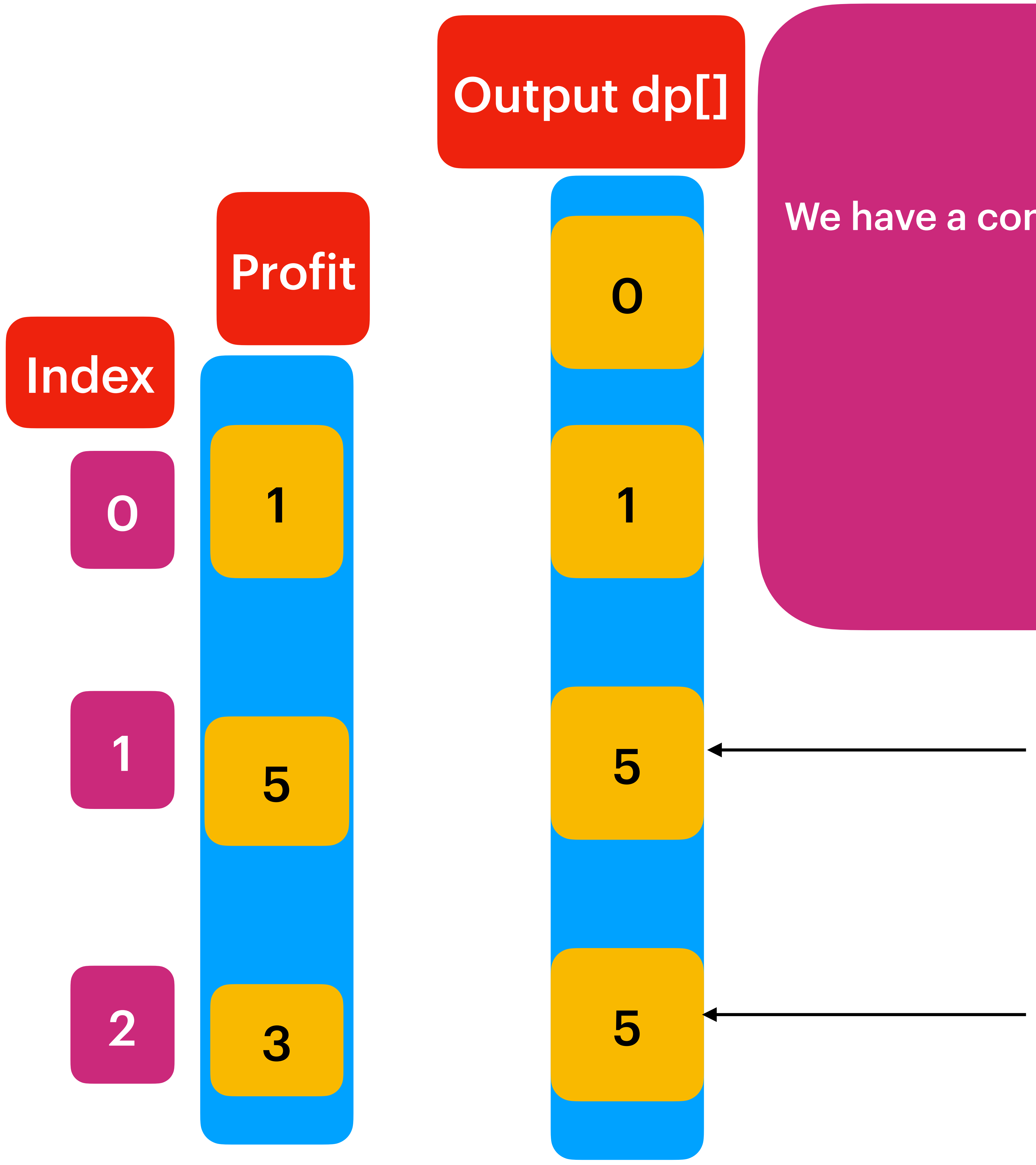
Actual Principle is =>
dp[i] = max(profits[i] +dp[i-2] , dp[i-1])

We have a constraint here array size is aways less than the dp size. Leads to index OutOf Bounds Exception.

So track this way
Current 'i' element profit would be stored in dp[i+1].

So principle is
dp[i+1] = Math.max(profits[i]+dp[i-1],dp[i]);



max(includeProfit, excludeProfit)

max(5+0,1) = 5

max(includeProfit, excludeProfit)

max(3+1,5) = 5

```
dp[0] = 0;
dp[1] = profits[0];
// When there is single House , we just take profit[0].

// Current 'i' element profit would be stored in dp[i+1]
for(int i = 1 ; i < profits.length;i++)
{
dp[i+1] = Math.max(profits[i]+dp[i-1],dp[i]);
}
```

Back Tracking ::
int[] profits = {1,5,7,8,11,10,3,22};
We know that profit[i], could be represented by dp[i +1].

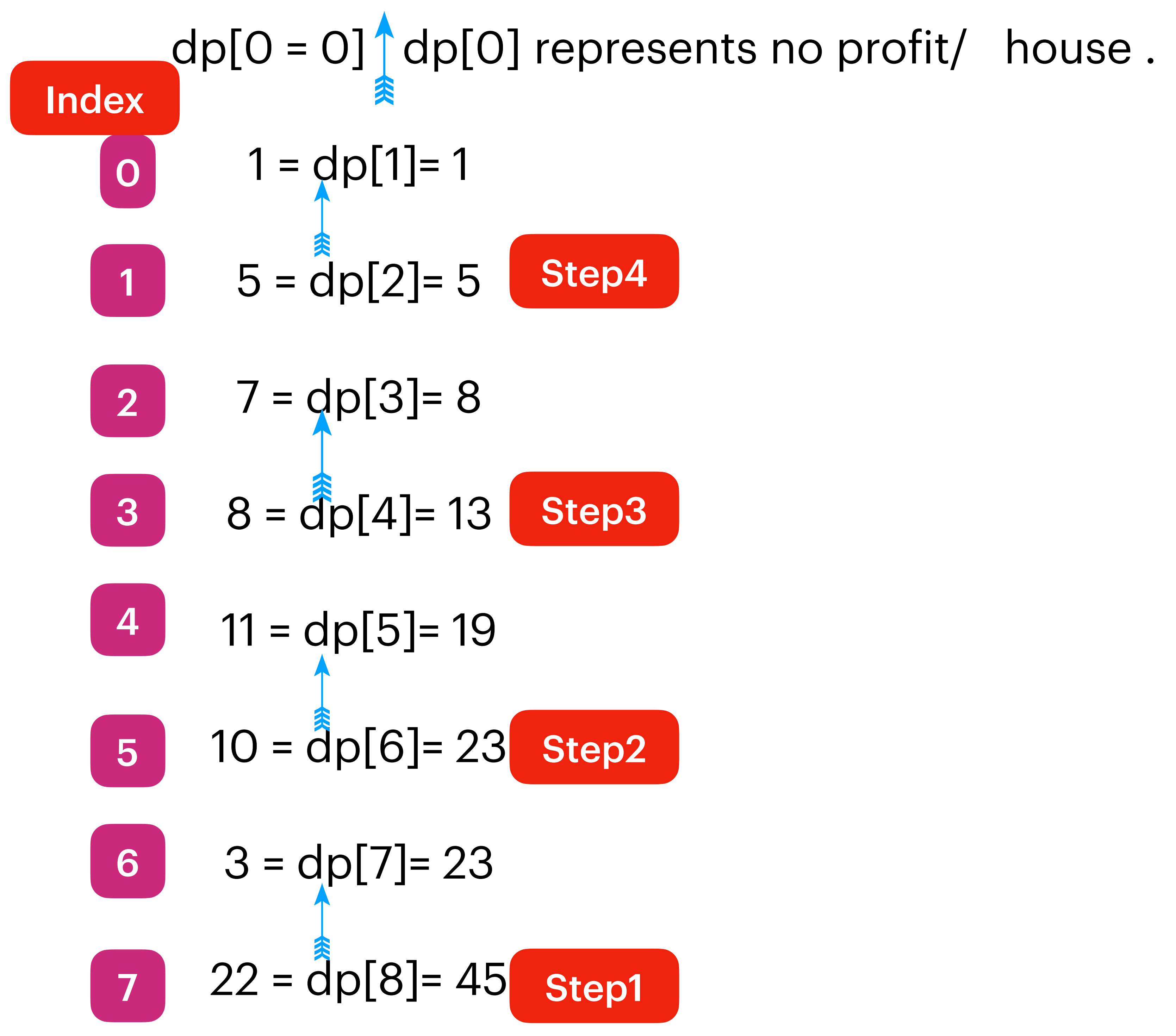
So that in BackTracking we can say
dp[i] can be represented by profits[i-1] .

When dp[i] != dp[i-1],it means you included current element i.e profits[i-1].
So update the totalProfit , move the index to i-2.
Because by including current element you got the MaxProfit, look back to other possibilities in same direction.

When dp[i] == dp[i-1],it means you did not includ current element so just move to previous row. “ i - -“.

```
int[] profits = {1,5,7,8,11,10,3,22};
```

```
dp[i+1] = Math.max(profits[i]+dp[i-1],dp[i]);
```



dp[0 = 0] dp[0] represents no profit/ house .

Index

0 1 = dp[1]= 1

1 5 = dp[2]= 5 Step4

2 7 = dp[3]= 8

3 8 = dp[4]= 13 Step3

4 11 = dp[5]= 19

5 10 = dp[6]= 23 Step2

6 3 = dp[7]= 23

7 22 = dp[8]= 45 Step1

MaxProfit = 45
Selected Profits :{ 22 , 10 , 8 , 5}