Array of Pointers | Coursera

coursera.org/learn/pointers-arrays-recursion/supplement/IpCYm/array-of-pointers

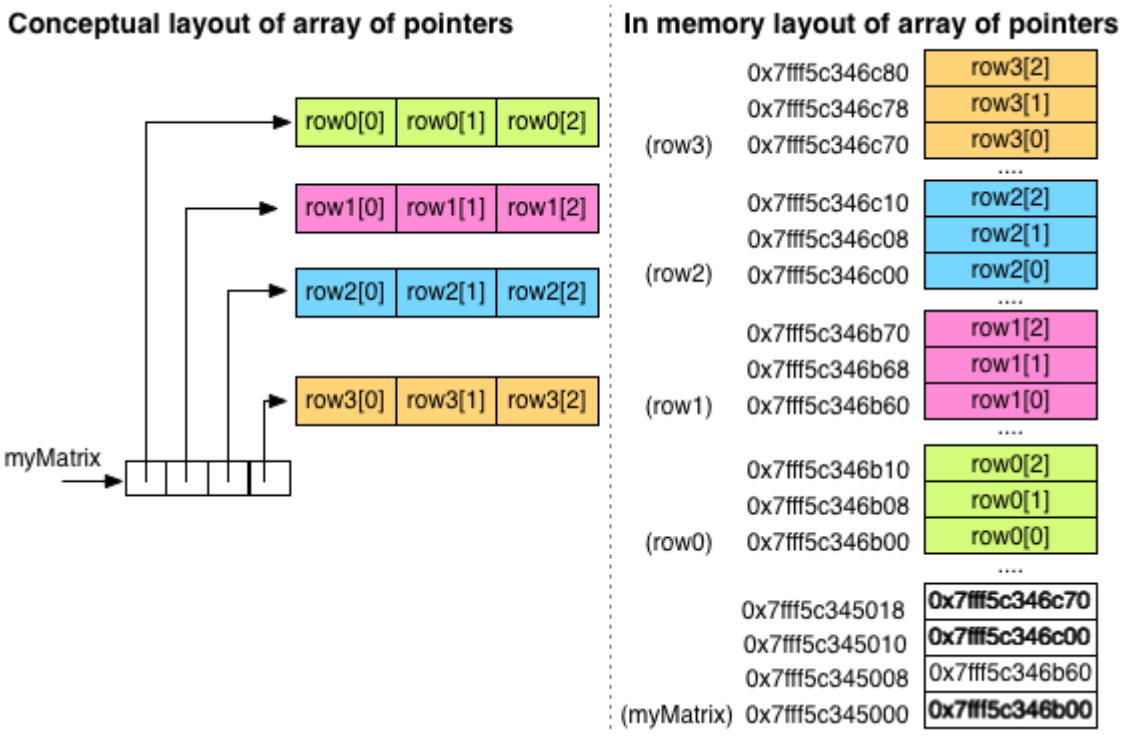
Array of Pointers

Array of Pointers

We can also represent multidimensional data with arrays that explicitly hold pointers to other arrays. For example, we might write the following:

```
5

double * myMatrix[4] = {row0, row1, row2, row3};
```



In the figure above, we again have a 4x3 matrix, however, this matrix is represented in a rather different fashion in memory. Here, *myMatrix* is an array of 4 pointers, which explicitly point at the arrays that are the rows of the matrix. On the left, this figure depicts the conceptual representation of this data structure: *myMatrix* is an array of 4 pointers, and each of these pointers points at one of the arrays *row0–row3*. The right side of this

figure depicts the in-memory layout of this data structure. Here, each of the row arrays may not be next to each other in memory (they might be, but do not have to be). The four entries of *myMatrix* now hold pointers to (the addresses of) the four row arrays.

Elements of the arrays are accessed in similar ways for both representations. For either representation, *myMatrix*[*2*], evaluates to a pointer to the array which is the second row of the matrix. Likewise, *myMatrix*[*2*] evaluates to a pointer to the **double** in the first column of the second row of the matrix.

However, there are some significant differences. First, in this array of pointers representation, the pointers to the rows are explicitly stored in memory. Accordingly, evaluating *myMatrix* [*i*] actually involves reading a value from memory, not just computing an offset. This difference has performance implications, which we will not go into here, as we are not prepared to discuss performance (such a discussion requires a detailed knowledge of hardware).

Explicitly storing the pointers to the rows of the matrix allows us to do some things with this representation which we cannot do with the first representation. First, we are not constrained to having each row be the same size as the other rows. Second, in the array of pointers representation, *myMatrix* [*i*] is an lvalue (recall that it is not if we just declare an array with multiple dimensions). Accordingly, we can change where the pointers point if we so desire. Third, we can have two rows point at the exact same array (aliasing each other). While these abilities may not be terribly useful for a mathematical matrix, they can be incredibly useful for a variety of other tasks which have data with multiple dimensions. This array of pointers representation will also prove quite useful when we learn about dynamic allocation.