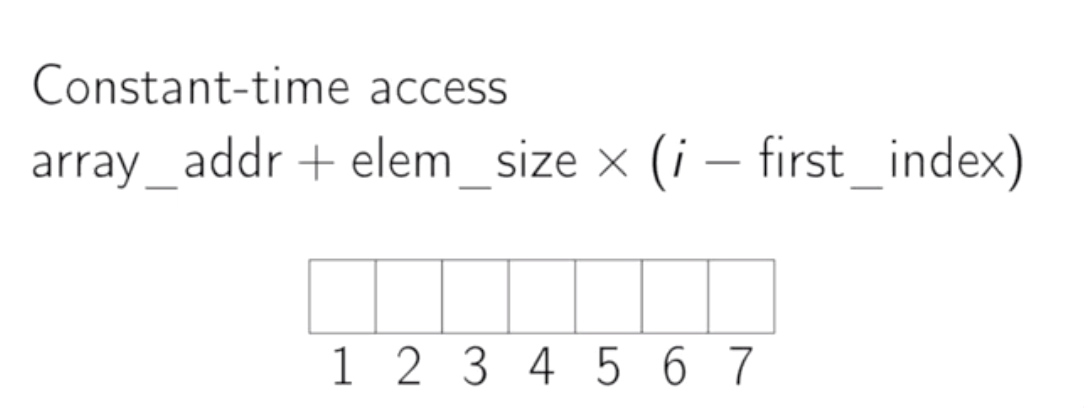
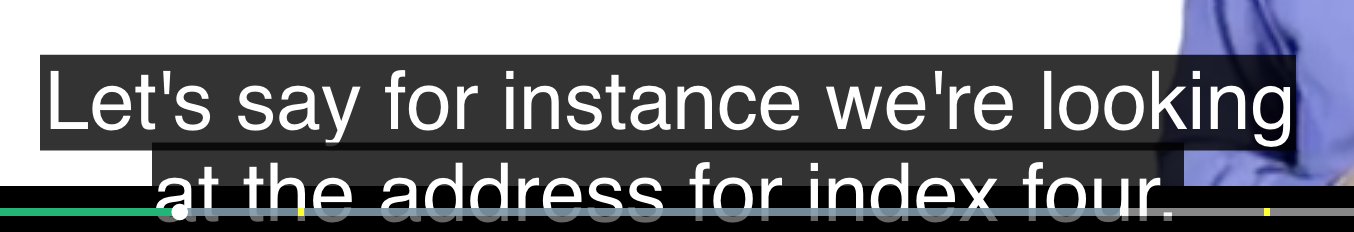
(This is 1 based indexing)(You can assume in 0 based indexing also)

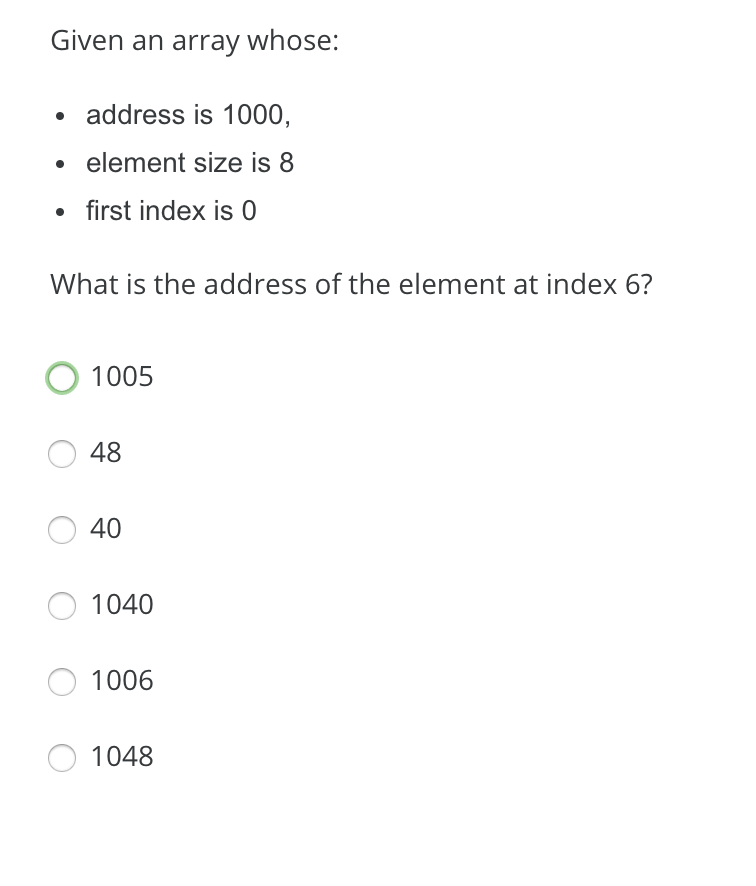
Array speciality:

1. We have random access or constant time access to any particular element in an array.(Constant time access to read and constant time access to write)



If you are doing 0 based indexing, ‘first\_indexing’ is not required in the formulae.

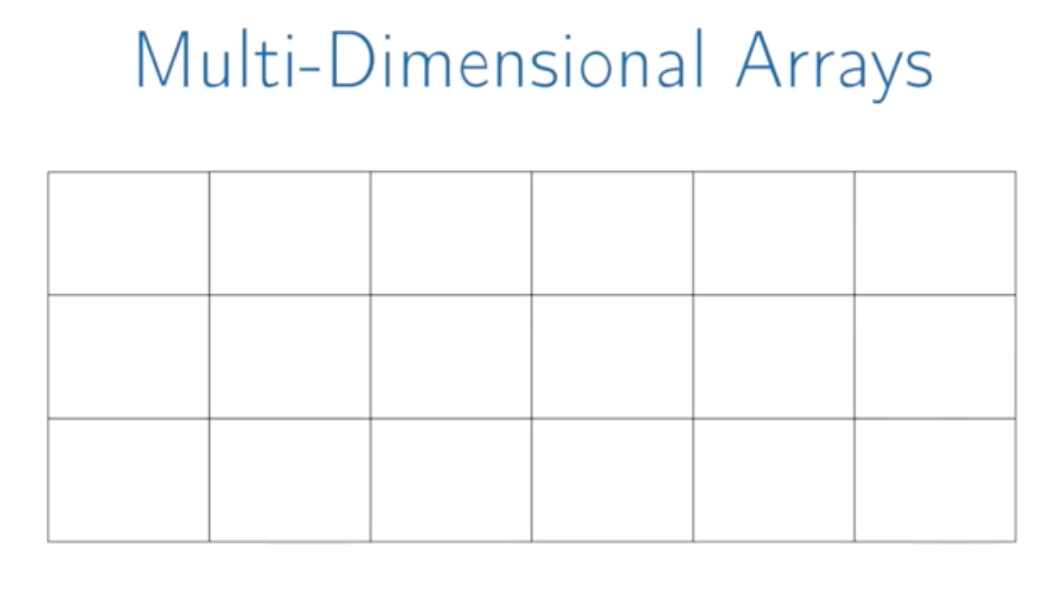


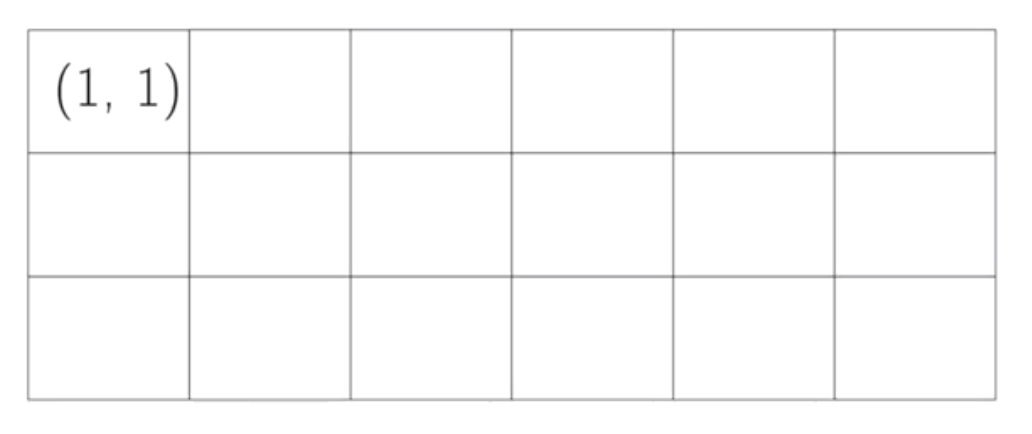


1048.

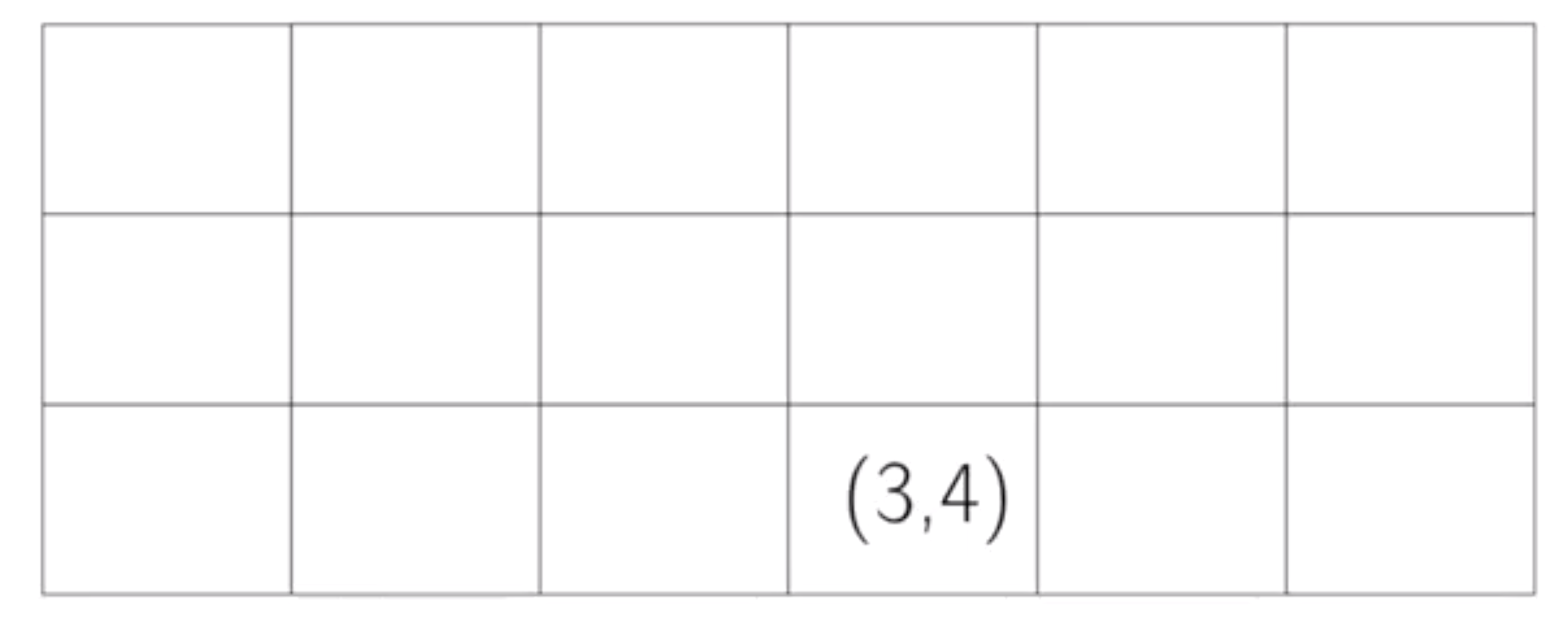
In arrays, the element size is equal for every element. If the element sizes are different, we need to sum all of it. (If there are n elements, we need O(n) time).

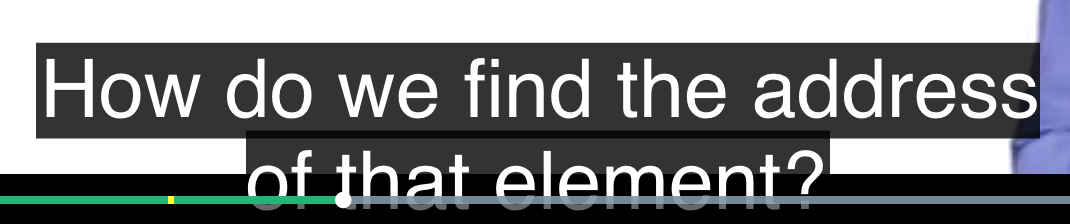
Anyway, the compiler will do all of this.



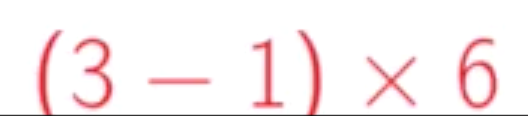




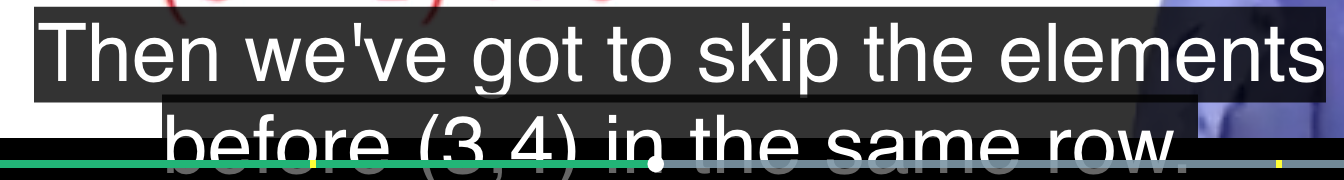




So first what we need to do is, skip all the rows which we are not using(row index -1). 1 is initial row index.

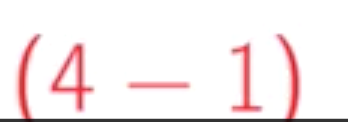


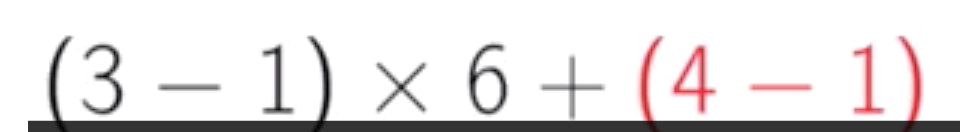
So we will get to row 3.



So there are 3 of them. How to skip those?

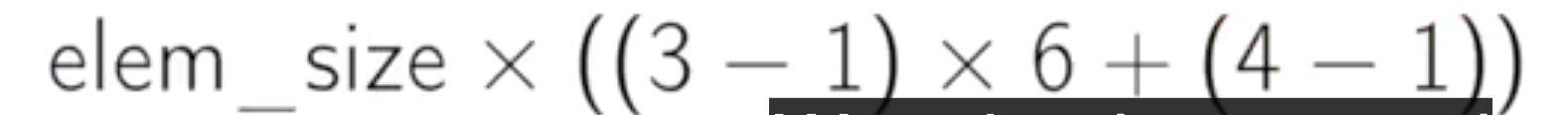
We take the column index and subtract initial column index for it. Here it is 1.



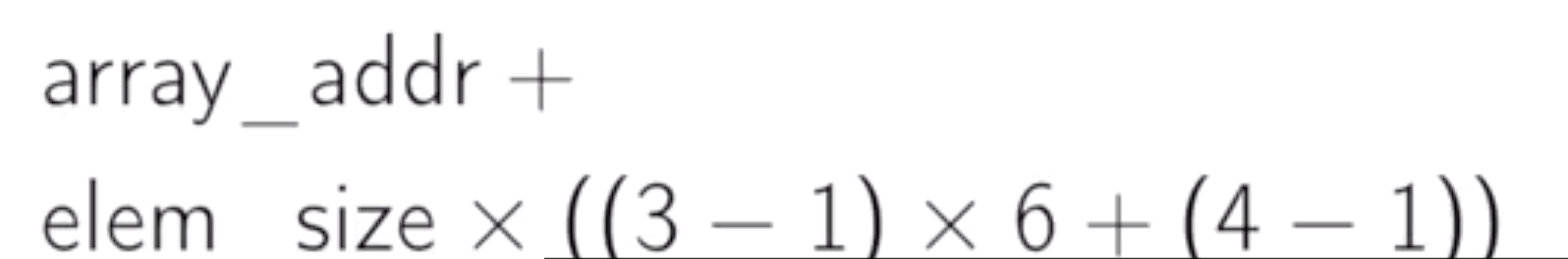


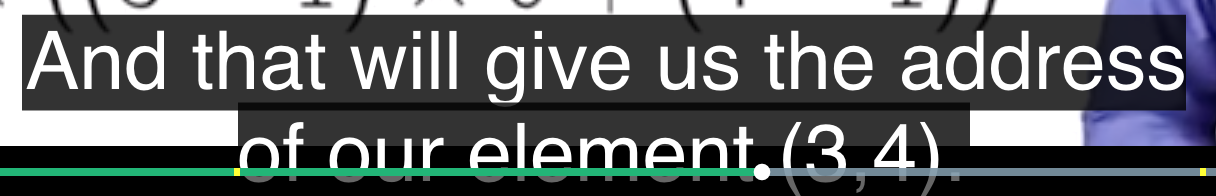
Total is 15.

We take that 15 and multiply with our element size.



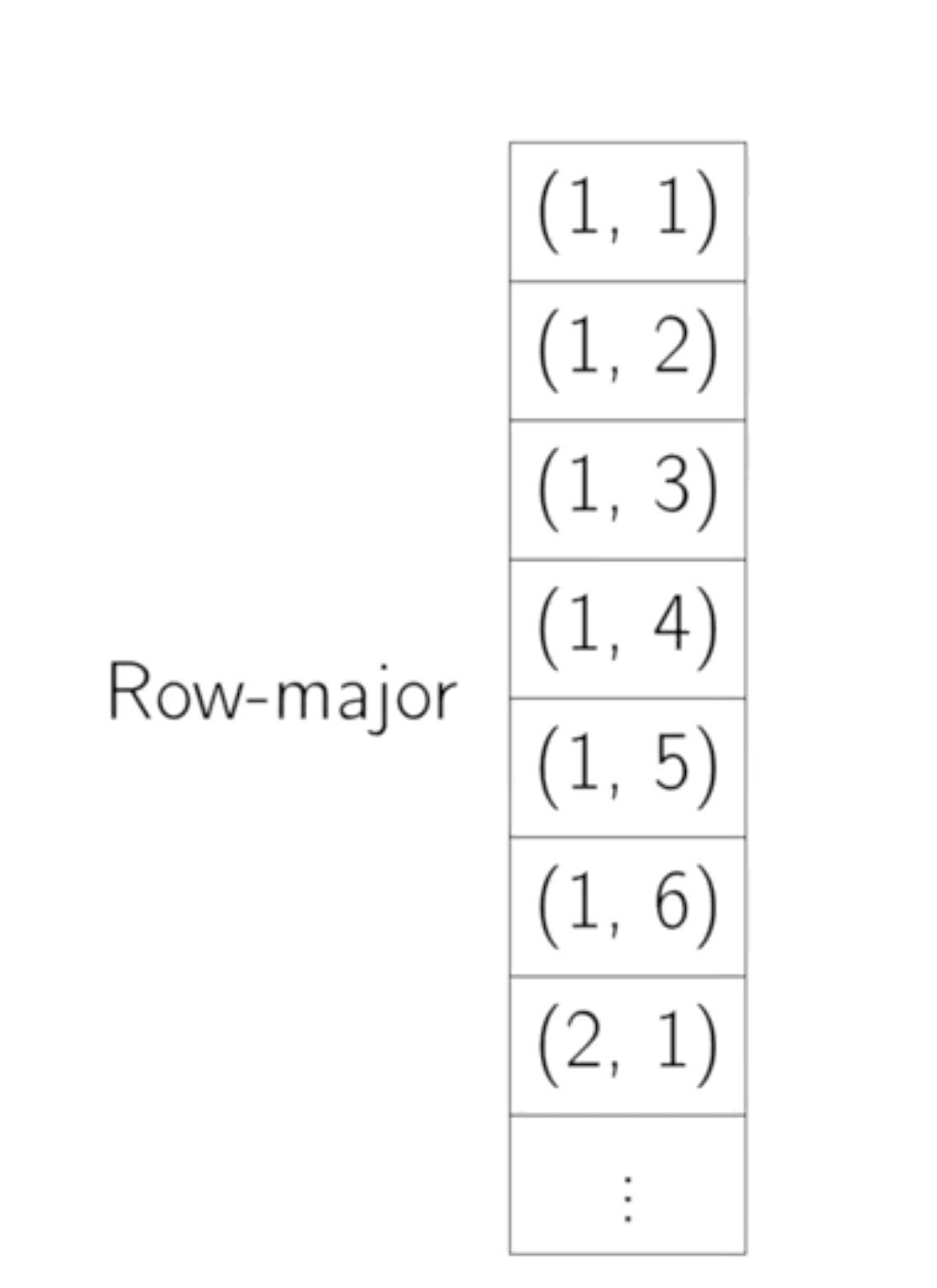
And then add it to our array address.





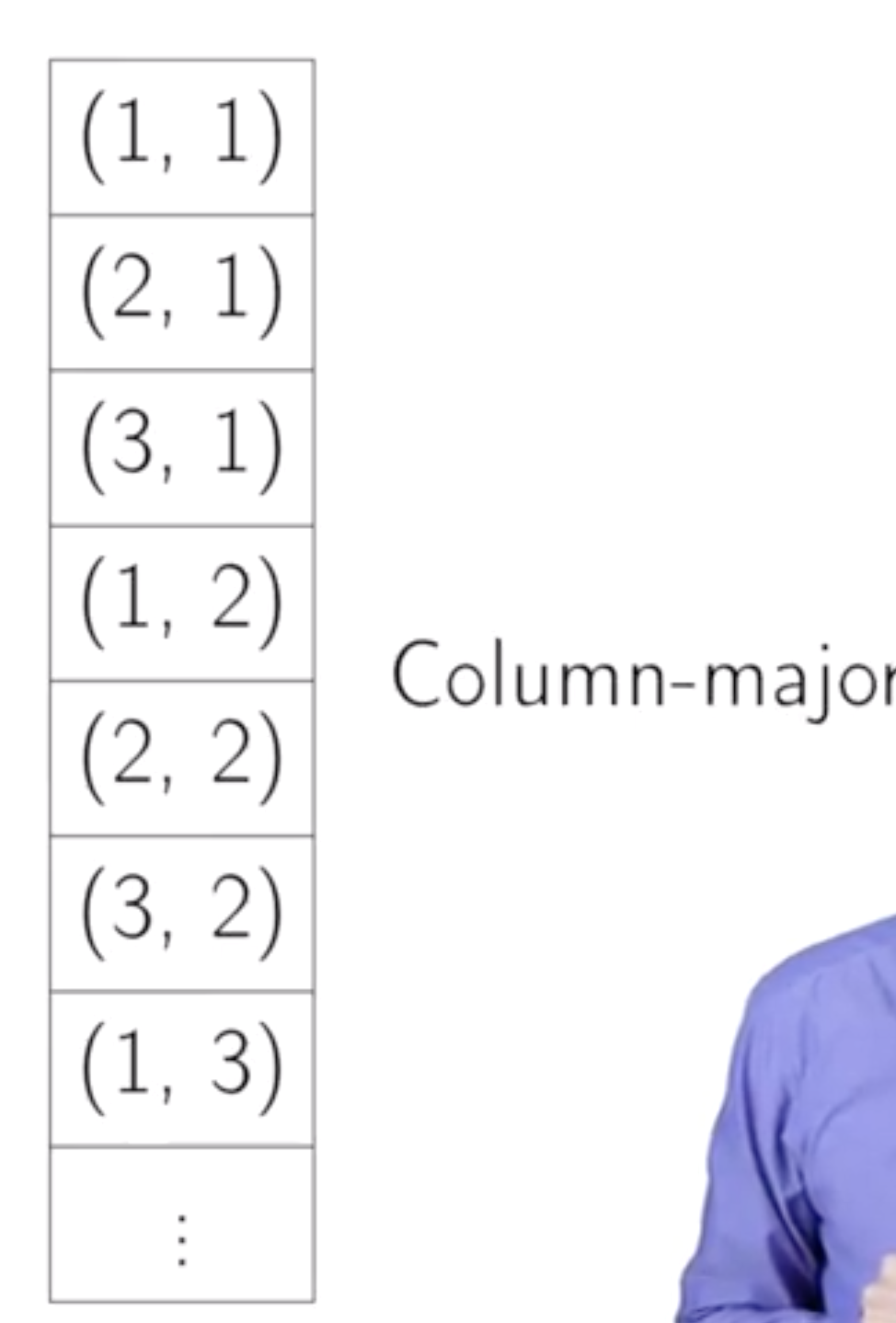
So this is, we laid out all the elements of the first row followed by all the elements of the second row and so on. This is called row-major ordering or row-major indexing.

Below is the image of how elements get stored in memory.

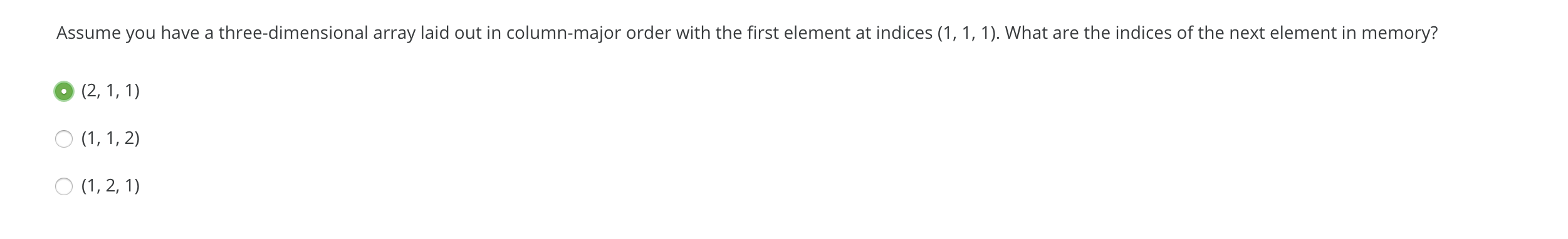


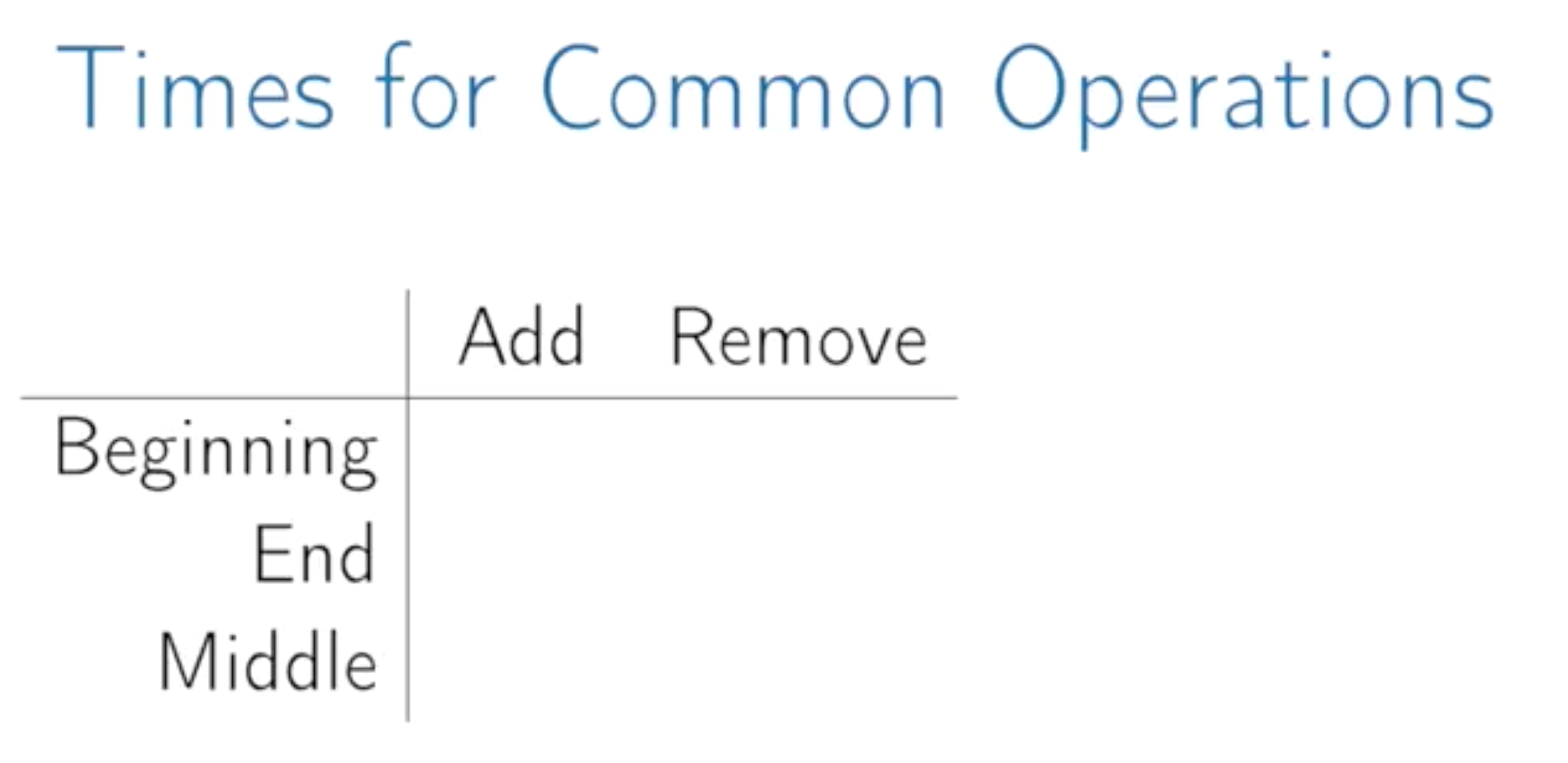
Here the column index is changing more rapidly. So this row-major.

Some compilers will actually do like, they lay out each column in order so you will get first column followed by second column and so on.

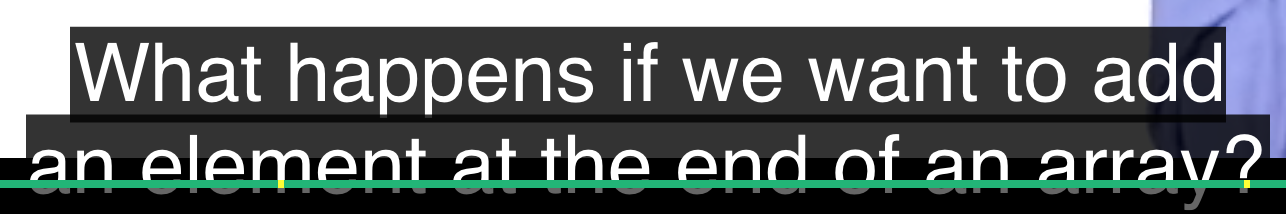


Here the row index is changing rapidly. So this column-major.

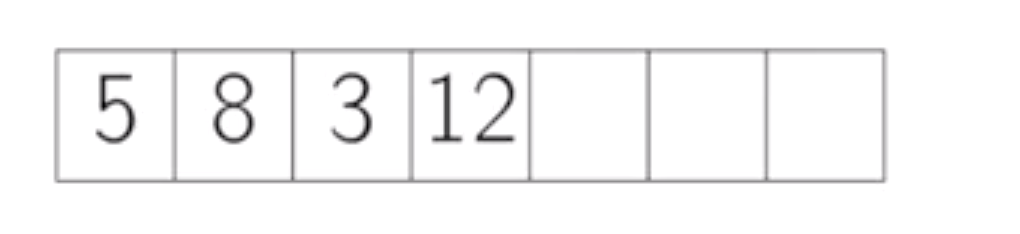




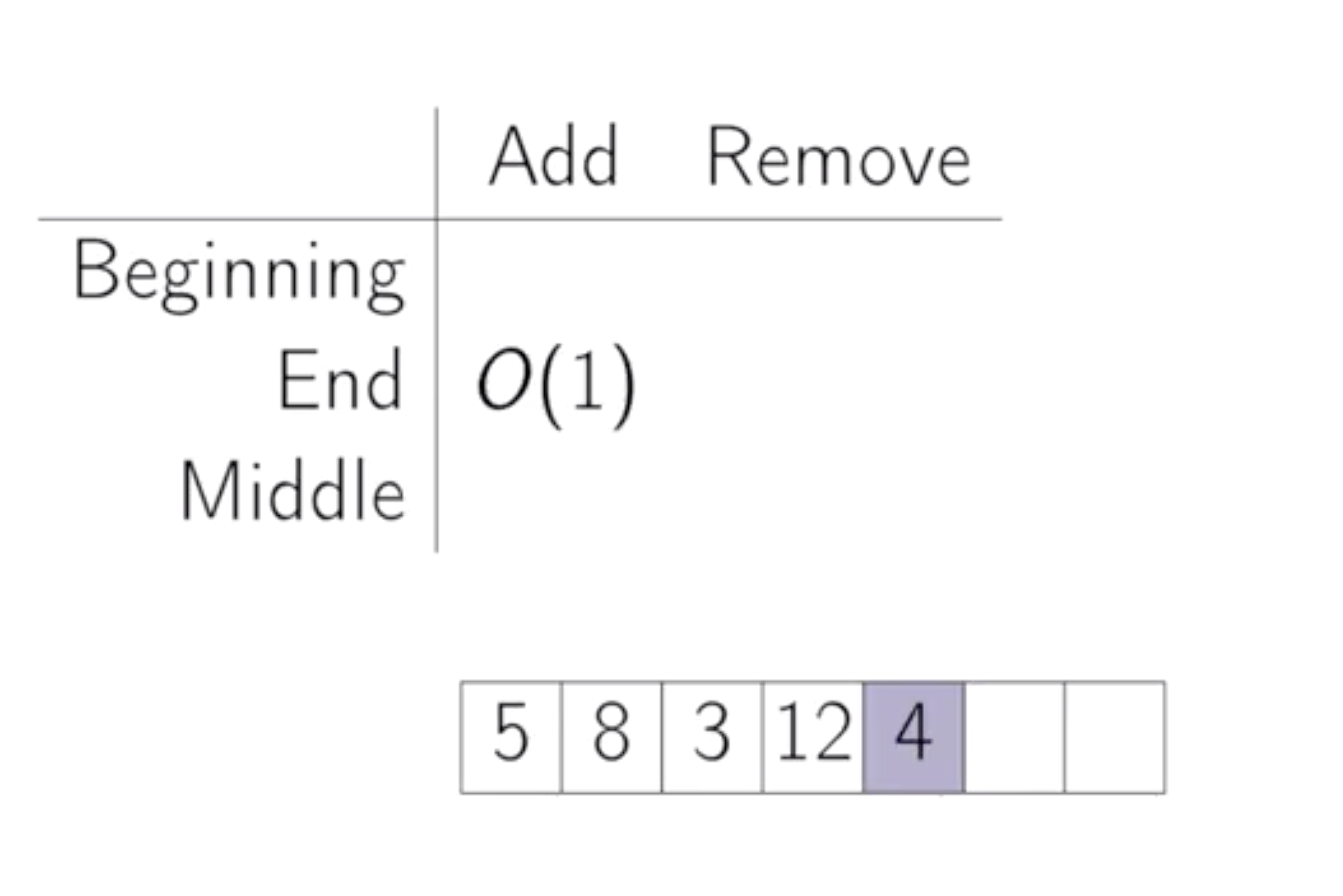
We already said that to read any element is O(1) and to write any element is O(1). This is standard feature of arrays.



Lets say we have allocated 7 elements for an array. We are only using 4 of them as below.

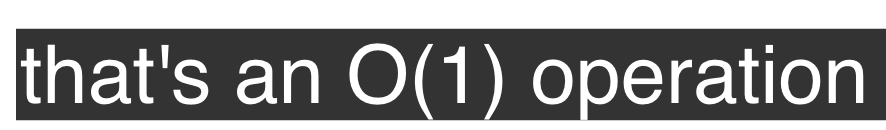


Now we want to add 5th element. Anyway there is room for 7. So just add it.

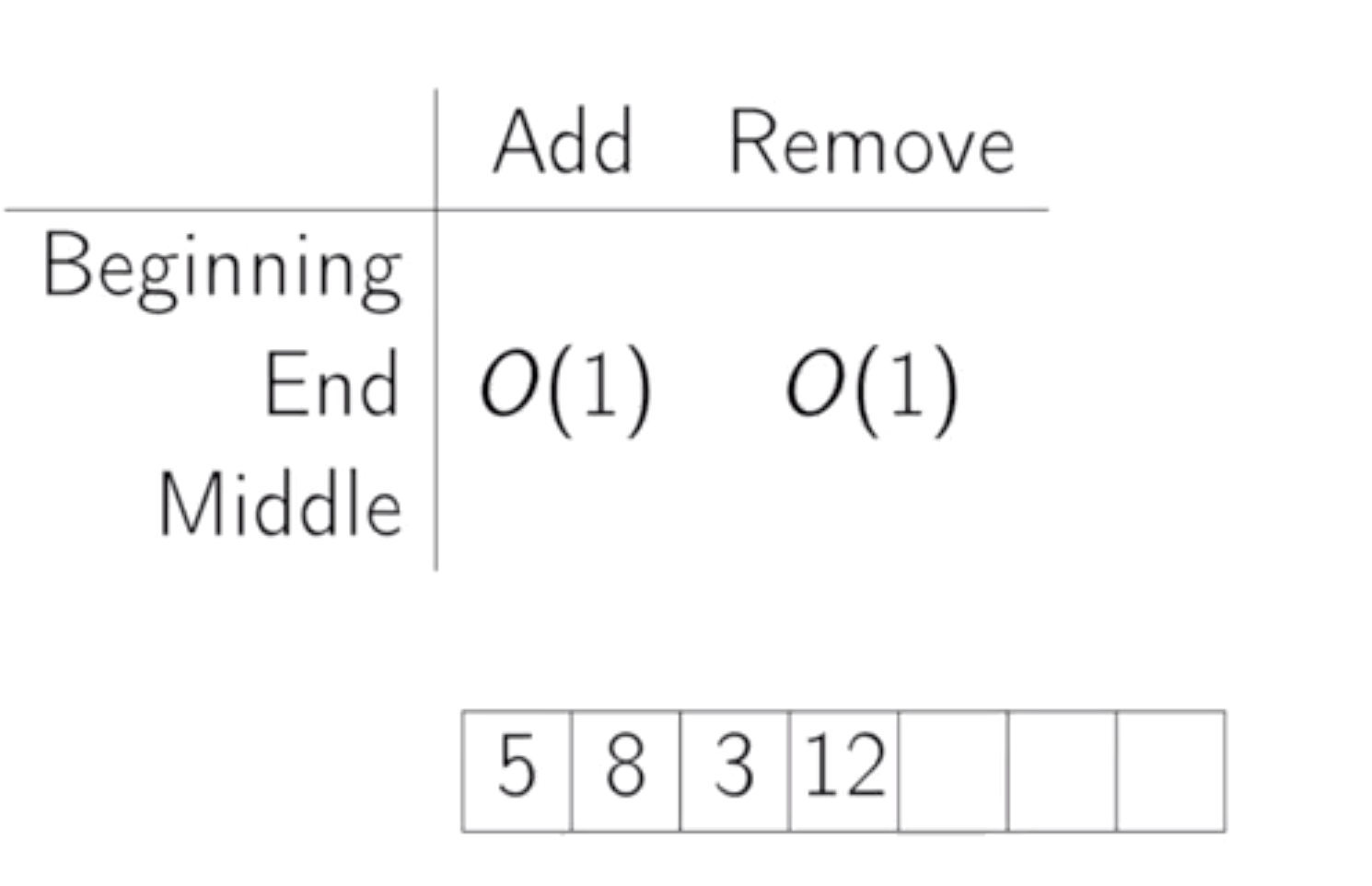


Now update the number of elements that are in use. This is an O(1) operation.





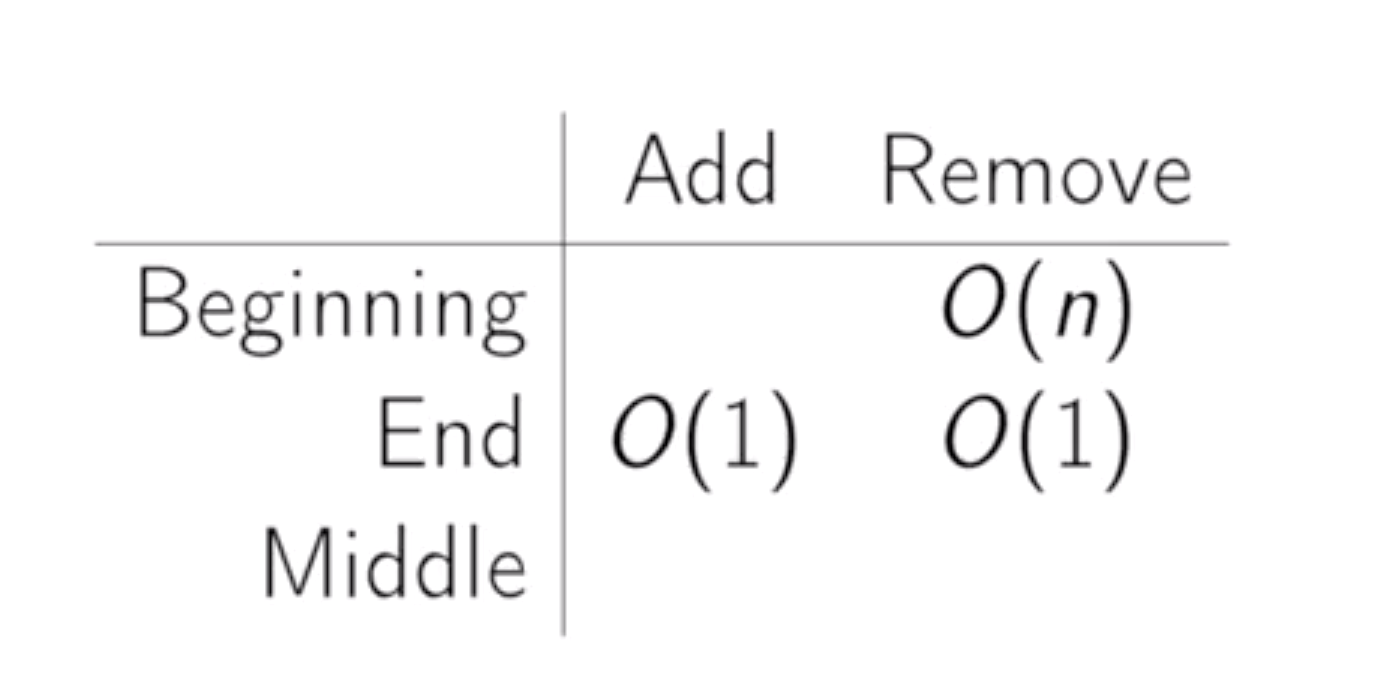
Because we just update the number of elements that are in use.



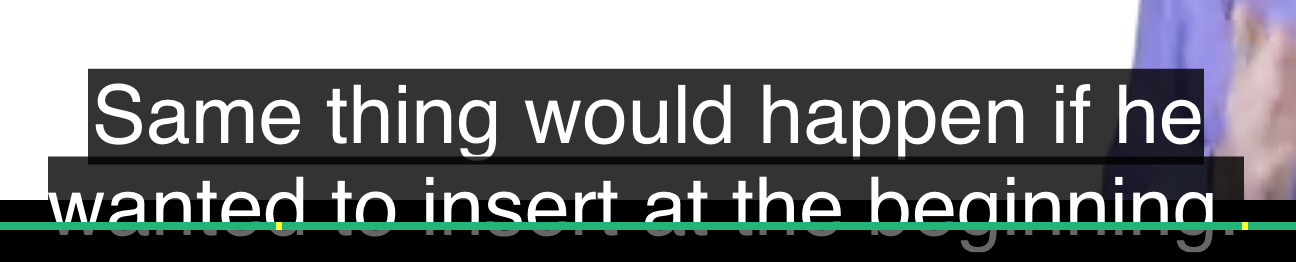
Now if we want to remove the first element,

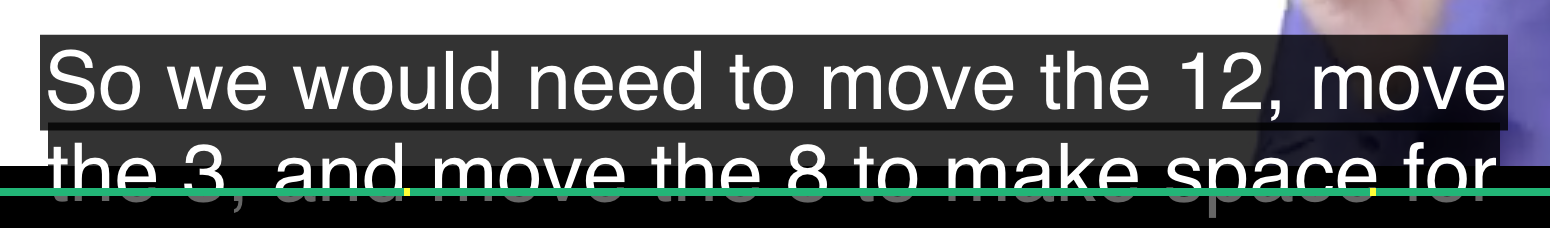
So we remove the 5 here. We don’t want a hole to be left there.



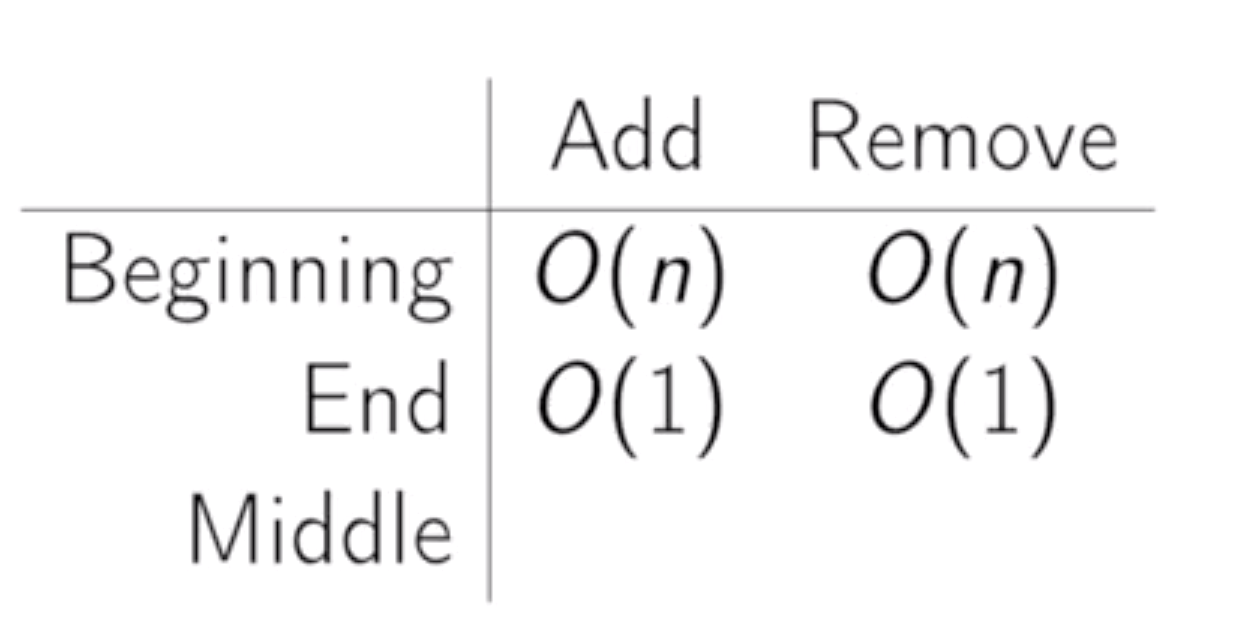


That’s an O(n) operation.

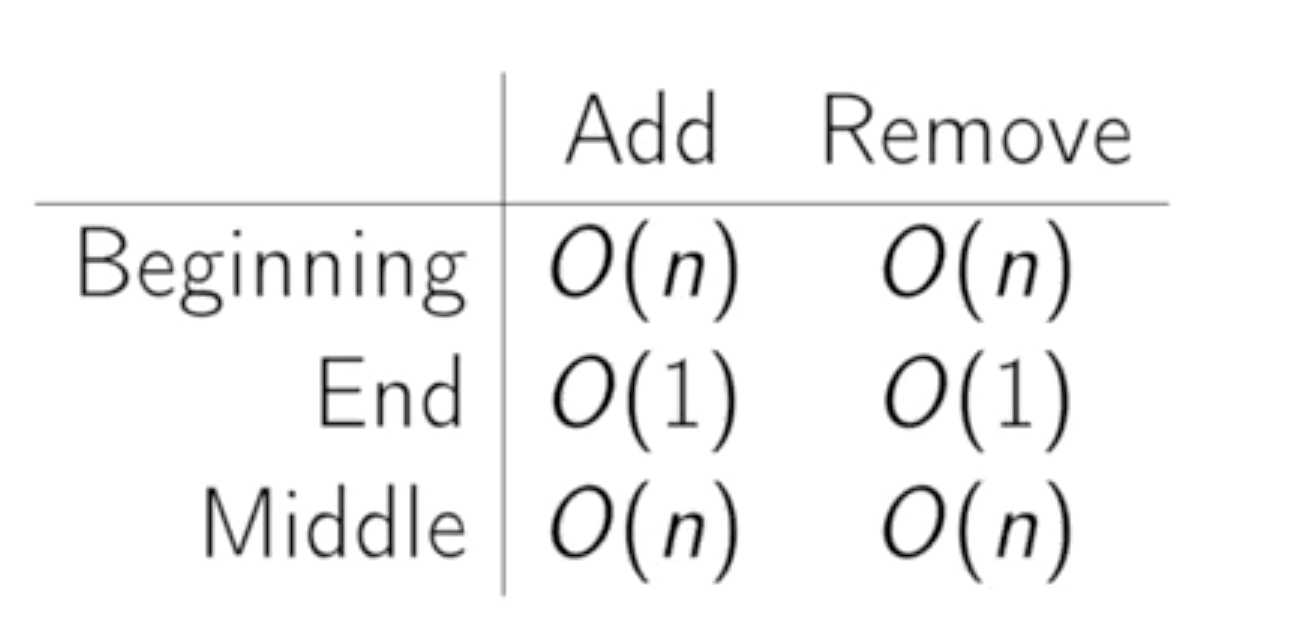




our new element. So that also O(n).



Similarly if we want to add or remove somewhere in the middle, this is also an O(n) operation.



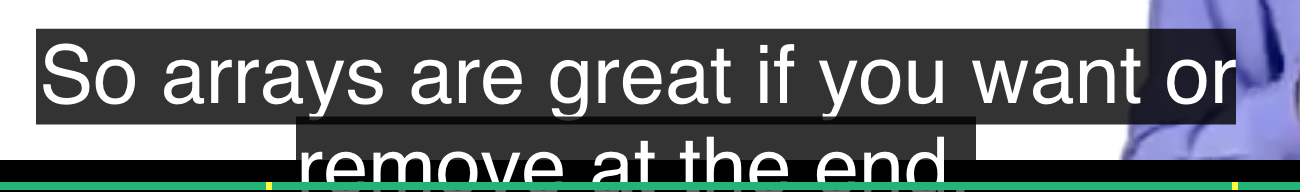


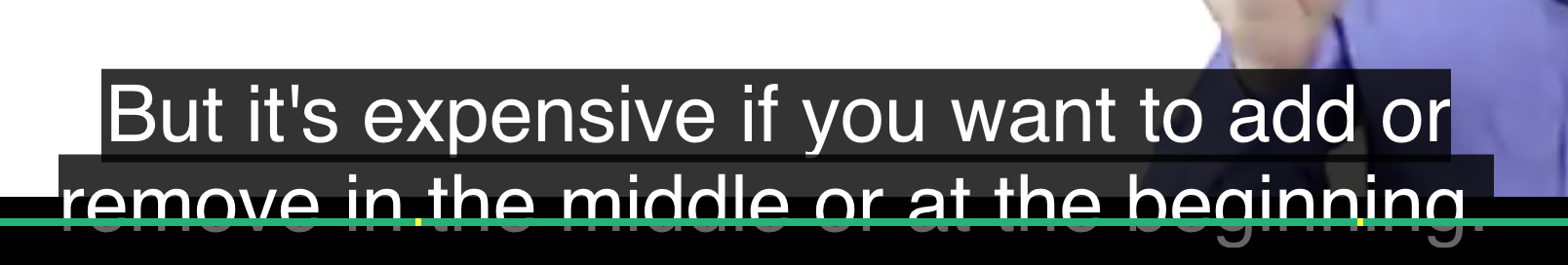


n/2 is n over 2.



Same for removal in middle.





But the huge advantage is, we have constant time access to elements either read or write.

