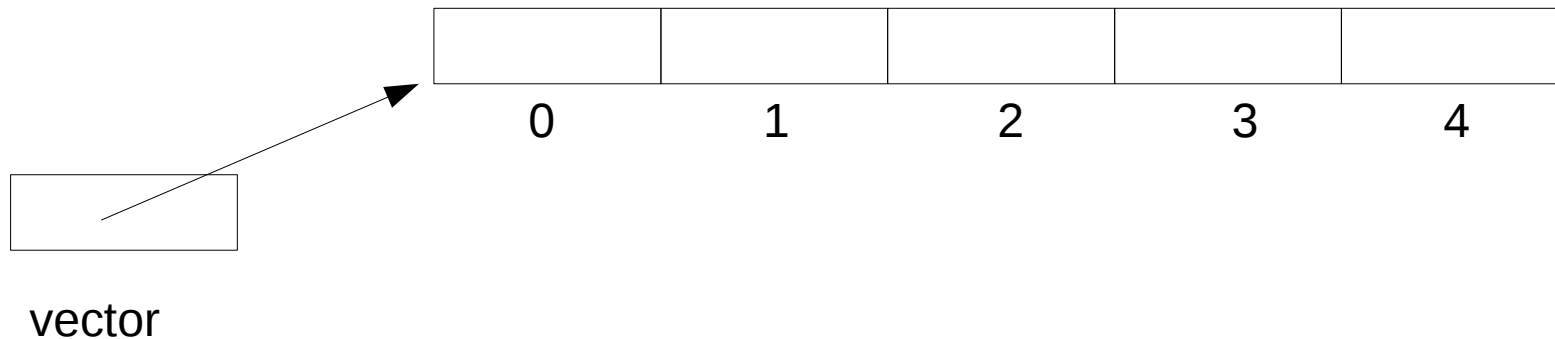


Memoria Dinámica

Vectores y matrices

Vectores

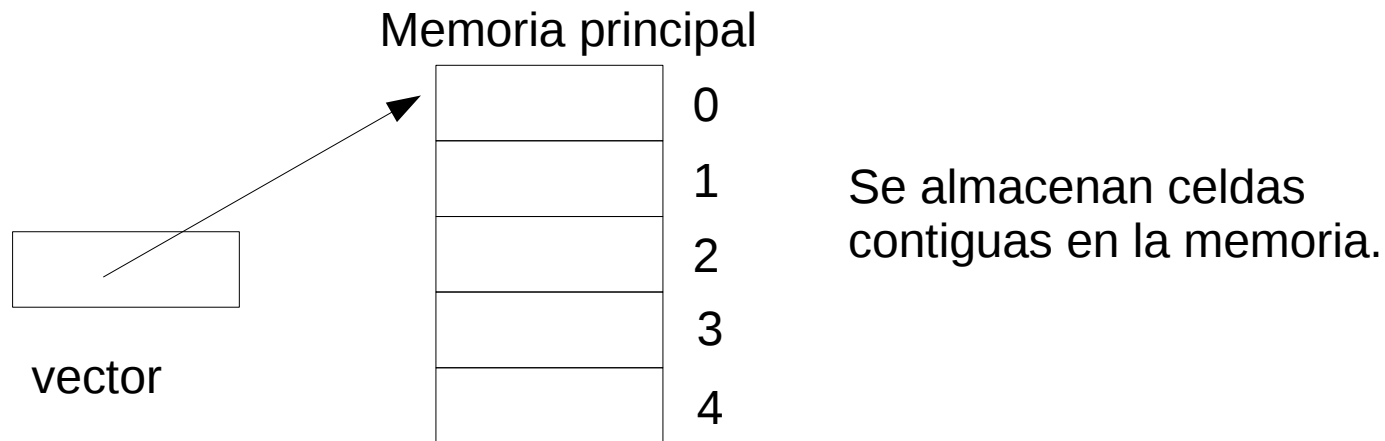
```
int size = n * sizeof(int);  
int *vector = malloc(size);
```



Visión Lógica

Vectores

```
int size = n * sizeof(int);  
int *vector = malloc(size);
```

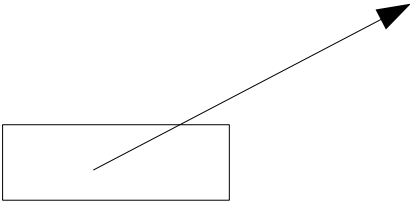


Visión física

Matrices

```
int size = N * N * sizeof(int);
```

```
int *matriz = malloc(size);
```



	0	1	2	3	4	
	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	0
	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	1
						2
						3
					(4,4)	4

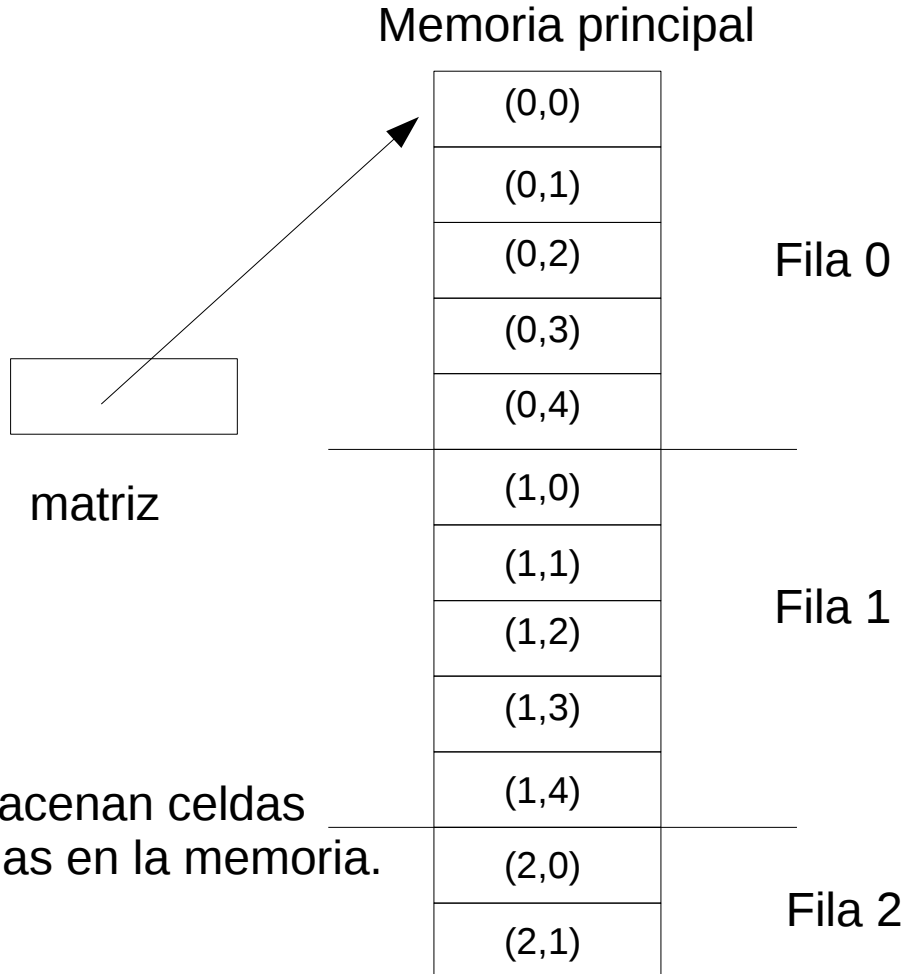
matriz

Visión lógica → 2 dimensiones!!

Matrices

```
int size = N * N * sizeof(int);
```

```
int *matriz = malloc(size);
```



Visión física:
1D!!

Matrices

```
int size = N * N * sizeof(int);
```

```
int *matriz = malloc(size);
```

matriz

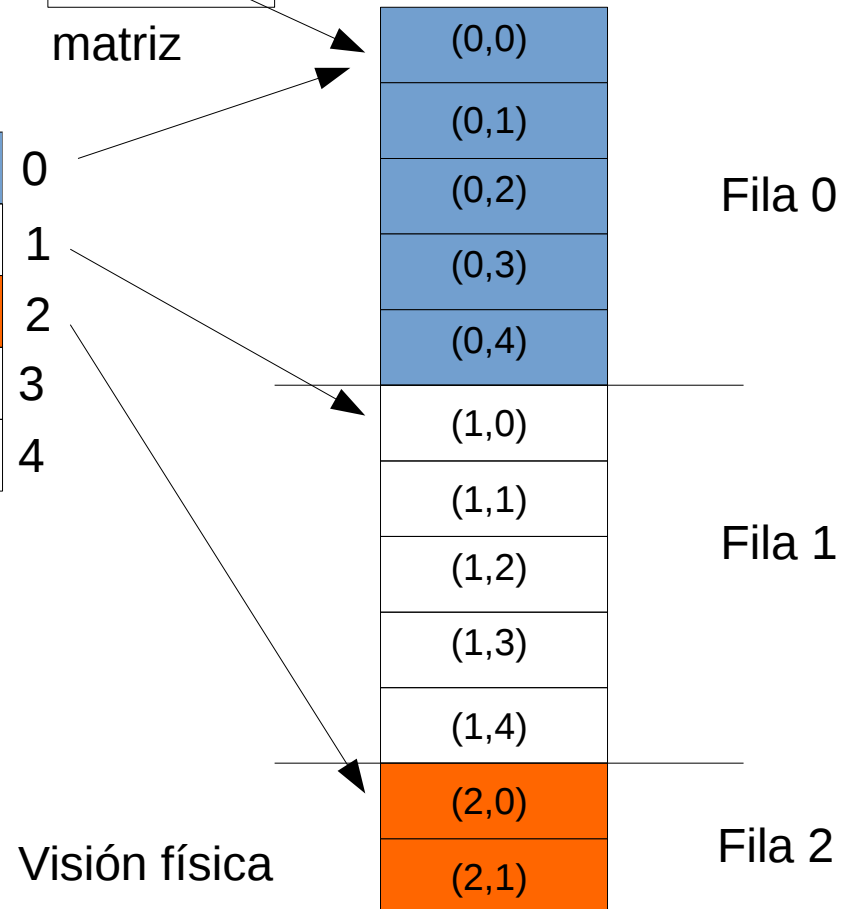


0	1	2	3	4
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
				(4,4)

Visión lógica

Memoria principal

matriz



Visión física

Matrices

```
int size = N * N * sizeof(int);
```

```
int *matriz = malloc(size);
```

matriz



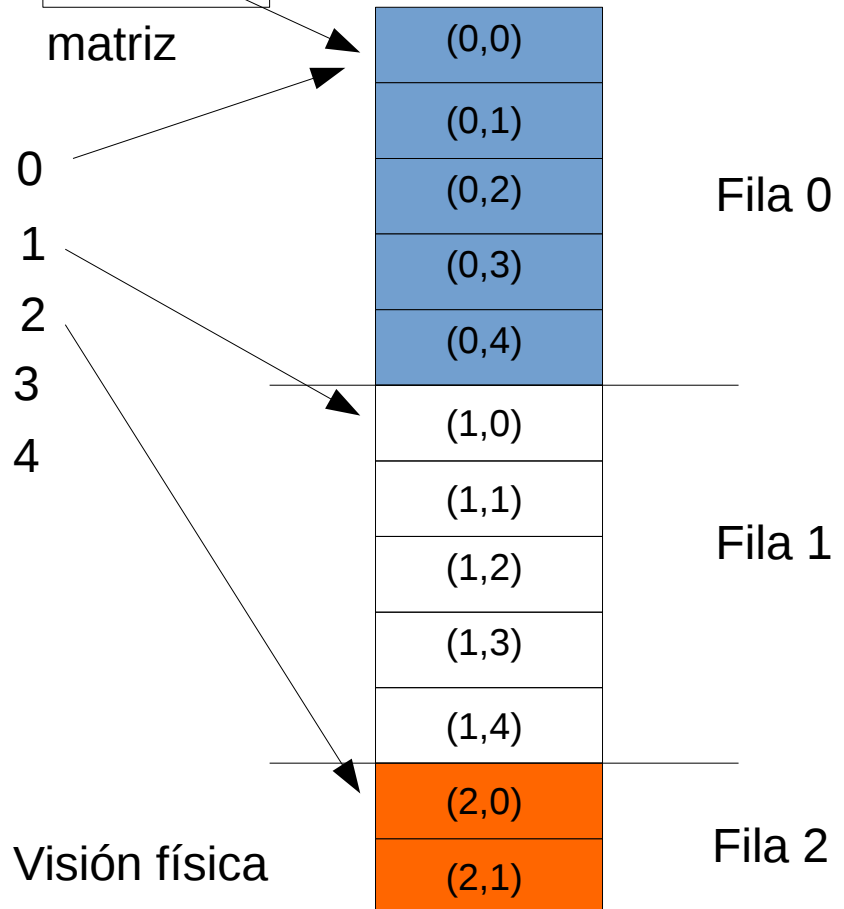
0	1	2	3	4
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
				(4,4)

Visión lógica `matriz[fila][columna]`

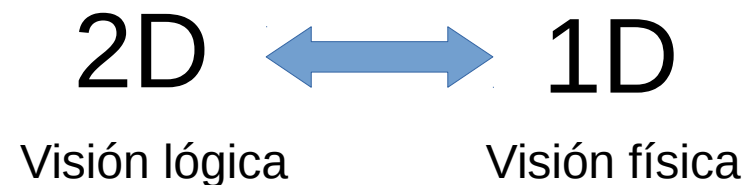


Visión física `matriz[??]`

Memoria principal
matriz



- Es necesario acceder a la matriz usando un único índice → pasar coordenadas



2D \longleftrightarrow 1D
Visión lógica Visión física

matriz[**fila**][**columna**] \longleftrightarrow matriz[**coordenada 1D**]
Visión lógica Visión física



fila * cantidad de columnas \longrightarrow comienzo de la fila

(**fila** * cantidad de columnas) + **columna** \longrightarrow **coordenada 1D**

2D \longleftrightarrow 1D
Visión lógica Visión física

matriz[**fila**][**columna**] \longleftrightarrow matriz[**coordenada 1D**]
Visión lógica Visión física



coordenada 1D / cantidad de columnas \longrightarrow **fila**

coordenada 1D % cantidad de columnas \longrightarrow **columna**

Matrices

0	1	2	3	4	
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	0
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	1
					2
					3
				(4,4)	4

$\text{matriz}[0][4] \rightarrow \text{matriz}[0 * 5 + 4] \rightarrow \text{matriz}[4]$

Memoria principal

0	(0,0)	
1	(0,1)	
2	(0,2)	
3	(0,3)	
4	(0,4)	Fila 0
5	(1,0)	
6	(1,1)	
7	(1,2)	
8	(1,3)	
9	(1,4)	Fila 1
10	(2,0)	
11	(2,1)	Fila 2

Visión física

Matrices

0	1	2	3	4	
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	0
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	1
					2
					3
				(4,4)	4

$\text{matriz}[1][2] \rightarrow \text{matriz}[1 * 5 + 2] \rightarrow \text{matriz}[7]$

Memoria principal

0	(0,0)	Fila 0
1	(0,1)	
2	(0,2)	
3	(0,3)	
4	(0,4)	
5	(1,0)	Fila 1
6	(1,1)	
7	(1,2)	
8	(1,3)	
9	(1,4)	
10	(2,0)	Fila 2
11	(2,1)	

Visión física

Matrices

0	1	2	3	4	
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	0
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	1
					2
					3
				(4,4)	4

`matriz[11] → matriz[11 / 5 = 2][11 % 5 = 1]`

`matriz[11] → matriz[2][1]`

Memoria principal

0	(0,0)	Fila 0
1	(0,1)	
2	(0,2)	
3	(0,3)	
4	(0,4)	
5	(1,0)	Fila 1
6	(1,1)	
7	(1,2)	
8	(1,3)	
9	(1,4)	
10	(2,0)	Fila 2
11	(2,1)	

Matrices

0	1	2	3	4	
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	0
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	1
					2
					3
				(4,4)	4

`matriz[8]` → `matriz[8 / 5 = 1][8 % 5 = 3]`

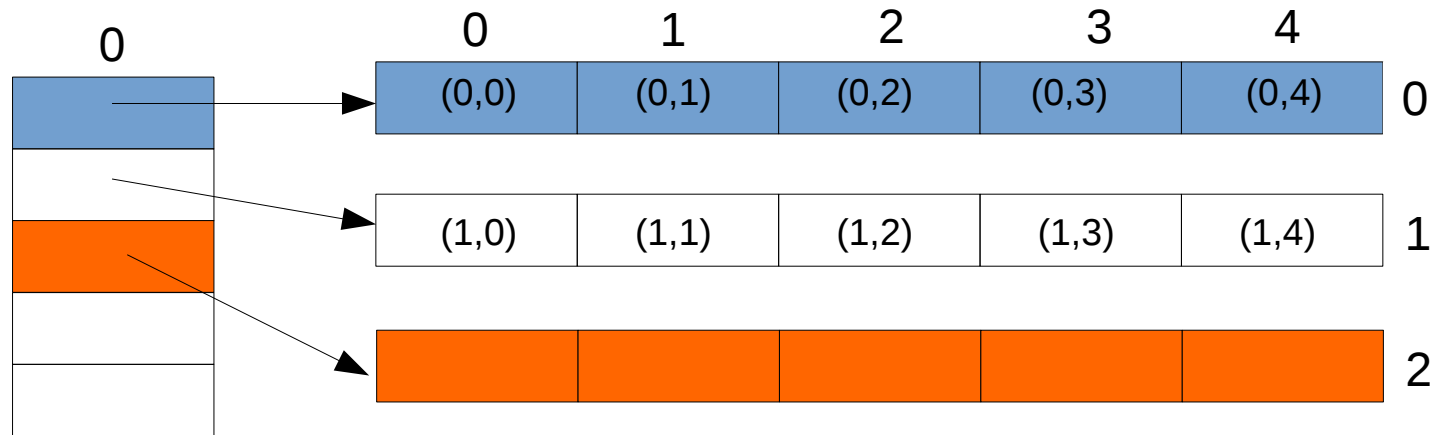
`matriz[8]` → `matriz[1][3]`

Memoria principal

0	(0,0)	Fila 0
1	(0,1)	
2	(0,2)	
3	(0,3)	
4	(0,4)	
5	(1,0)	Fila 1
6	(1,1)	
7	(1,2)	
8	(1,3)	
9	(1,4)	
10	(2,0)	Fila 2
11	(2,1)	

Matrices

- Más opciones: usar doble puntero



Arreglo de punteros

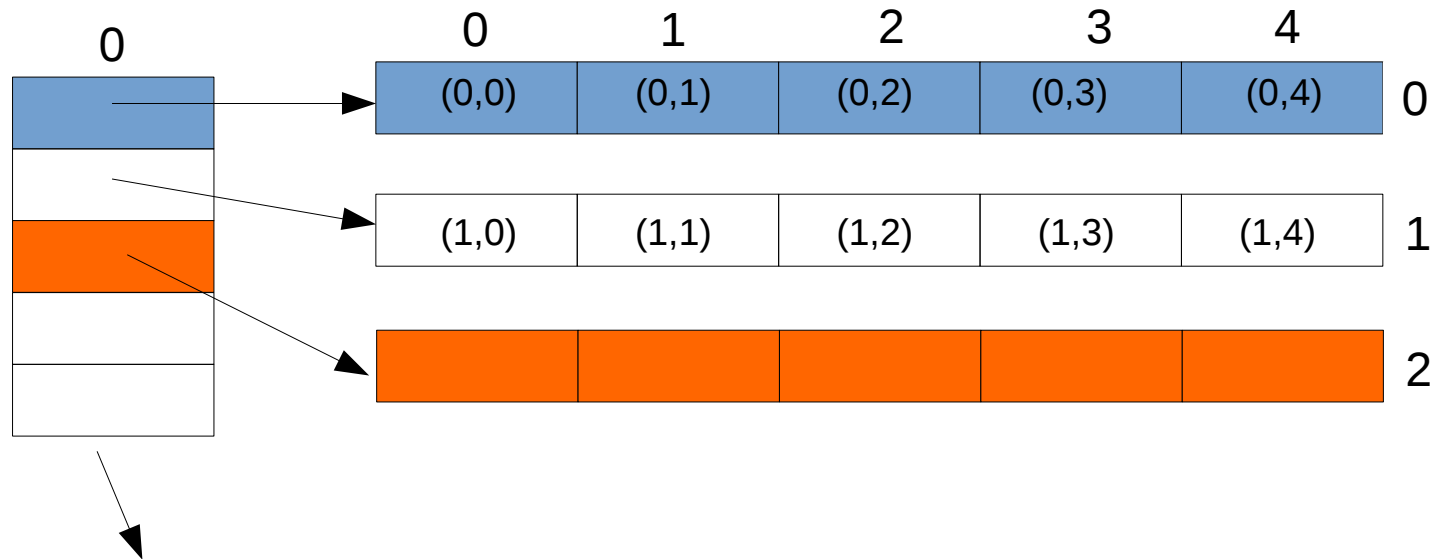
`matriz[i]` = puntero a un
vector de enteros

```
int **matriz, i, j;

matriz = malloc(filas * sizeof(int*));
if (matriz != NULL) {
    for (i = 0; i < filas; i++) {
        matriz[i] = malloc(columnas * sizeof(int));
        if (!matriz[i]) {printf("Error allocating row %d \n", i); exit(-1);}
    }
} else {printf("Error allocating matrix \n"); exit(-1);}
```

Matrices

- Más opciones: usar doble puntero



Arreglo de punteros

```
for (i = 0; i < filas; i++)  
    for (j = 0; j < columnas; j++)  
        matriz[i][j] = rand()%100;
```


Puntero a función

- Los punteros pueden apuntar a cualquier tipo de variables, estructuras o arreglo.
- Y también pueden apuntar a funciones!! (apuntan a código ejecutable). Al igual que los datos, las funciones se almacenan en memoria y tienen direcciones iniciales.

- Sintaxis

`tipo_de_retorno (*puntero_funcion) (<lista_de_parametros>)`

- Inicialización

`puntero_funcion = una_funcion;`

- Ejemplo

`int funcion(int);`

`int (*pf)(int);`

`pf = funcion;`