ALOCAÇÃO DINÂMICA DE MEMÓRIA

Prof. Muriel Mazzetto

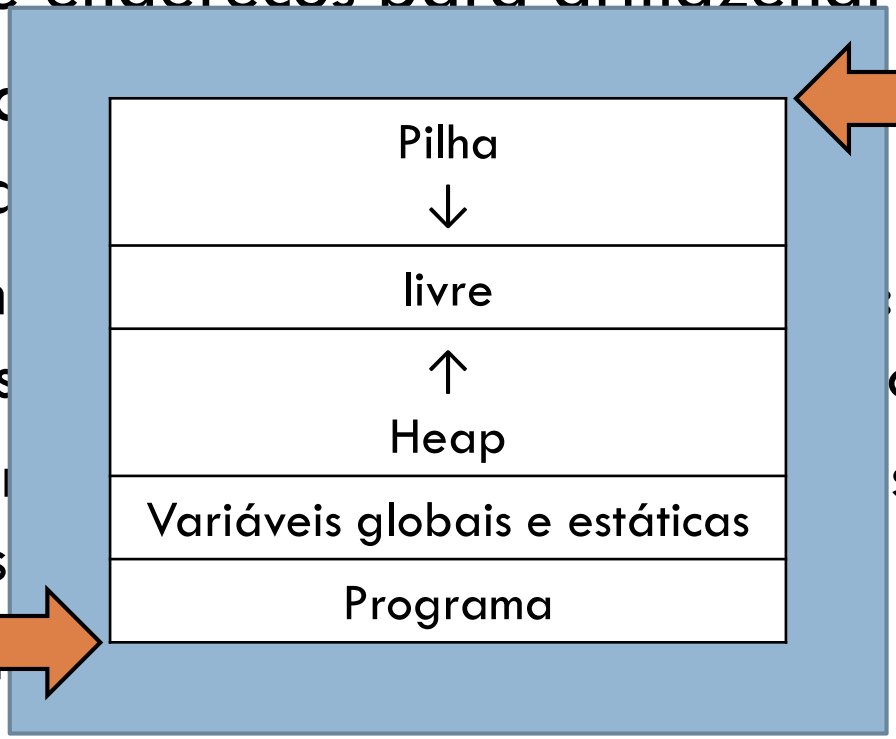
Algoritmos 2

Memória Principal

- Região de endereços para armazenar bytes.
- Subdividido para evitar conflitos durante execução de programas:
  - Programas, variáveis globais e estáticas: armazenar variáveis definidas durante programação;
  - Pilha: armazenamento de variáveis locais, não estáticas;
  - Heap: memória livre para alocação.

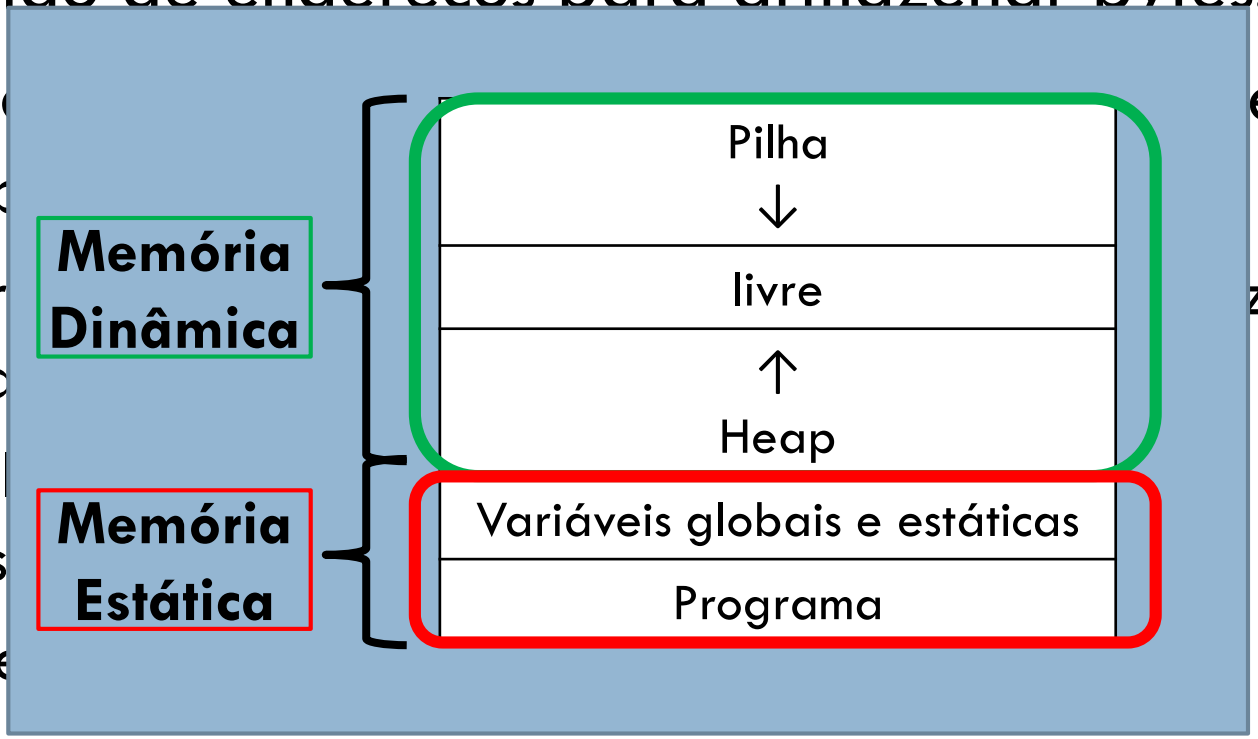
Memória Principal

- Região de endereços para armazenar bytes.
- Subdividi… …ução de progra…
  - Program… …armazenar variáveis… …o;
  - Pilha: ar… …s, não estáticas…
  - Heap: …



Memória Principal

- Região de endereços para armazenar bytes.
- Sub… …ecução de p…
  - Pr… …zenar va…
  - Pi… es…
  - He…



Alocação Estática

- Variáveis globais (e estáticas):
  - Espaço reservado existe enquanto o **programa** estiver executando.
- Variáveis locais:
  - Espaço reservado existe enquanto a **função** estiver executando. Liberado ao final da execução (Pilha).

Alocação Estática

- Variáveis globais (e estáticas):
  - Espaço reservado existe enquanto o **programa** estiver executando.
- Variáveis locais:
  - Espaço reservado existe enquanto a **função** estiver executando. Liberado ao final da execução (Pilha).
- Variáveis unitárias ou vetores.

Alocação Estática

- Variáveis globais (e estáticas):
  - Espaço reservado existe enquanto o **programa** estiver executando.
- Variáveis locais:
  - Espaço reservado existe enquanto a **função** estiver executando. Liberado ao final da execução (Pilha).
- Variáveis unitárias ou vetores.
- **Problemas:**
  - Análise de memória necessária durante programação;
  - Disponibilidade de faixa de memória.

Alocação Dinâmica

- Espaço de memória requisitado em tempo de execução.

Alocação Dinâmica

- Espaço de memória requisitado em tempo de execução.
- Utiliza funções de sistema para alocar e liberar espaços do ***heap***.
  - Espaços alocados e não liberados são desalocados apenas no final da execução do programa.

Alocação Dinâmica

- Espaço de memória requisitado em tempo de execução.
- Utiliza funções de sistema para alocar e liberar espaços do ***heap***.
  - Espaços alocados e não liberados são desalocados apenas no final da execução do programa.
- **Vantagens:**
  - Minimiza o desperdício de recursos.
  - Otimiza o gerenciamento de memória no sistema.

Alocação Dinâmica

- Funções da biblioteca padrão **“stdlib.h”**:
  - malloc();
  - calloc();
  - realloc();
  - free();

Alocação Dinâmica: malloc()

- **void* malloc(size_t x)**: responsável por alocar um tamanho x de memória, e retornar um ponteiro para o endereço base de memória.

```
int main(void)
{
    int *pi;
    int quantidade = 10;
    pi = (int*) malloc(quantidade * sizeof(int));

    return 0;
}
```

Alocação Dinâmica: malloc()

- **void* malloc(size_t x)**: responsável por alocar um tama[illegible] x de memória, e retornar um ponteiro p[illegible] e memória.

SE RETORNAR NULL?

```
    pi = (int*) malloc(quantidade * sizeof(int));

    return 0;
}
```

Alocação Dinâmica: malloc()

- **void* malloc(size_t x)**: responsável por alocar um tamanho x de memória, e retornar um ponteiro

```c
int main(void)
{
    int *pi;
    int quantidade = 10;
    pi = (int*) malloc(quantidade * sizeof(int));

    if(pi == NULL)
    {
        printf("Falha ao alocar.");
        exit(1);
    }

    return 0;
}
```

Alocação Dinâmica: calloc()

- **void* calloc(size_t x, size_t y)**: responsável por alocar x vezes o tamanho y, devolvendo um ponteiro para o endereço base da região alocada.
- Inicializa o conteúdo da memória com valor **zero**.

```
int main(void)
{
    double *pd;
    double quantidade = 10;
    pd = (double*) calloc(quantidade, sizeof(double));

    return 0;
}
```

Alocação Dinâmica: realloc()

- **void* realloc(void* ptr, size_t x)**: responsável por modificar o tamanho de memória já alocada. Altera o tamanho da memoria apontada pelo ponteiro ptr para x bytes. Retorna um ponteiro para o local.
- Não perde o conteúdo da faixa de memória inicial.

Alocação Dinâmica: realloc()

- **v**
  m
  A
  p
  p
- N

```c
int main(void)
{
    char *string;
    //armazenar 22 caracteres e \0
    string = (char*) malloc(23 * sizeof(char));
    //colocando frase na string
    strcpy(string, "isso sao 22 caracteres");
    //realozando espaço para mais um caractere
    string = realloc(string, 24 * sizeof(char));
    //adicionando ponto final na string
    strcat(string, ".");

    printf("%s", string);

    return 0;
}
```

Alocação Dinâmica: free()

- **void free(void* ptr)**: responsável por devolver ao heap a memória apontada por ptr.
- Aceita apenas ponteiros alocados dinamicamente.

```c
int main(void)
{
    int *pi;
    int quantidade = 10;
    pi = (int*) malloc(quantidade * sizeof(int));

    free(pi);//liberar o espaço da heap

    pi = NULL;//resetar o ponteiro

    return 0;
}
```

Alocação Dinâmica

- Alocação em funções:
  - Declaração estática: perde referência ao fim da execução.
  - Declaração dinâmica: disponível para acesso após fim da execução.

Alocação Dinâmica

- Alocaçã
  - Declar
    exec
  - Declar
    da ex

im da

ap s fim

```
int* funcao(int *u, int *v)
{
    int p[2];
    p[0] = u[0] + v[0];
    p[1] = u[1] + v[1];
    return p;
}
```

```
int* funcao(int *u, int *v)
{
    int *p = (int*) malloc(2*sizeof(int));
    p[0] = u[0] + v[0];
    p[1] = u[1] + v[1];
    return p;
}
```

Alocação Dinâmica


```c
int* funcao(int *u, int *v)
{
    int *p = (int*) malloc(2*sizeof(int));
    p[0] = u[0] + v[0];
    p[1] = u[1] + v[1];
    return p;
}
```

```c
int main(void)
{
    int *p_retorno;
    int u[2] = {10, 20};
    int v[2] = {30, 40};

    p_retorno = funcao(u, v);

    printf("P[%d] = %d\n", 0, p_retorno[0]);
    printf("P[%d] = %d\n", 1, p_retorno[1]);

    free(p_retorno);
    p_retorno = NULL;

    return 0;
}
```

Alocação **ESTÁTICA** de Matriz

- Conjunto de vários vetores.
- Organizados por linhas e colunas.
- **Todos os dados estão dispostos sequencialmente na memória principal.**

```
int linhas = 5;
int colunas = 5;

int matriz[linhas][colunas];
```

Alocação **ESTÁTICA** de Matriz

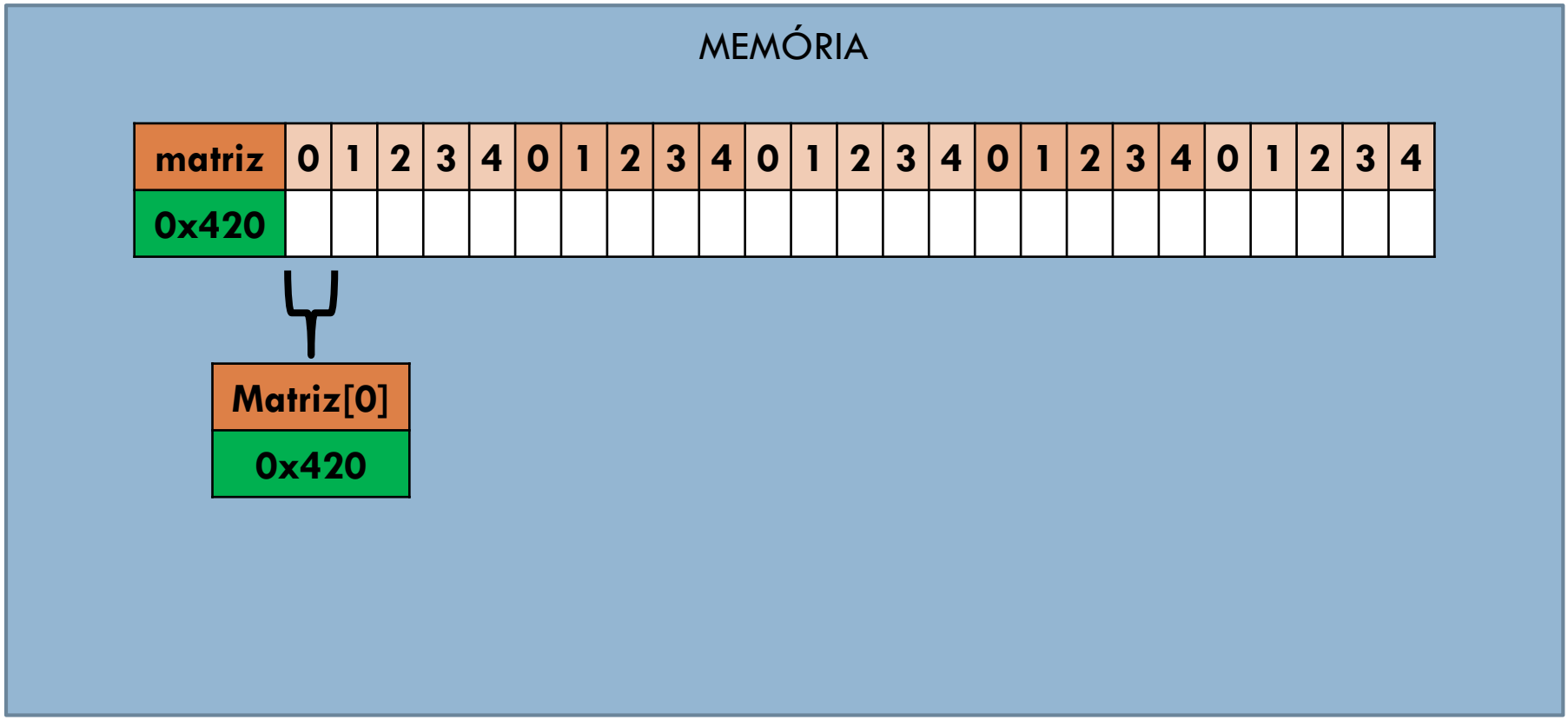

```
int linhas = 5;
int colunas = 5;

int matriz[linhas][colunas];
```

MEMÓRIA

matriz	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
0x420																									

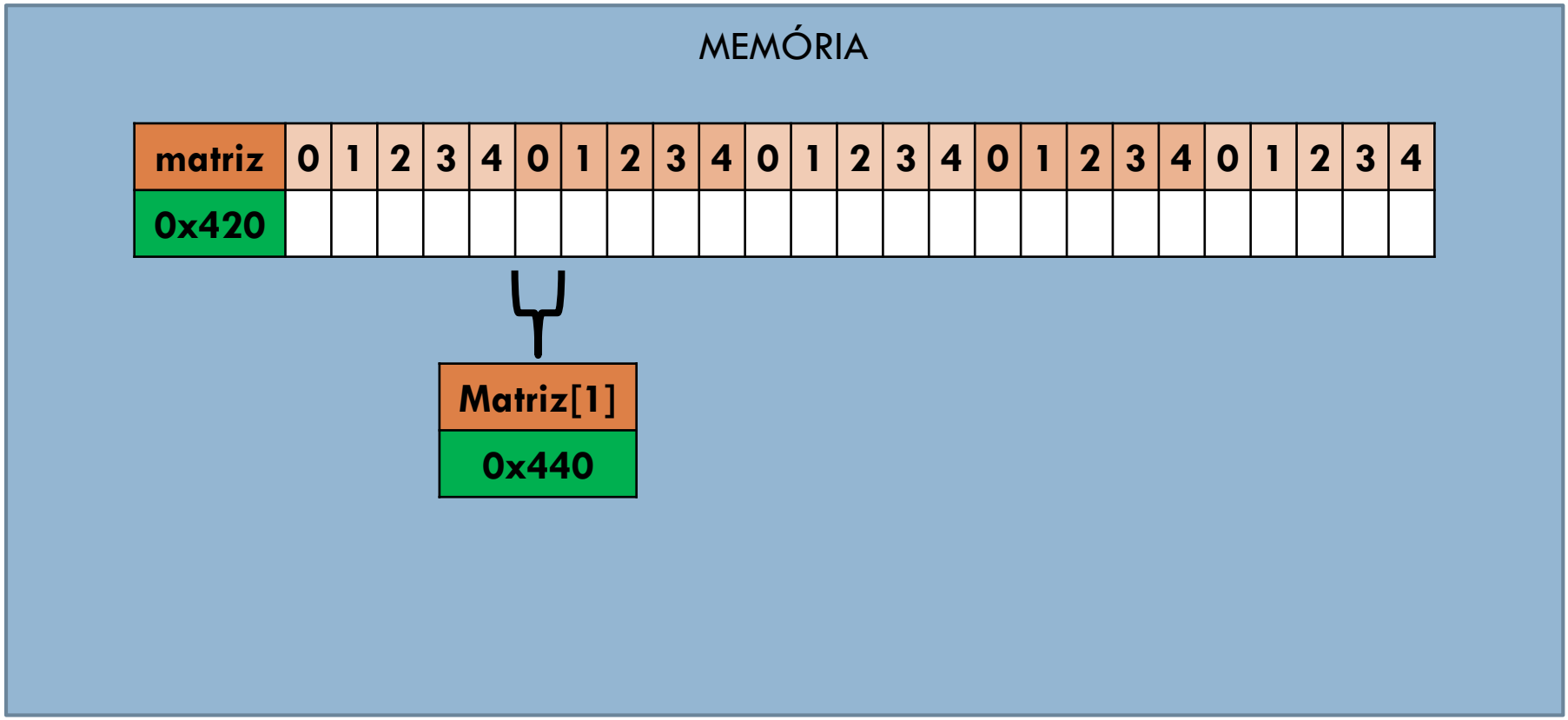
Alocação **ESTÁTICA** de Matriz


```
int linhas = 5;
int colunas = 5;

int matriz[linhas][colunas];
```



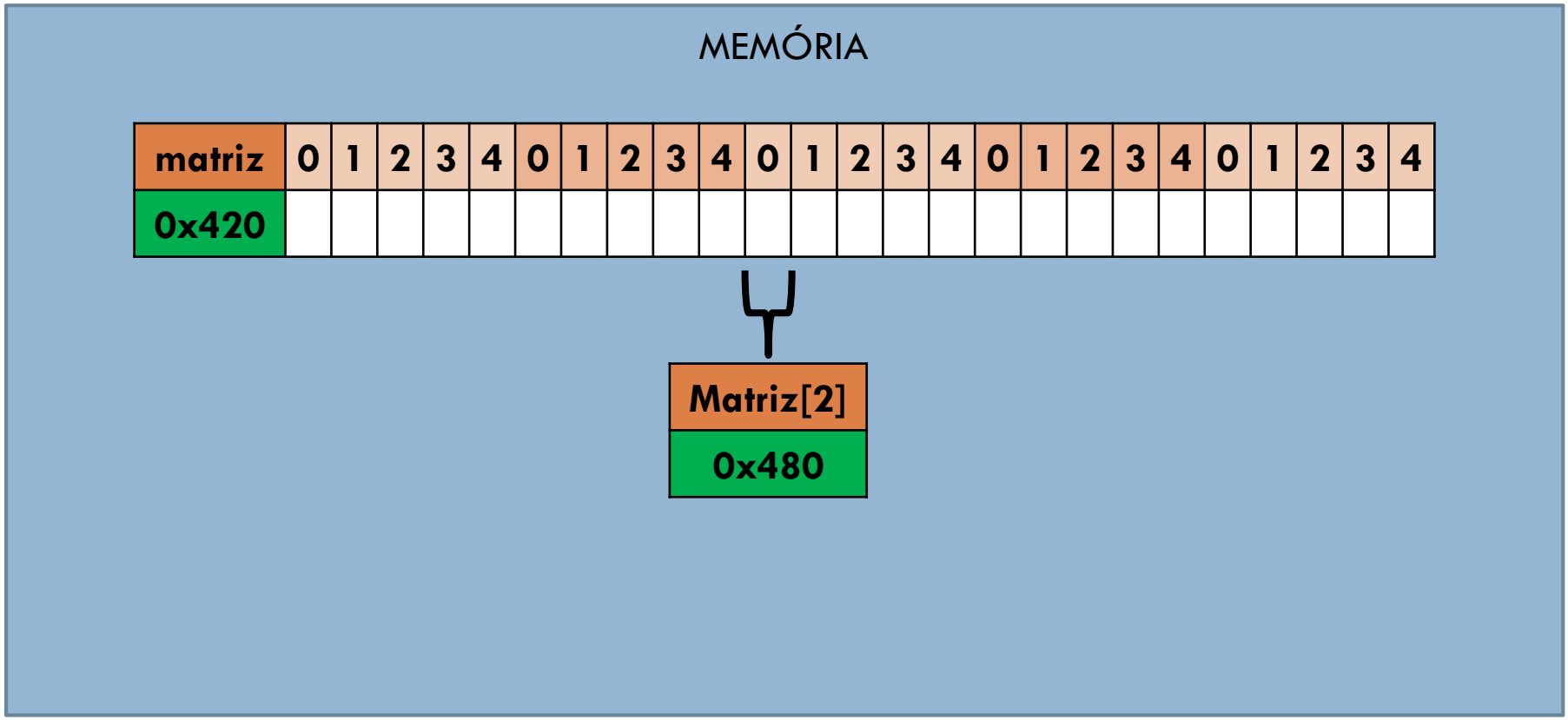
Alocação **ESTÁTICA** de Matriz


```
int linhas = 5;
int colunas = 5;

int matriz[linhas][colunas];
```



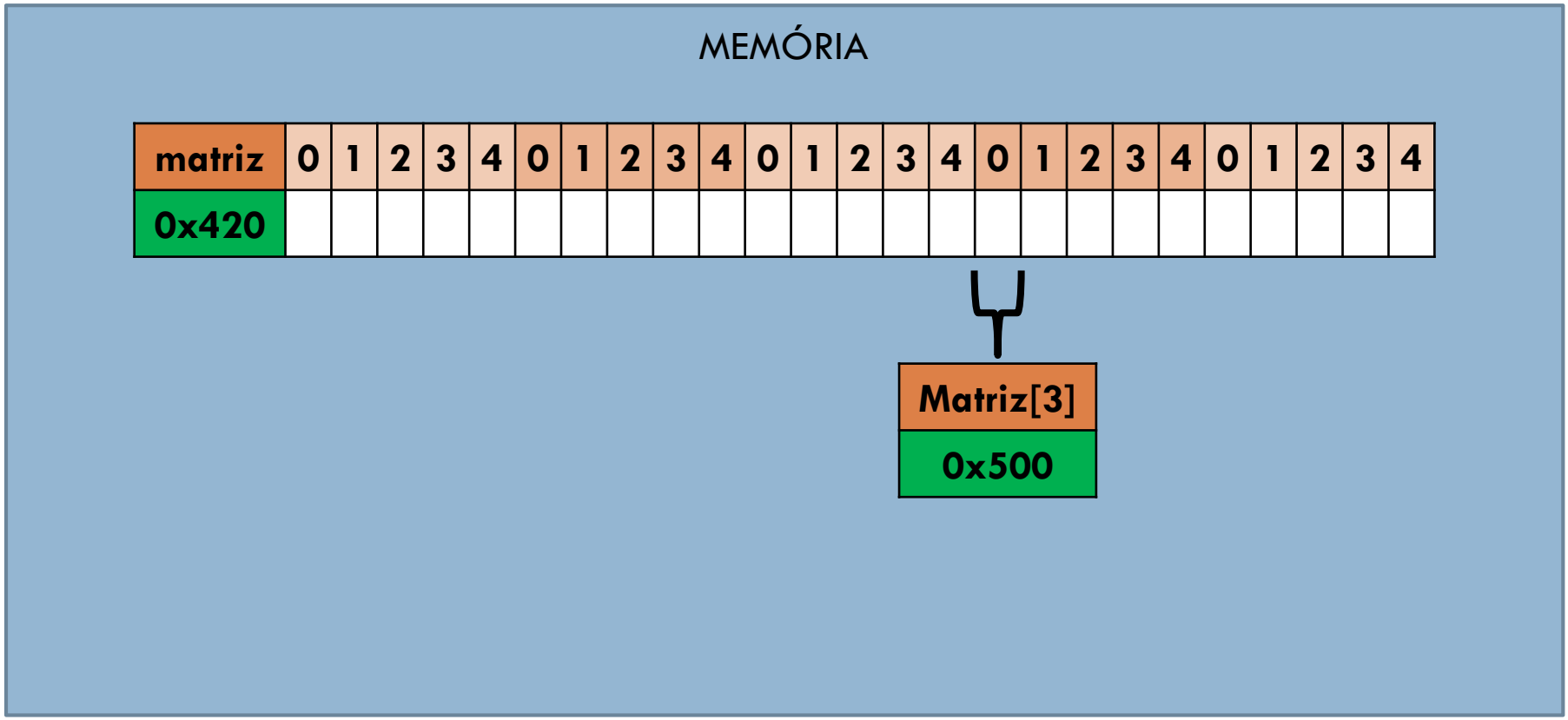
Alocação **ESTÁTICA** de Matriz


```
int linhas = 5;
int colunas = 5;

int matriz[linhas][colunas];
```



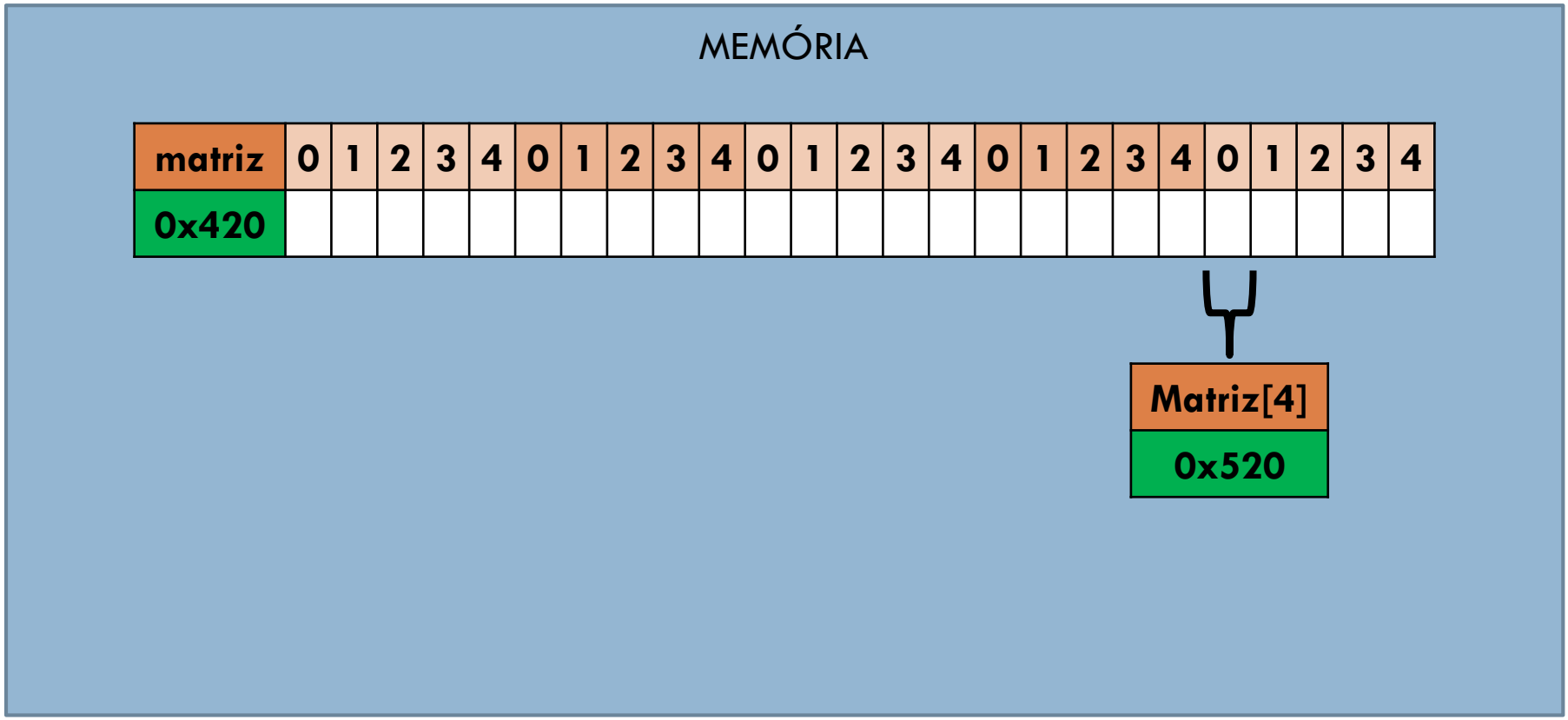
Alocação **ESTÁTICA** de Matriz


```
int linhas = 5;
int colunas = 5;

int matriz[linhas][colunas];
```



Alocação **ESTÁTICA** de Matriz


```
int linhas = 5;
int colunas = 5;

int matriz[linhas][colunas];
```



Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

FUNÇÃO PARA ALOCAR MATRIZ QUADRADA ***nXn***

```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

USO DE **PONTEIRO DE PONTEIRO**

```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n *

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

USO DE **PONTEIRO DE PONTEIRO**

Tipo de "variável" que armazena o **endereço de outro ponteiro.**

Alocação Dinâmica de Matriz

ALOCAR UM **VETOR DE PONTEIROS**

```
int** a
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

ALOCAR UM **VETOR DE PONTEIROS**

Uma "coluna" com *n* ponteiros, cada qual irá apontar para um vetor.

```
int** a
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

ALOCAR UM **VETOR PARA CADA LINHA**

```
int** a
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

ALOCAR UM **VETOR PARA CADA LINHA**

Cada ponteiro da "coluna" recebe um vetor alocado dinamicamente.

```
int** a
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

ALOCAR UM **VETOR PARA CADA LINHA**

Cada ponteiro da "coluna" recebe um vetor alocado dinamicamente.

Os vetores não estarão necessariamente em sequência na memória.

```
int** a
{
    int i;
    int **mat;

    mat = (int**) malloc(n *

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

RETORNAR O PONTEIRO DE PONTEIRO.

```
int** a
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

RETORNAR O PONTEIRO DE PONTEIRO.

Ao ser alocado **na *HEAP* a matriz continuará existindo após finalizar a função.**

```
int** a
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz

RETORNAR O PONTEIRO DE PONTEIRO.

Ao ser alocado **na *HEAP* a matriz continuará existindo após finalizar a função.**

A matriz só será liberada quando o programa finalizar ou ser dado um comando *free(mat)*.

```
int** a
{
    int i;
    int **mat;

    mat = (int**) malloc(n *

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

Alocação Dinâmica de Matriz


```c
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

Alocação Dinâmica de Matriz

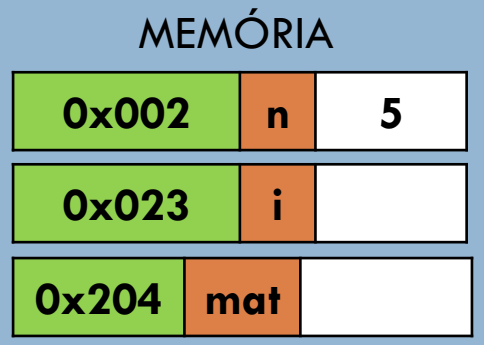

```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```



Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

PRIMEIRO RESERVA O ESPAÇO NA *HEAP*.

MEMÓRIA

0x002	n	5
0x023	i	
0x204	mat	

0x420	
0	
1	
2	
3	
4	

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

VARIÁVEIS NA **HEAP NÃO POSSUEM *NOMES*.**

MEMÓRIA

0x002	n	5
0x023	i	
0x204	mat	

0x420	
0	
1	
2	
3	
4	

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

APÓS RESERVAR, ARMAZENA O ENDEREÇO NO PONTEIRO.

MEMÓRIA

0x002	n	5
0x023	i	
0x204	mat	0x420

0x420	
0	
1	
2	
3	
4	

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	0
0x204	mat	0x420

0x420	
0	
1	
2	
3	
4	

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	0
0x204	mat	0x420

0x420	
0	
1	
2	
3	
4	

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

PRIMEIRO RESERVA O ESPAÇO NA *HEAP*.

MEMÓRIA

0x002	n	5
0x023	i	0
0x204	mat	0x420

0x420	
0	
1	
2	
3	
4	

0x850	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

APÓS RESERVAR, ARMAZENA O ENDEREÇO NO PONTEIRO.

MEMÓRIA

0x002	n	5
0x023	i	0
0x204	mat	0x420

0x420	
0	0x850
1	
2	
3	
4	

0x850	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	1
0x204	mat	0x420

0x420	
0	0x850
1	
2	
3	
4	

0x850	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	1
0x204	mat	0x420

0x420	
0	0x850
1	
2	
3	
4	

0x850	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

PRIMEIRO RESERVA O ESPAÇO NA *HEAP*.

MEMÓRIA

0x002	n	5
0x023	i	1
0x204	mat	0x420

0x420	
0	0x850
1	
2	
3	
4	

0x850	0	1	2	3	4

0x300	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

APÓS RESERVAR, ARMAZENA O ENDEREÇO NO PONTEIRO.

MEMÓRIA

0x002	n	5
0x023	i	1
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	
3	
4	

0x850	0	1	2	3	4

0x300	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	2
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	
3	
4	

0x850	0	1	2	3	4

0x300	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	2
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	
3	
4	

0x850	0	1	2	3	4

0x300	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

PRIMEIRO RESERVA O ESPAÇO NA *HEAP*.

MEMÓRIA

0x002	n	5
0x023	i	2
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	
3	
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

APÓS RESERVAR, ARMAZENA O ENDEREÇO NO PONTEIRO.

MEMÓRIA

0x002	n	5
0x023	i	2
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	3
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	3
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

PRIMEIRO RESERVA O ESPAÇO NA *HEAP*.

MEMÓRIA

0x002	n	5
0x023	i	3
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

APÓS RESERVAR, ARMAZENA O ENDEREÇO NO PONTEIRO.

MEMÓRIA

0x002	n	5
0x023	i	3
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	4
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	4
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

PRIMEIRO RESERVA O ESPAÇO NA *HEAP*.

MEMÓRIA

0x002	n	5
0x023	i	4
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

0x580	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

APÓS RESERVAR, ARMAZENA O ENDEREÇO NO PONTEIRO.

MEMÓRIA

0x002	n	5
0x023	i	4
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	0x580

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

0x580	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	5
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	0x580

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

0x580	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

MEMÓRIA

0x002	n	5
0x023	i	5
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	0x580

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

0x580	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

RETORNAR O PONTEIRO DE PONTEIRO.

MEMÓRIA

0x002	n	5
0x023	i	5
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	0x580

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

0x580	0	1	2	3	4

Alocação Dinâmica de Matriz


```
int** aloca_matriz(int n)
{
    int i;
    int **mat;

    mat = (int**) malloc(n * sizeof(int*));

    for(i = 0; i < n; i++)
        mat[i] = (int*) malloc(n * sizeof(int));

    return mat;
}
```

COM O PONTEIRO DE MAT É POSSIVEL LOCALIZAR TODOS OS OUTROS ENDEREÇOS.

MEMÓRIA

0x002	n	5
0x023	i	5
0x204	mat	0x420

0x420	
0	0x850
1	0x300
2	0x630
3	0x720
4	0x580

0x850	0	1	2	3	4

0x630	0	1	2	3	4

0x300	0	1	2	3	4

0x720	0	1	2	3	4

0x580	0	1	2	3	4

Alocação Dinâmica de Matriz

- Vetores são estruturas alocadas sempre sequencialmente na memória.
- Com a alocação dinâmica de matrizes é possível:
  - Alocar matrizes maiores.
    - Menor restrição de tamanho por faixas de memória.
    - Fragmentação dos dados em espaços adequados.
  - Alocar matrizes onde cada linha possui um tamanho diferente, otimizando espaço durante a execução.