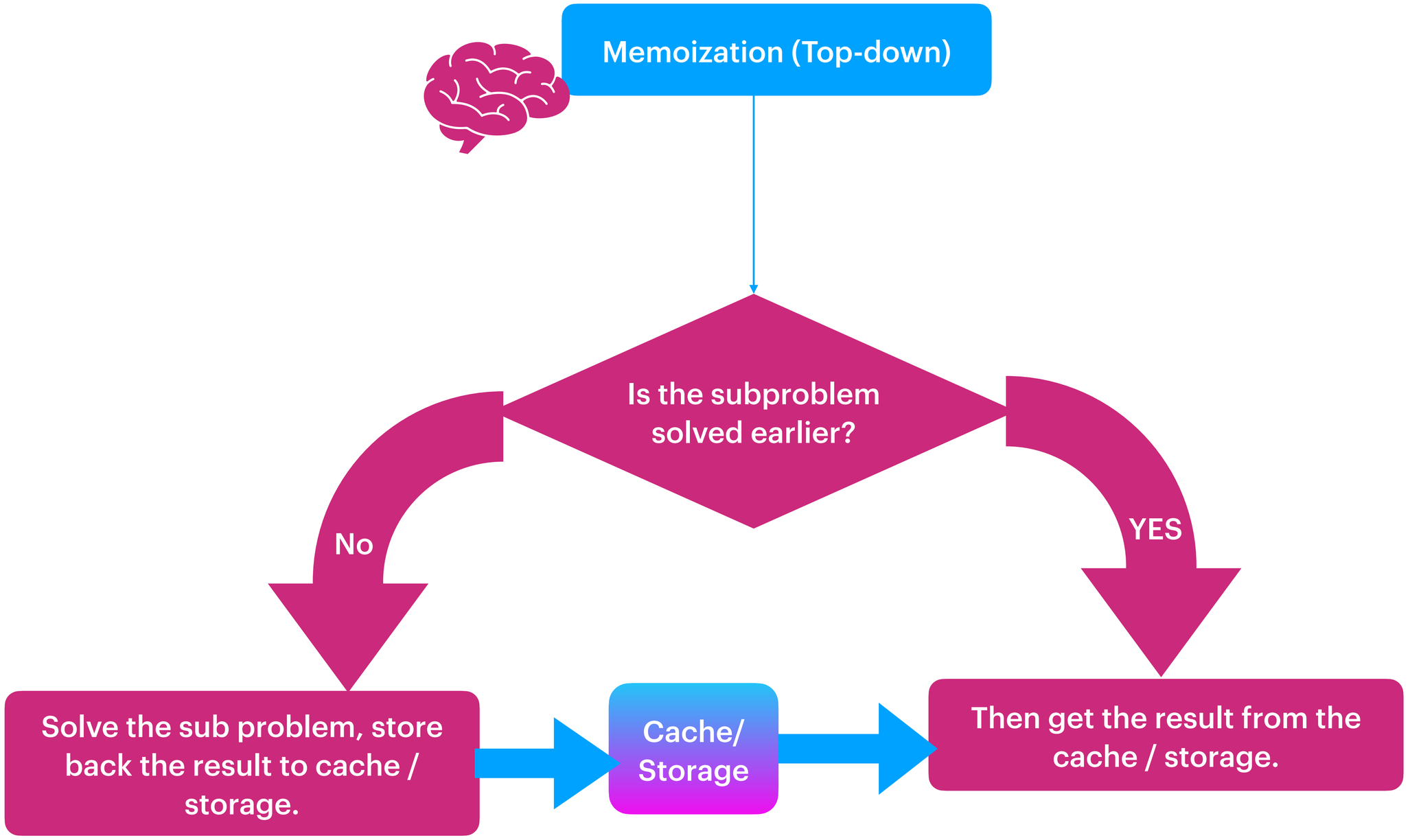

Memoization (Top-down)
Is the subproblem solved earlier?
No
YES
Solve the sub problem, store back the result to cache / storage.
Cache/ Storage
Then get the result from the cache / storage.

Memoization (Top-down)

How to identify
the sub problems for caching
/ storing?

Have a proper knowledge on the data structure design, observe what are the parameters are being changed & encouraging us to move to all possible sub problems.

Hint : In simple what ever parameters we use in base condition of recursion, can be used to identify the caching / storing.

Memoization (Top-down)

0/1 Knapsack Problems

For the 0/1 Knapsack problems, there are two parameters, can encourage us to move forward with possible sub problems.

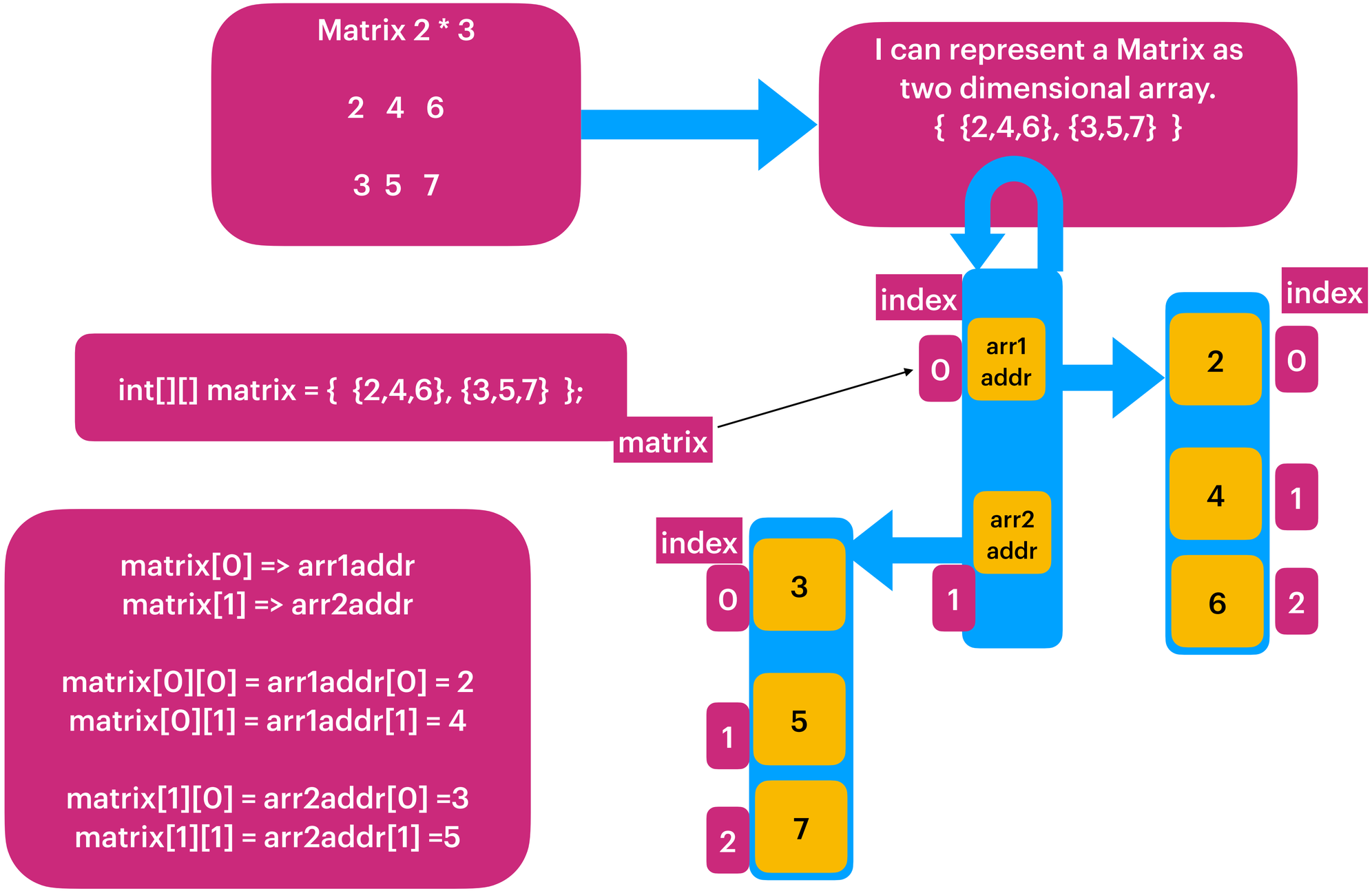
1. currentIndex (moving from index 0 to n-1)
2. capacity (Moving from capacity value “c” to 0)

So finally we can have a storage/caching on these two variable for all possible combinations.

All possible combinations? How ?

Matrix is the place where we can design all possible combinations.

We can represent a matrix with two dimensional array.


Matrix 2 * 3
2 4 6
3 5 7
I can represent a Matrix as two dimensional array.
{ {2,4,6}, {3,5,7} }
int[][] matrix = { {2,4,6}, {3,5,7} };
matrix
index
0
arr1 addr
1
arr2 addr
index
0
2
1
4
2
6
index
0
3
1
5
2
7
matrix[0] => arr1addr
matrix[1] => arr2addr
matrix[0][0] = arr1addr[0] = 2
matrix[0][1] = arr1addr[1] = 4
matrix[1][0] = arr2addr[0] =3
matrix[1][1] = arr2addr[1] =5

Algorithm For 0/1 Knapsack Memoization

design a matrix with currentIndex and capacity.

Base Condition in Recursion :
1. As we are moving from index 0 to n, the max possible valid value for currentIndex = n-1
2.As and when we add item to the knapsack, we reduce the capacity so the smallest possible value for capacity is '0'.

int[][] dp = new int[n][capacity+1];

Which is representing currentIndex. Index starts from 0 to n-1

Should be capacity+1 as would like to have a combination storage on index & capacity .

Check in matrix, does this combination of (index, capacity) sub problem is already been solved?

YES

Just return the result from the matrix.

No

Solve the sub problem for the current combination of index & capacity. Store back the result to Matrix.

Current Item Weight

When currentItem weight is less than or equals to capacity.

weights[currentIndex] <= capacity

When currentItem weight is greater than capacity.
weights[currentIndex] > capacity

Find profit1 with including item.
Find profit2 without including item.
Return max of profit1, profit2

Find profit without including item.
return the profit.