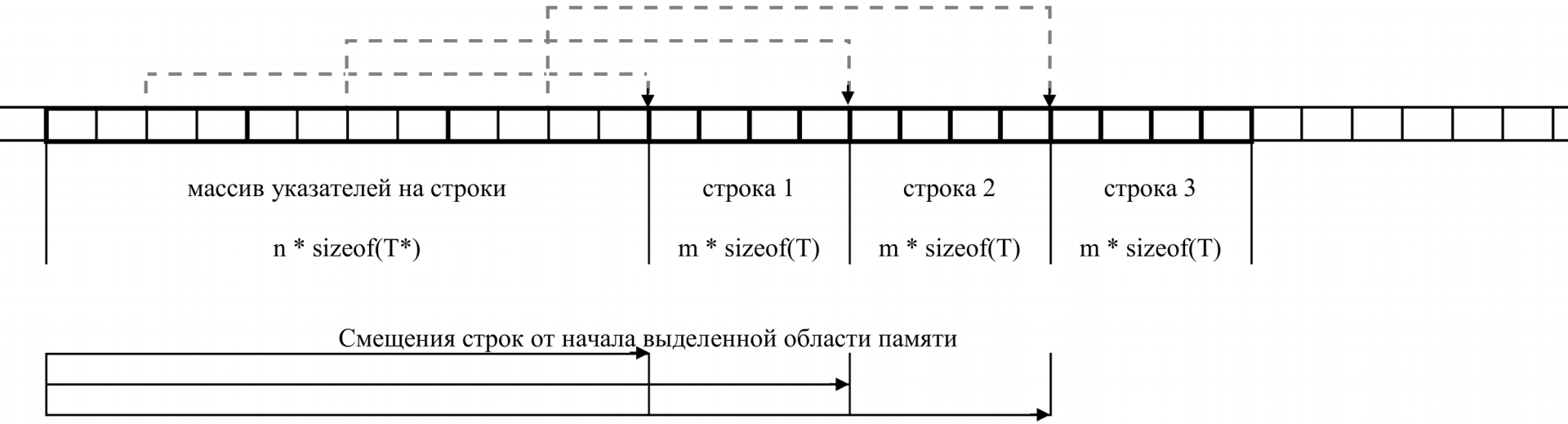
- Недостатки:
  - Сложность выделения и освобождения памяти.
  - Память под матрицу "не лежит" одним куском.

Объединение подходов

n = 3 – количество строк

m = 4 – количество столбцов

T – тип элементов матрицы



Объединение подходов

Алгоритм выделения памяти

Вход: количество строк (n) и количество столбцов (m)

Выход: указатель на массив строк матрицы (p)

- Выделить память под массив указателей на строки и элементы матрицы (p)
- Обработать ошибку выделения памяти
- В цикле по количеству строк матрицы (0 <= i < n)
  - Вычислить адрес i-ой строки матрицы (q)
  - p[i]=q

Объединение подходов

```
double** allocate_matrix_solid(int n, int m)
{
    double **data = malloc(n * sizeof(double*) +
                           n * m * sizeof(double));
    if (!data)
        return NULL;

    for (int i = 0; i < n; i++)
        data[i] = (double*)((char*) data +
                            n * sizeof(double*) +
                               i * m * sizeof(double));
    return data;
}
```

Объединение подходов

Преимущества:

- Простота выделения и освобождения памяти.
- Возможность использовать как одномерный массив.
- Перестановка строк через обмен указателей.

Недостатки:

- Сложность начальной инициализации.
- СРПР не может отследить выход за пределы строки.