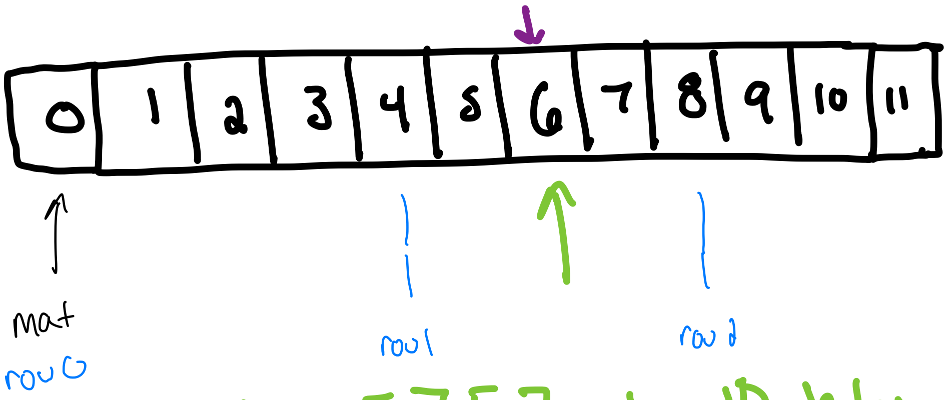
C - MULTIDIMENSIONAL ARRAYS

2D ARRAYS - DYNAMIC ALLOCATION

```
int *mat = (int *) malloc(nrows*ncols*sizeof(int));
```

- One contiguous block of memory
- Can't use `[][]` notation:
- Can use `[]` notation
- Pointer arithmetic to handle rows and columns

```
int * mat = (int *) malloc(nrows * ncols * sizeof(int));
```



← what it actually looks like

conceptually what we are representing

	0	1	2	3
0	0	1	2	3
1	4	5	$\boxed{6}$	7
2	8	9	10	11

Goal: go from 2d [i][j] to 1D index

$$*(mat + \underbrace{\boxed{i * ncols + j}}_{\text{1D offset for [i][j]}})$$

$$*(mat + 1 * 4 + 2) = *(mat + 6)$$

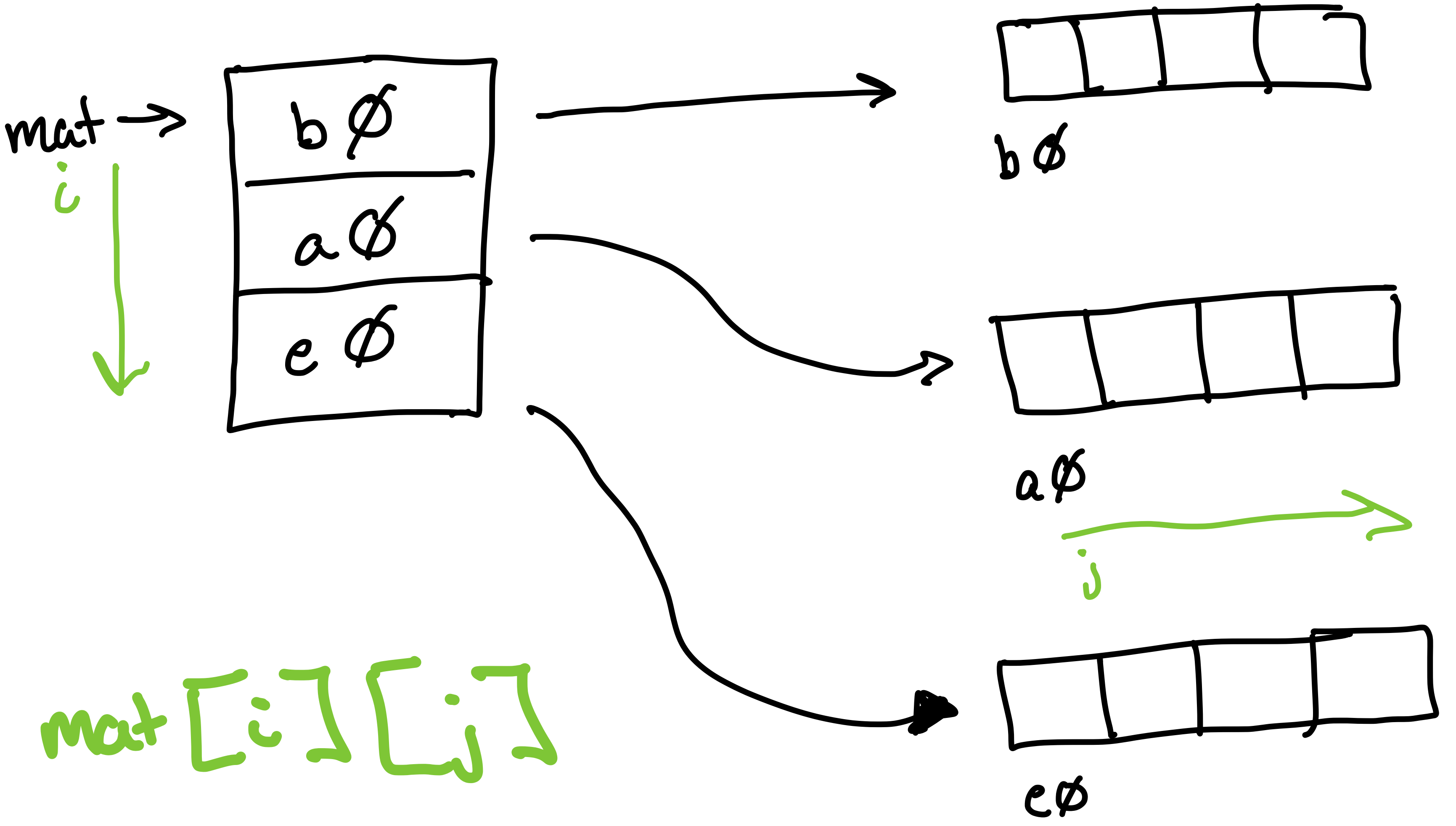
row = 1
col = 2

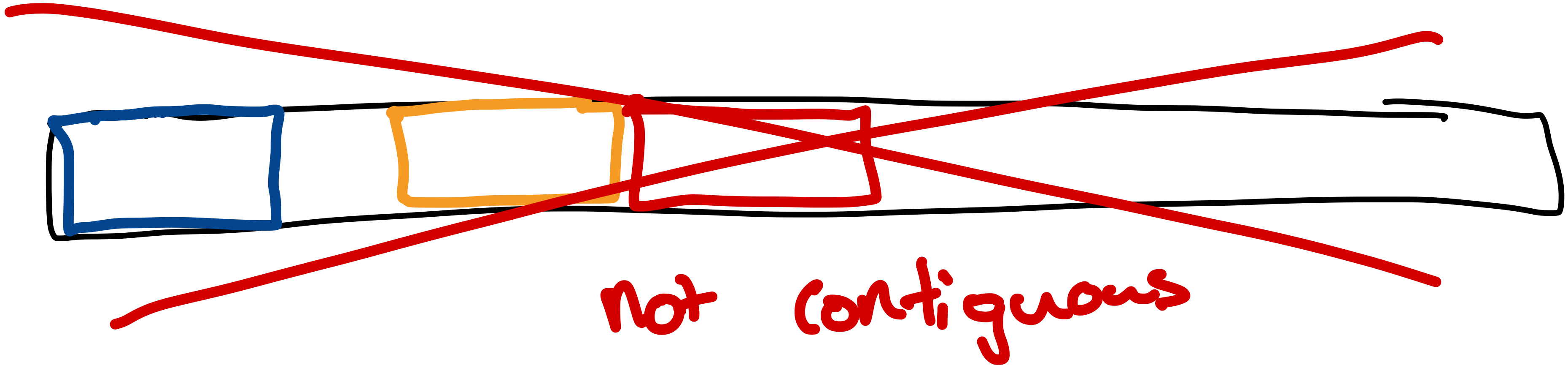
mat[1][2]

2D ARRAYS - DYNAMIC ALLOCATION

```
int **mat = (int **) malloc(nrows*sizeof(int *));
for (int i=0; i<nrows; i++) {
    *(mat+i) = (int *) malloc(ncols*sizeof(int));
}
```

- Could also use `mat[i]` inside the loop
- Can use `[][]` notation now
- No longer one contiguous block of memory

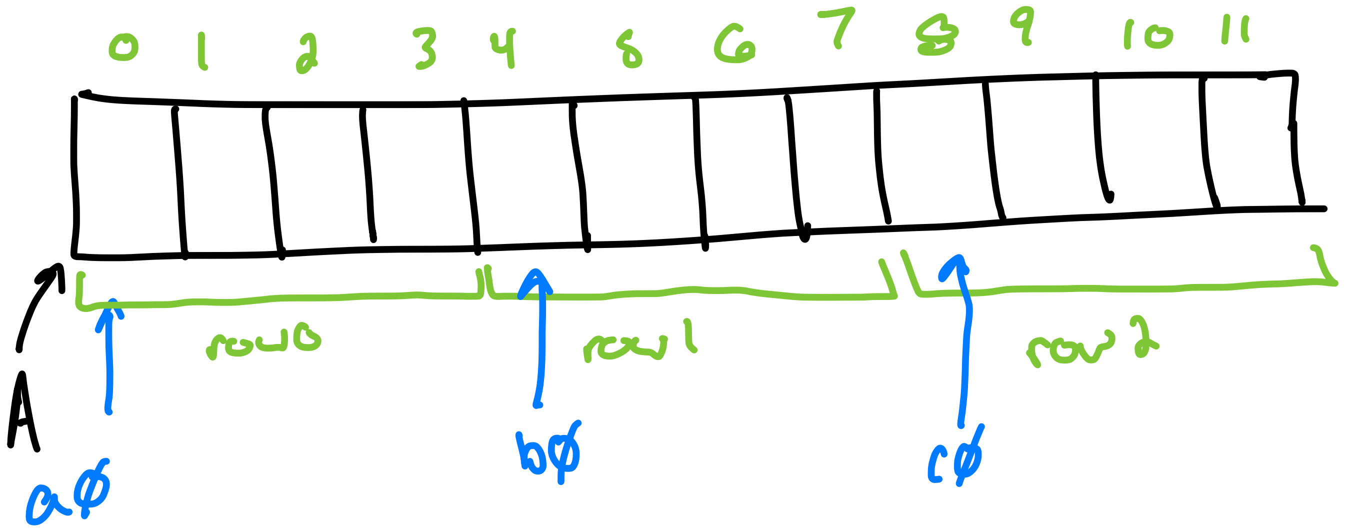

mat
i
b∅
a∅
e∅
b∅
a∅
j
e∅
mat[i][j]

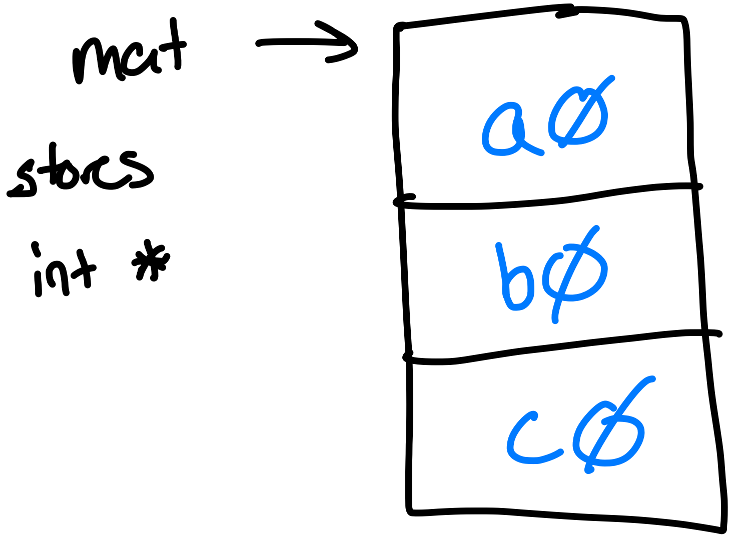

not contiguous

2D ARRAYS - DYNAMIC ALLOCATION

```
int *A = (int *) malloc(nrows*ncols*sizeof(int));
int **mat = (int **) malloc(nrows*sizeof(int *));
for (int i=0; i<nrows; i++) {
    mat[i] = A + i*ncols;
}
```

- Allows use of `[][]` notation
- Meomory for actual entries is contiguous





$A + 0 * \overset{ncols}{4} = A + 0$ i= 0

$A + 1 * 4 = A + 4$ i= 1

$A + 2 * 4 = A + 8$ i=2

malloc(nrows * ncols * sizeof(int))

contiguous

$mat[i] = A + i * ncols$