

Algorithm efficiency can be calculated based on Processing time and Space It's occupied .

RAM



Cluster

It's a virtual network integrates Group Of Machines.

30GB

RAM



Machine1

RAM



Machine2

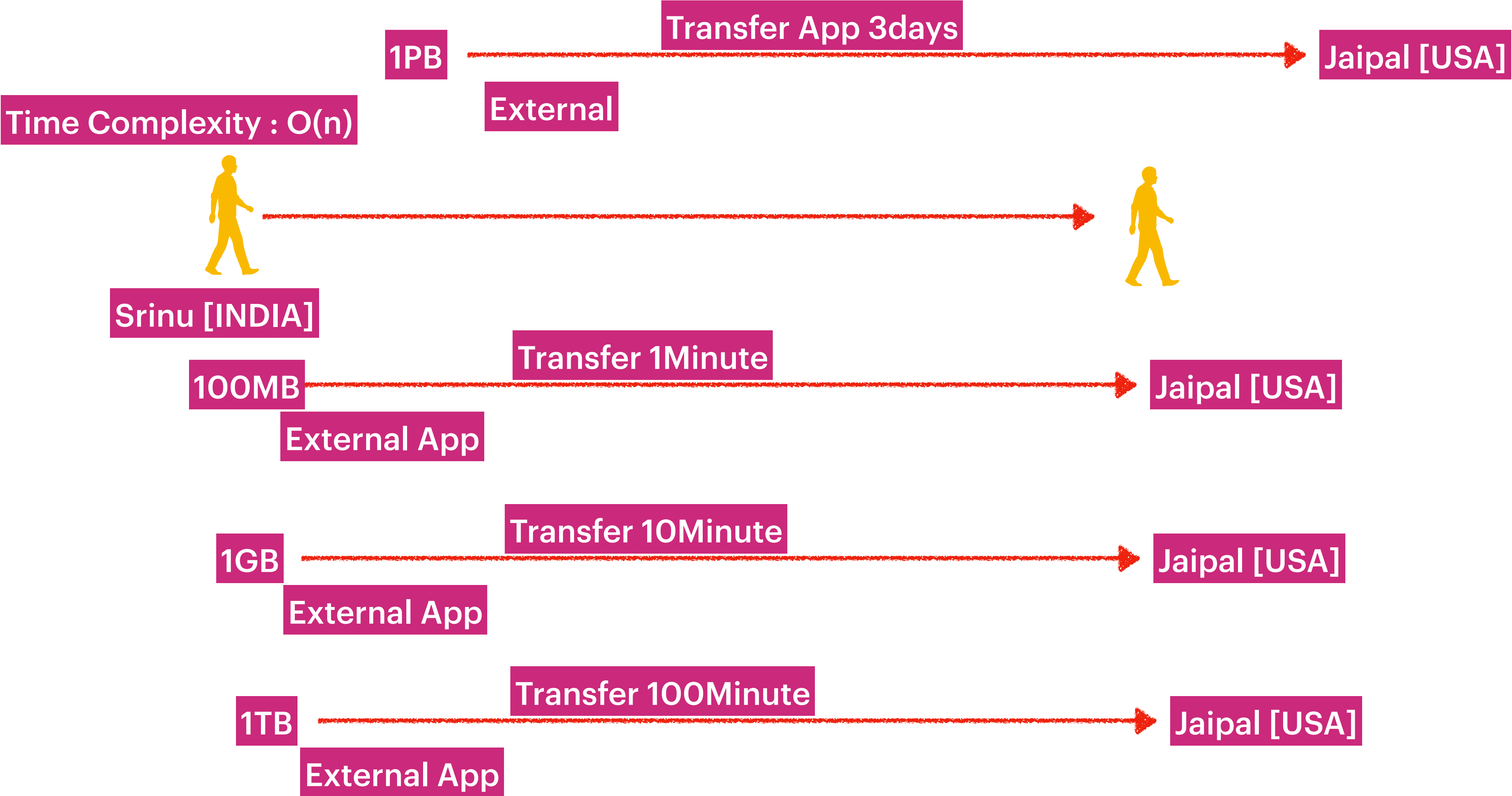
RAM



Machine3

Processing Time → Time Complexity

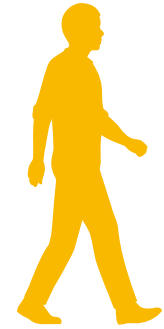
Space → Space Complexity



Processing Time → Time Complexity

Space → Space Complexity

Time Complexity : $O(1)$
Space Complexity : $O(n)$



Srinu [INDIA]

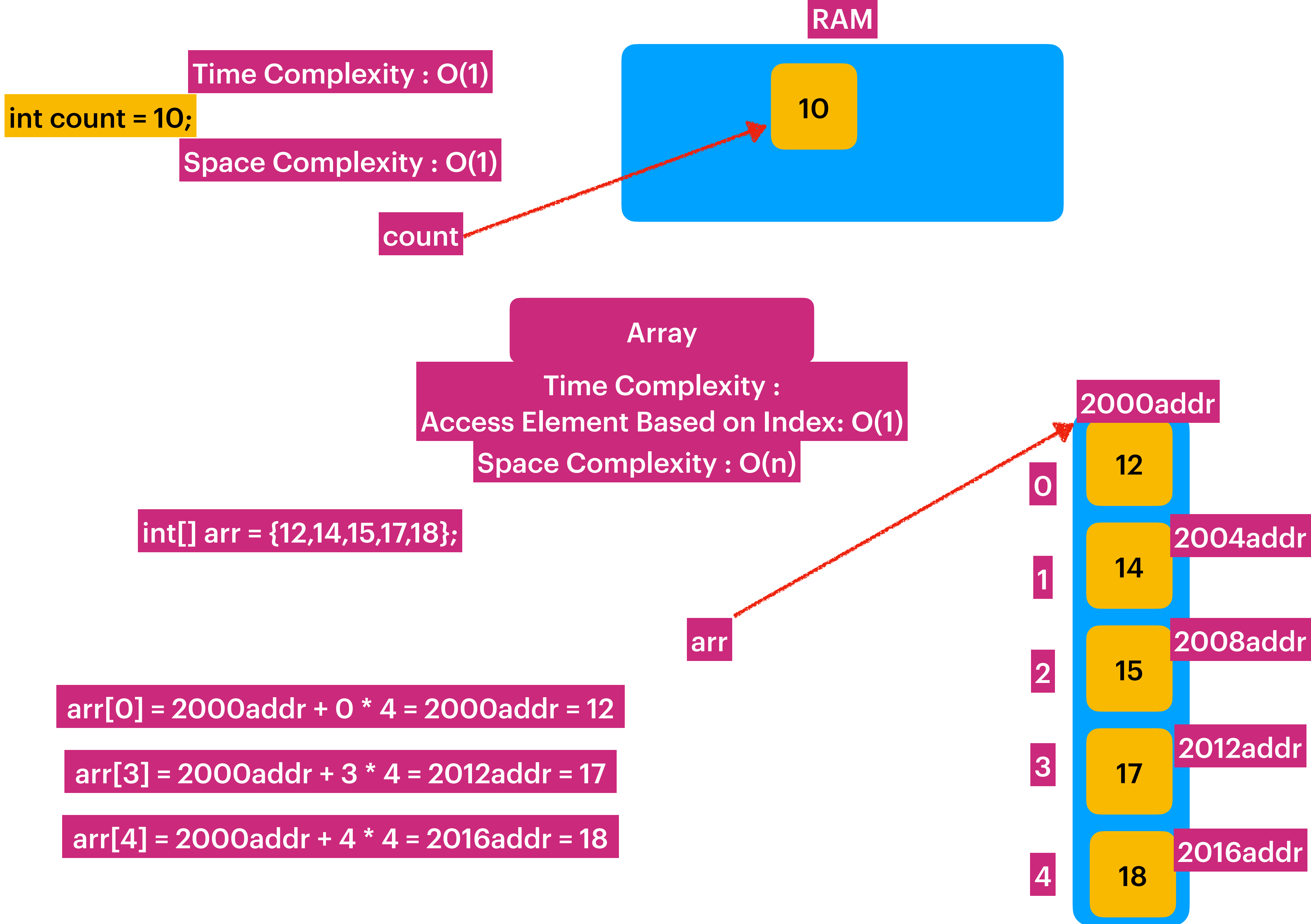


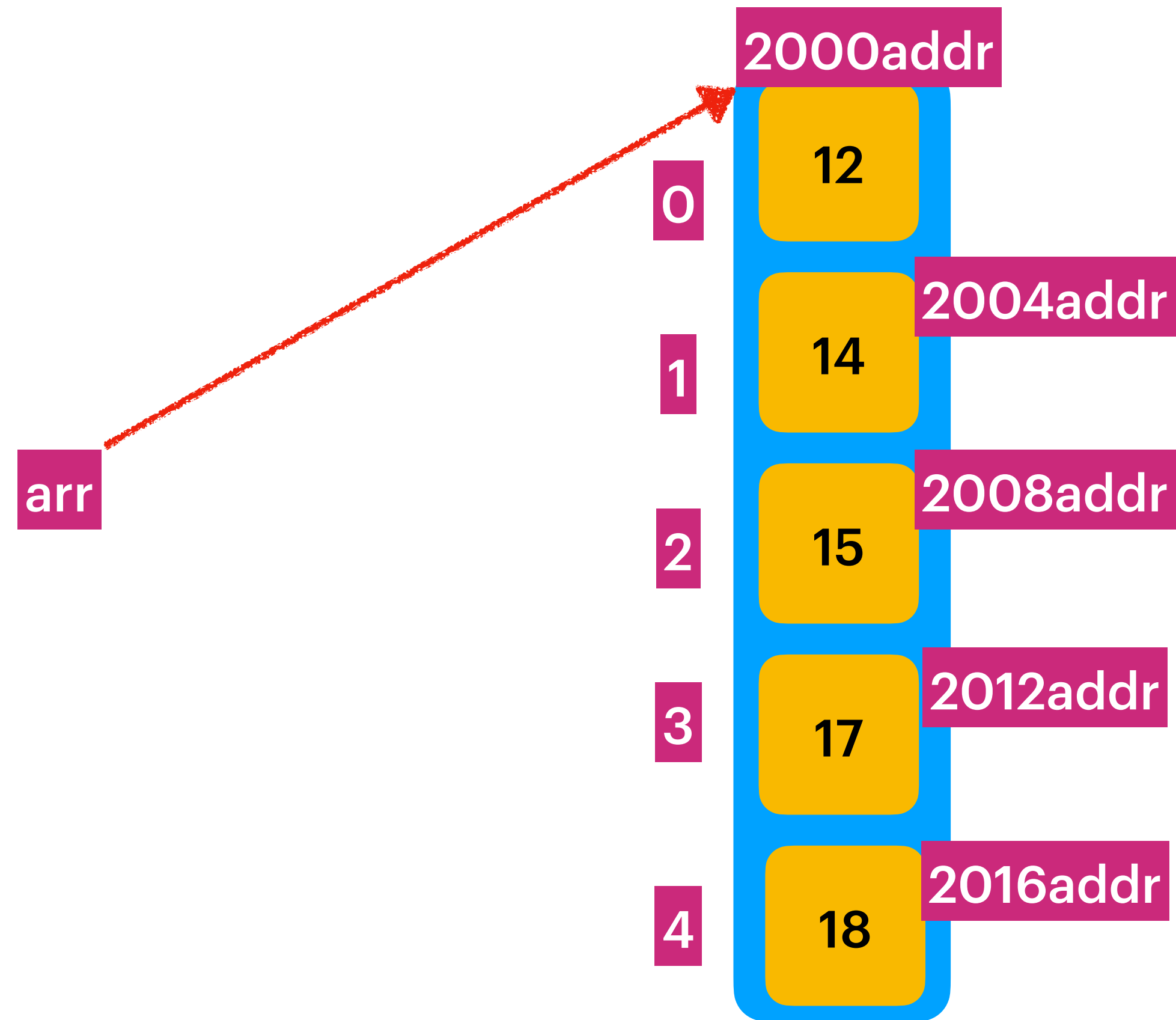
Jaipal [USA]

Flight : 7 Hours

1PB







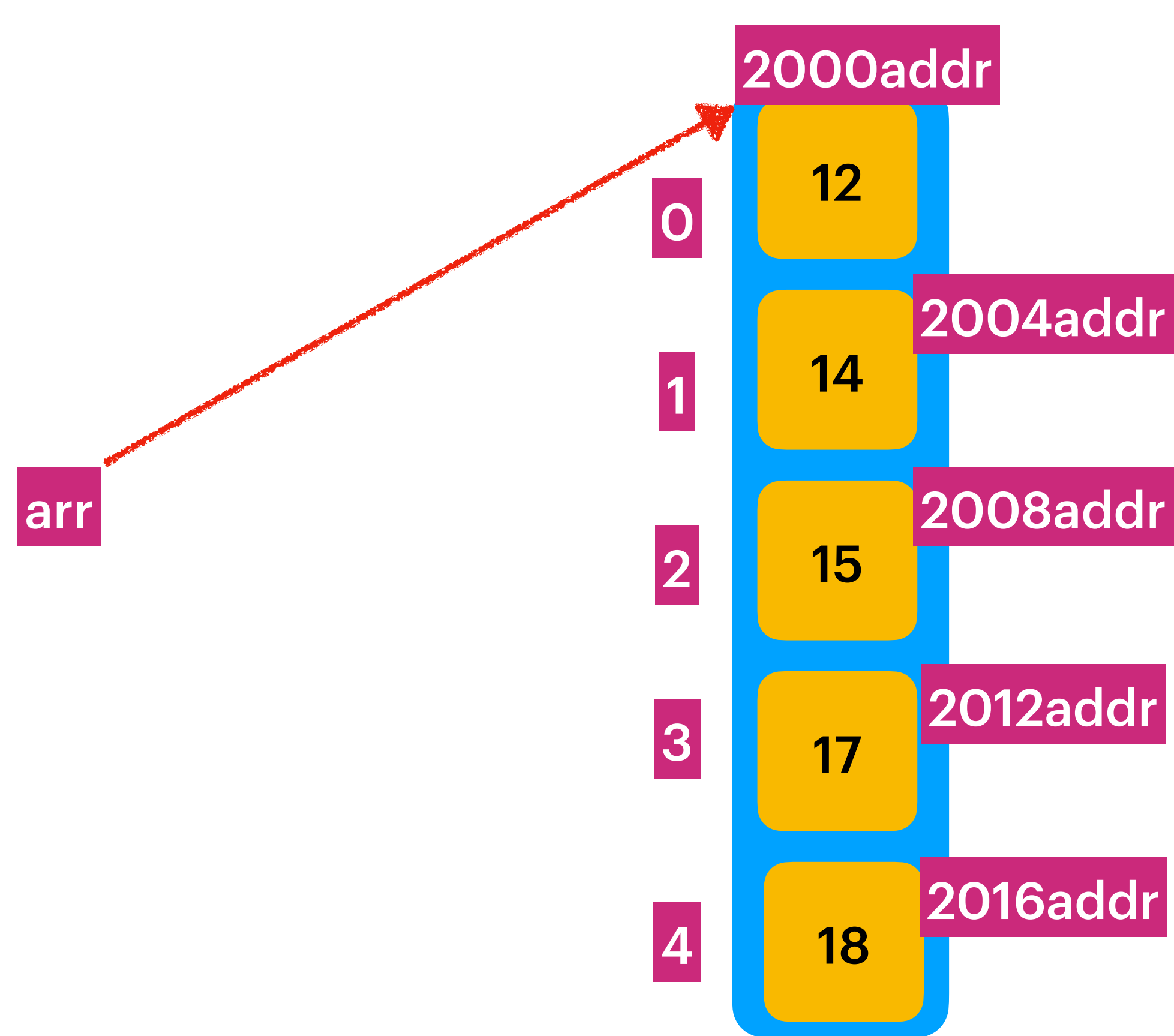
Search an Element in an Array :
Time Complexity : $O(n)$
Space Complexity : $O(1)$

```
SearchElement 10

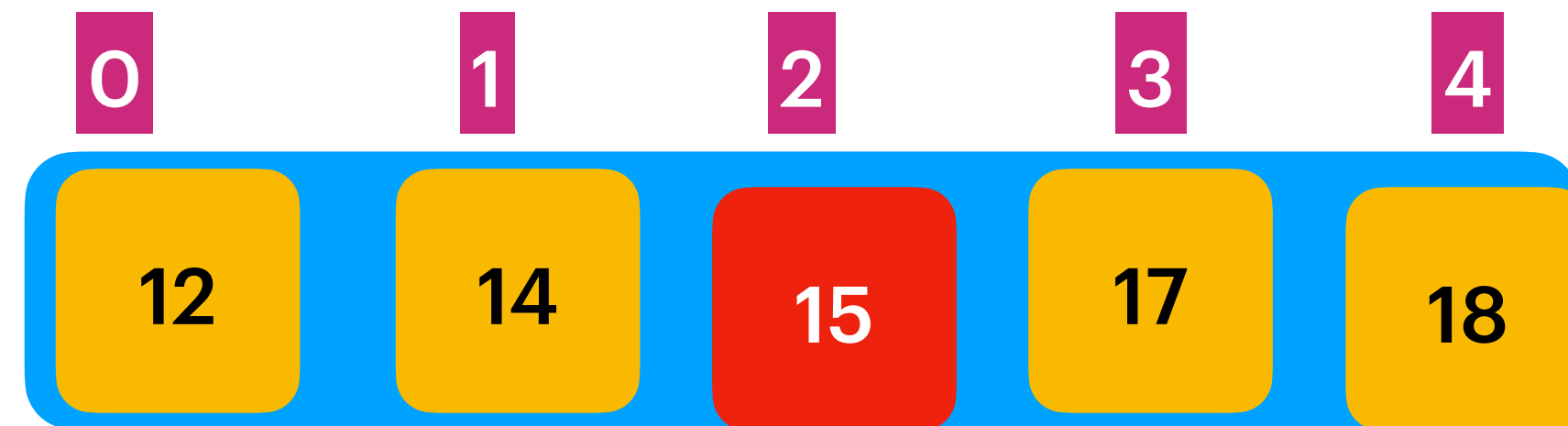
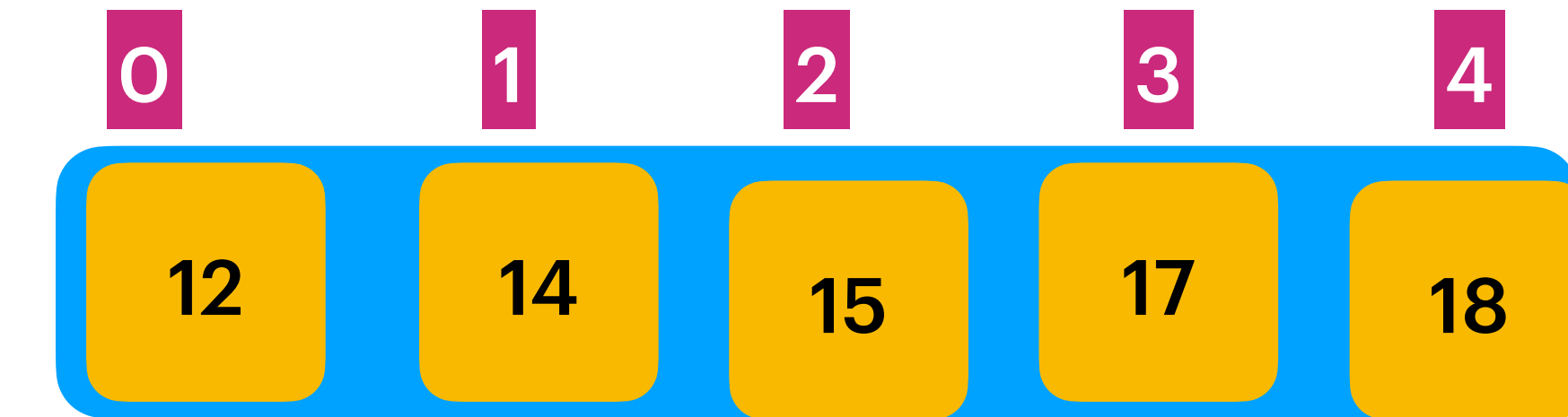
Time Complexity :  $O(n)$ 
Space Complexity :  $O(1)$ 

for(int i = 0; i < n ; i++)
{
    if(arr[i] == 10)
    {
        return true;
    }
}

return false;
```

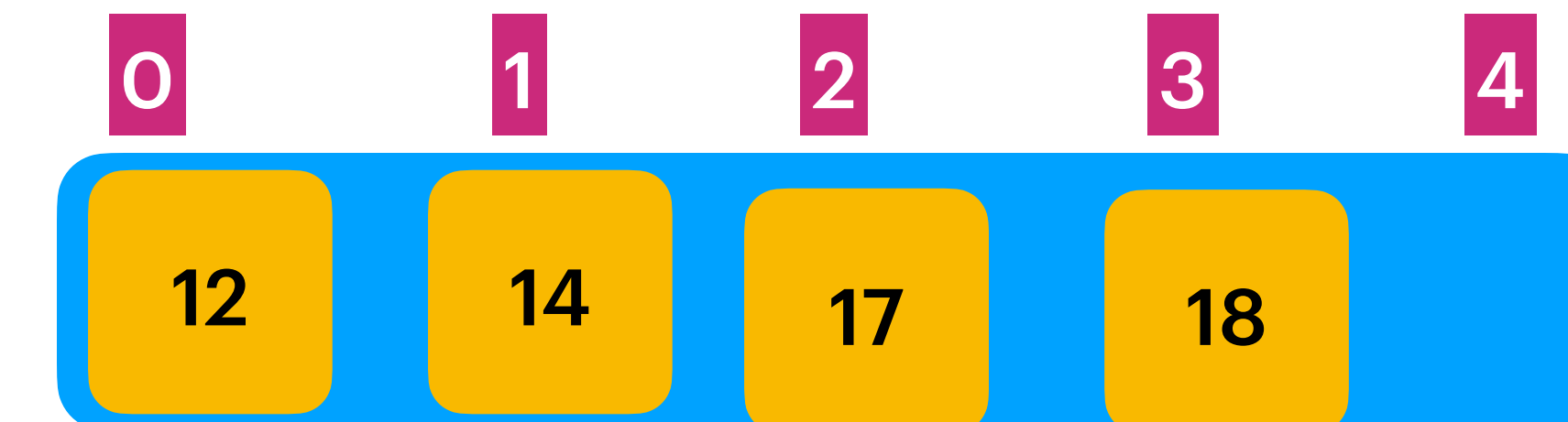


Delete an Element an Array :



Delete an Element an Array :
Search & Delete : $O(n)$
Shifting operation : $O(n)$
Space Complexity : $O(1)$

When we delete an element in an Array, we would need to Shift all the right index values before.



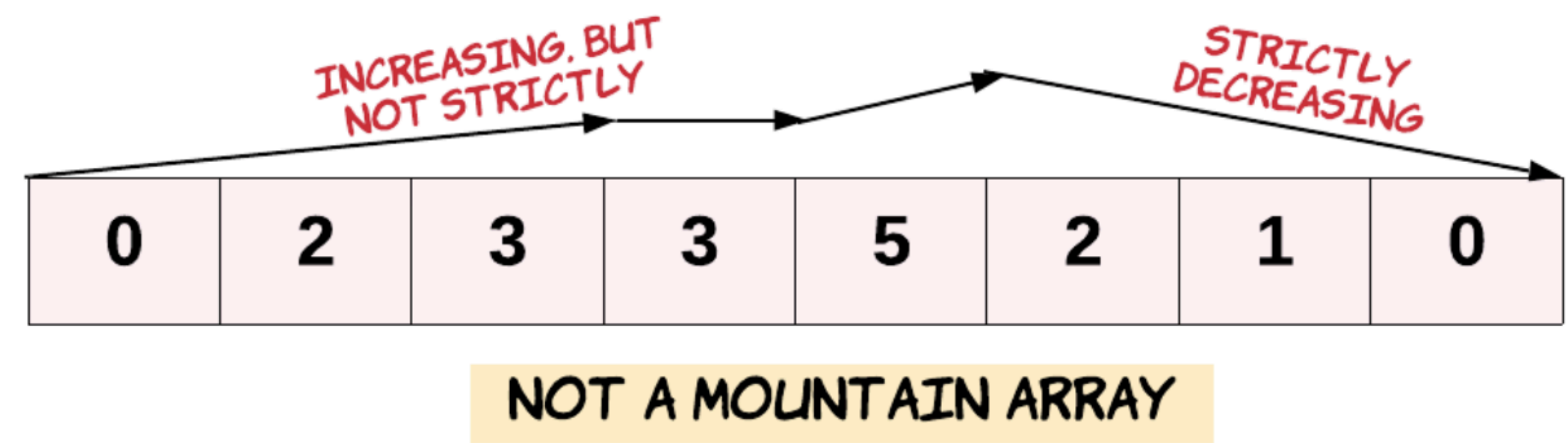
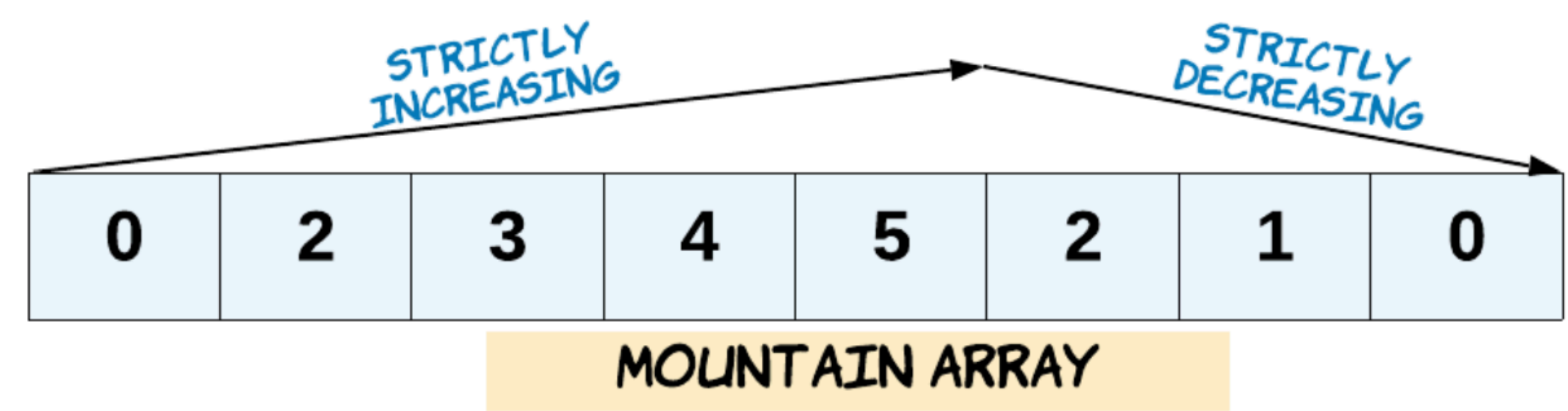
941. Valid Mountain Array

Easy 2073 145 Add to List Share

Given an array of integers `arr`, return `true` if and only if it is a valid mountain array.

Recall that `arr` is a mountain array if and only if:

- `arr.length >= 3`
- There exists some `i` with `0 < i < arr.length - 1` such that:
 - `arr[0] < arr[1] < ... < arr[i - 1] < arr[i]`
 - `arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`



Example 1:

Input: `arr = [2,1]`
Output: `false`

Example 2:

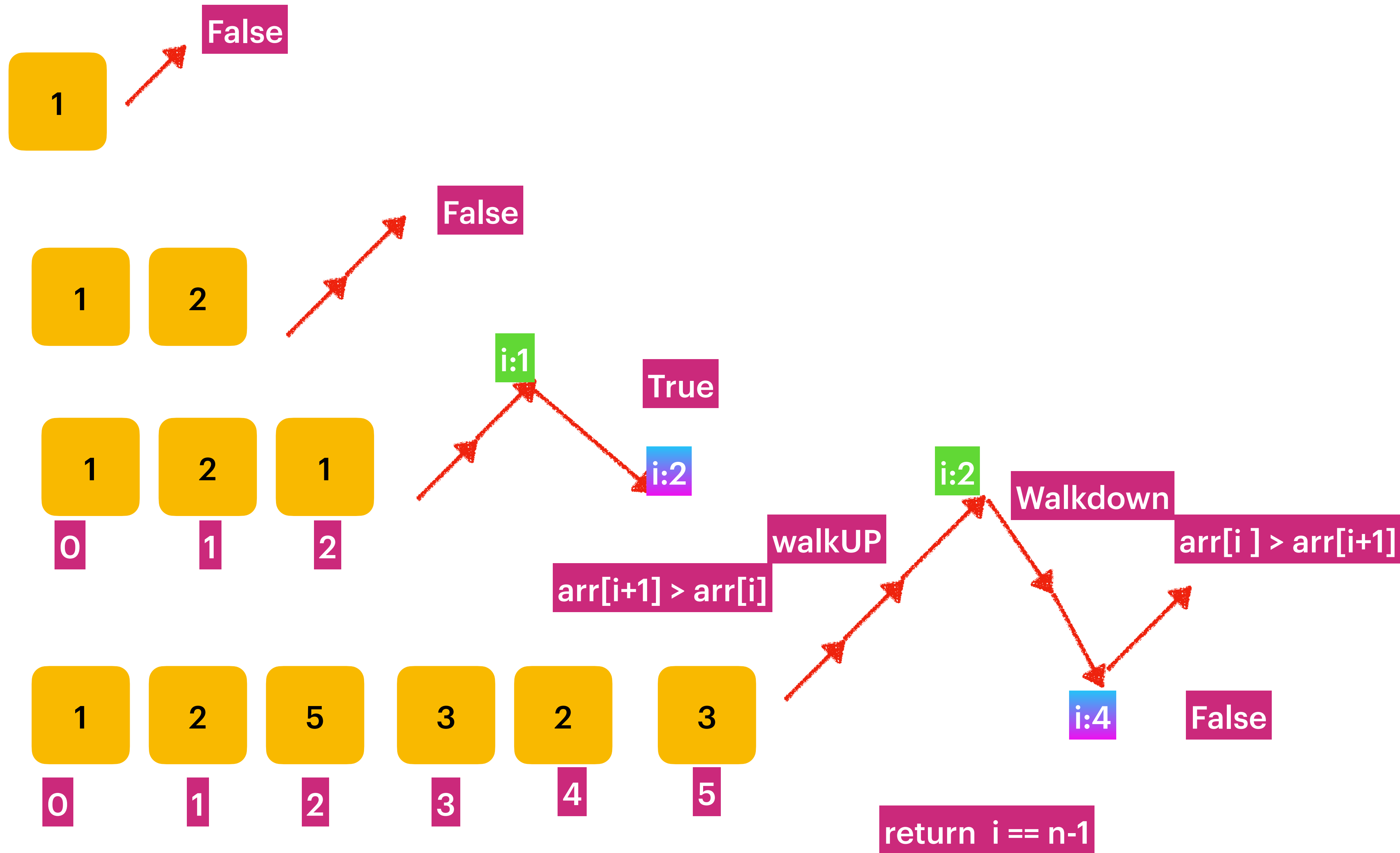
Input: `arr = [3,5,5]`
Output: `false`

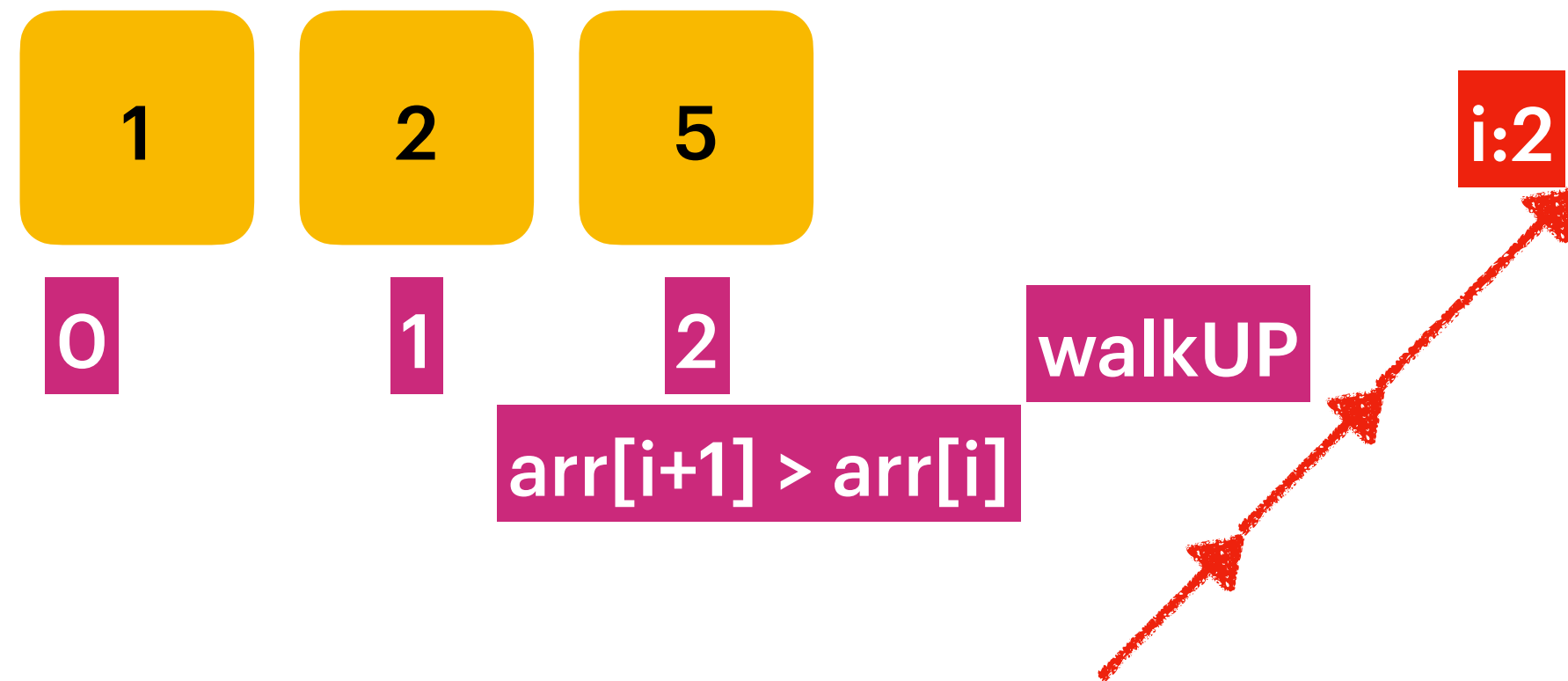
Example 3:

Input: `arr = [0,3,2,1]`
Output: `true`

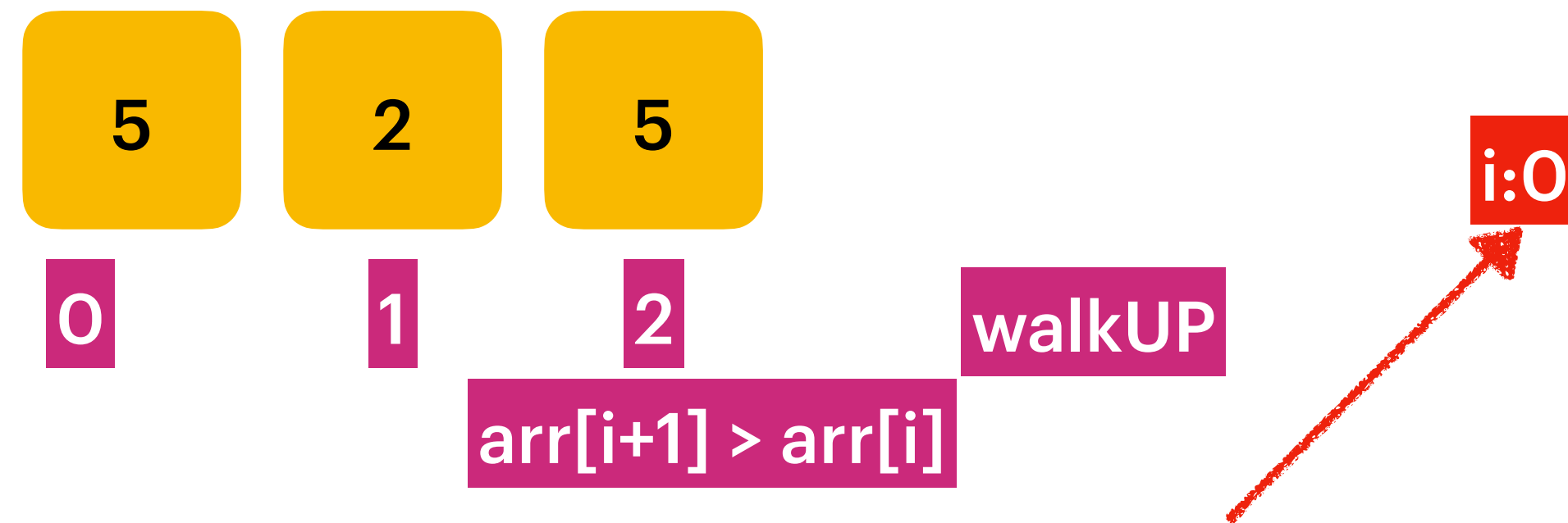
Constraints:

- `1 <= arr.length <= 104`
- `0 <= arr[i] <= 104`





Base Checks :
After Walkup if your index either '0' Or 'n-1'
return false



Algorithm

Time Complexity : $O(n)$
Space Complexity : $O(1)$

1. WalkUP till $\text{arr}[i + 1] > \text{arr}[i]$
2. Have Base check
either index '0' or 'n-1' return false.
If the elements are ascending order then
walkup results to index value as 'n-1'
If the second element is less than the first
element then index value '0'
3. WalkDown till $\text{arr}[i] > \text{arr}[i + 1]$
4. Return true if the index reached to last
element otherwise return false:
return $i == n - 1$