# 4. Median of Two Sorted Arrays

Hard 15850 1951 Add to List Share

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.

**Example 1:**

```
Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.
```

**Example 2:**

```
Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3)
/ 2 = 2.5.
```

**Constraints:**

- `nums1.length == m`
- `nums2.length == n`
- `0 <= m <= 1000`
- `0 <= n <= 1000`
- `1 <= m + n <= 2000`
- $-10^6 <= nums1[i], nums2[i] <= 10^6$

# 4 Median Of Two Sorted Arrays → Let's Apply Math

nums1

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 3 | 5 | 8 | 9 |

n1:5

nums2

|   |   |   |   |   |
|---|---|---|---|---|
| 2 | 4 | 6 | 7 | 10 |

n2:5

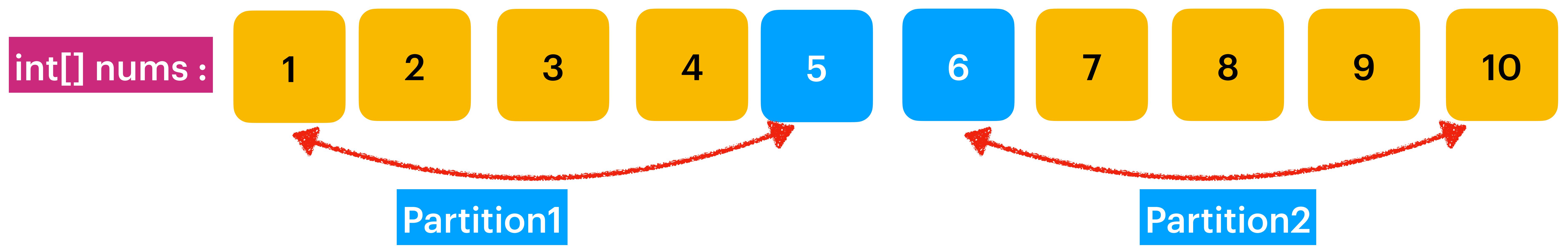
Lets understand the key points

↓

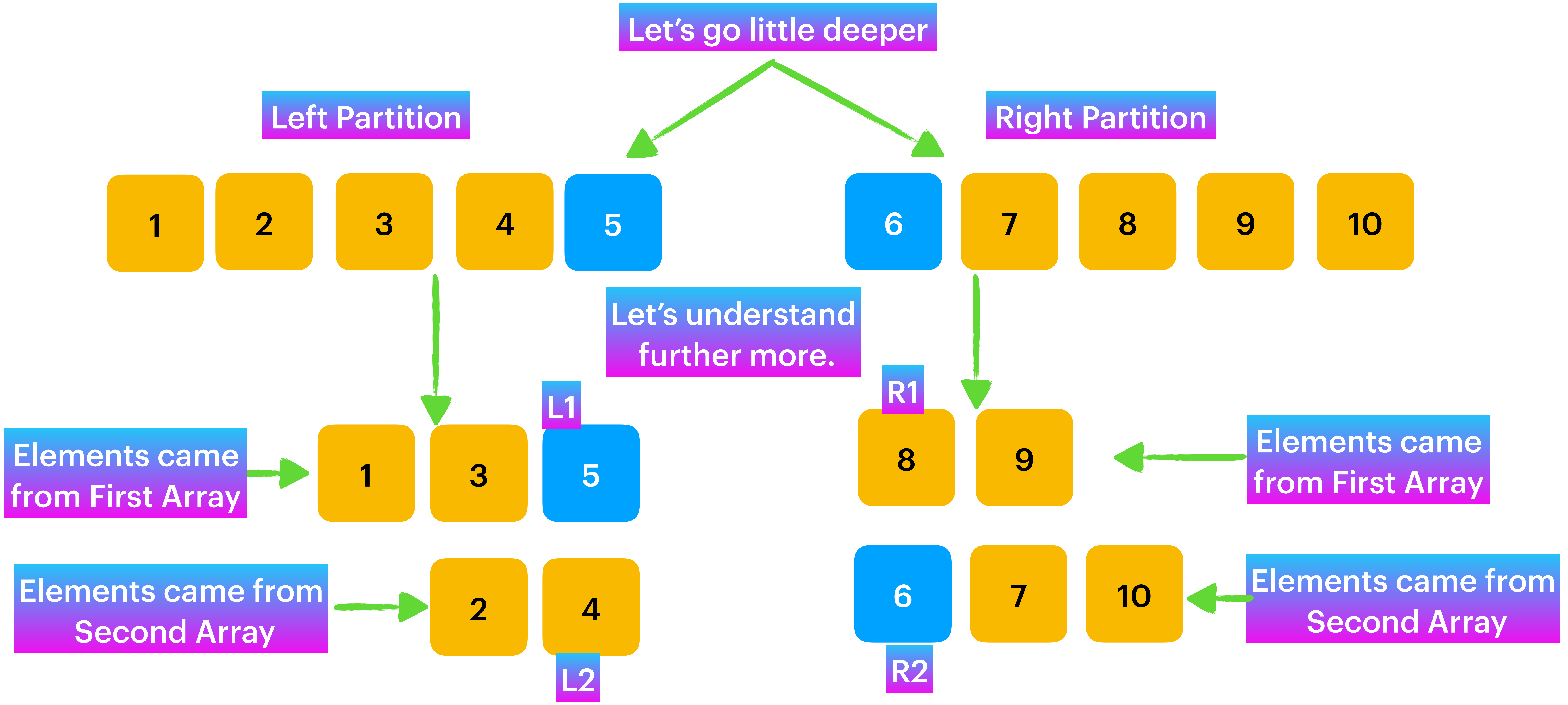
Both the arrays are sorted.

↓

Let's see the magic what happened
earlier when we combine and find out the Median.



If we analyse the above array, we found two equal partitions and the median is average of
Left Most element [5] From Partition1 and Right First Element [6]
from Partition2. —> (5+6) / 2


Let's go little deeper
Left Partition
Right Partition
1
2
3
4
5
6
7
8
9
10
Let's understand further more.
L1
R1
Elements came from First Array
1
3
5
8
9
Elements came from First Array
Elements came from Second Array
2
4
L2
6
7
10
R2
Elements came from Second Array

If we are able to divide both the input arrays into equal partitions on the Fly Then —->

L1 → Left Most Element From arr1 on the Left Partition.

L2 → Left Most Element From arr2 on the Left Partition.

R1 → Right First Element From arr1 on the Right Partition.

R2 → Right First Element From arr2 on the Right Partition.

When can we decide to find out the median.

↓

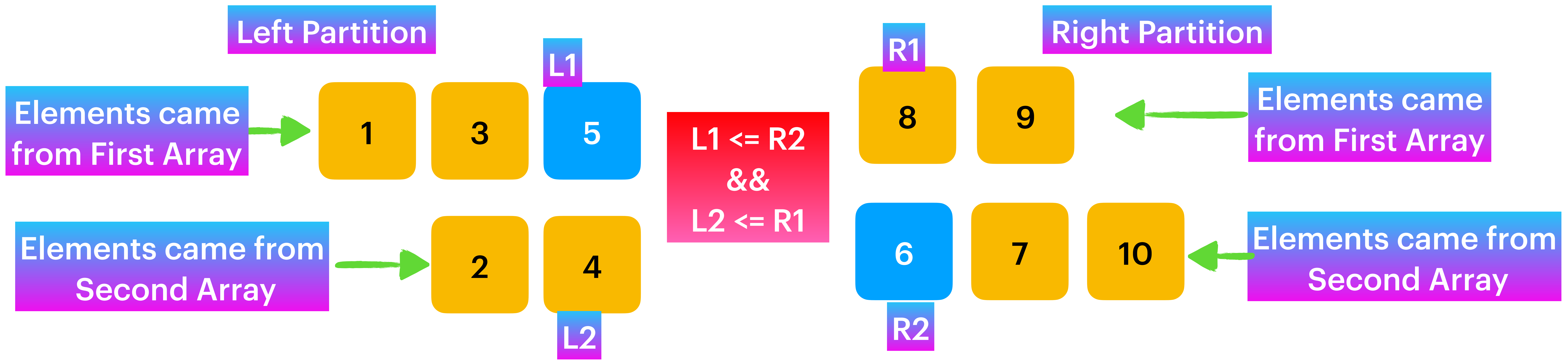
As we know that arr1 and arr2 are sorted so always L1 <= R1 && L2 <= R2

↓

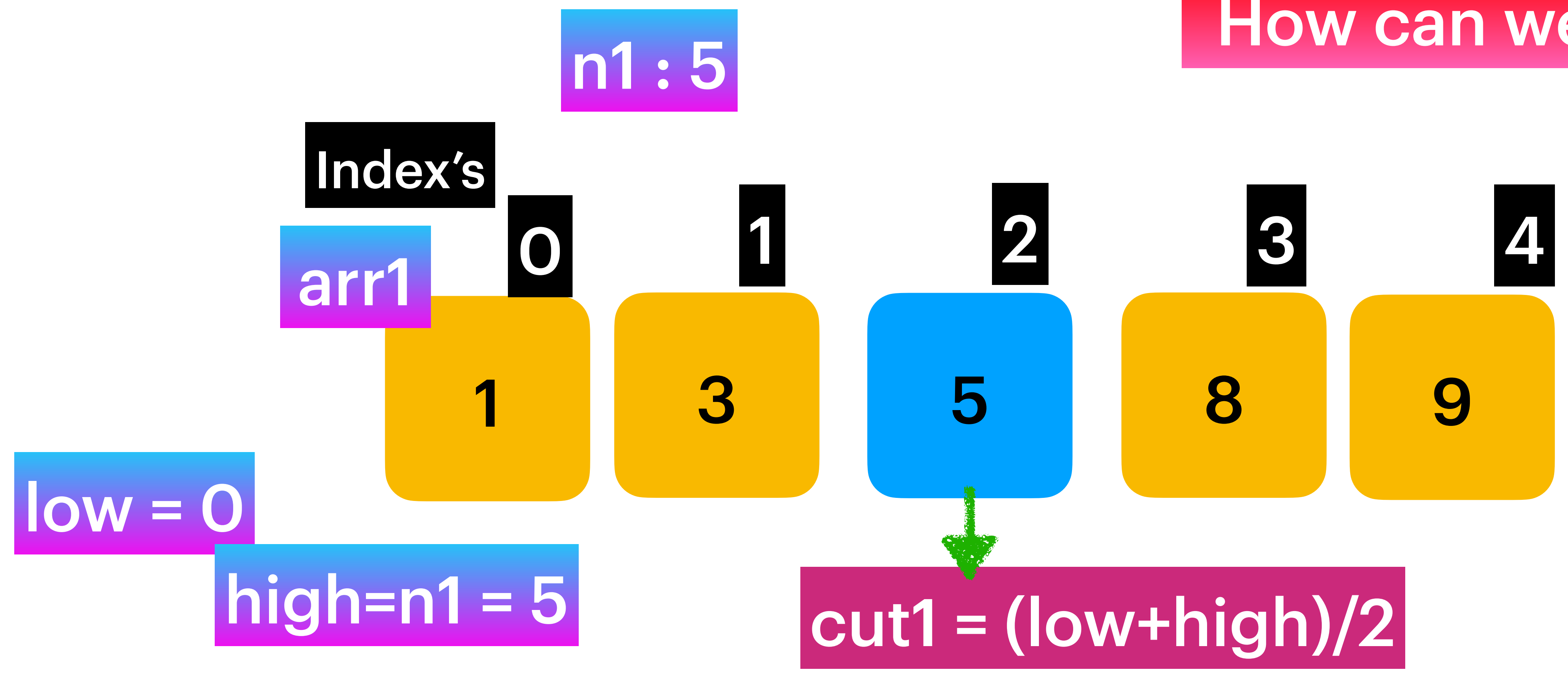
So that time to find the median is When L1 <= R2 && L2 <= R1

↓

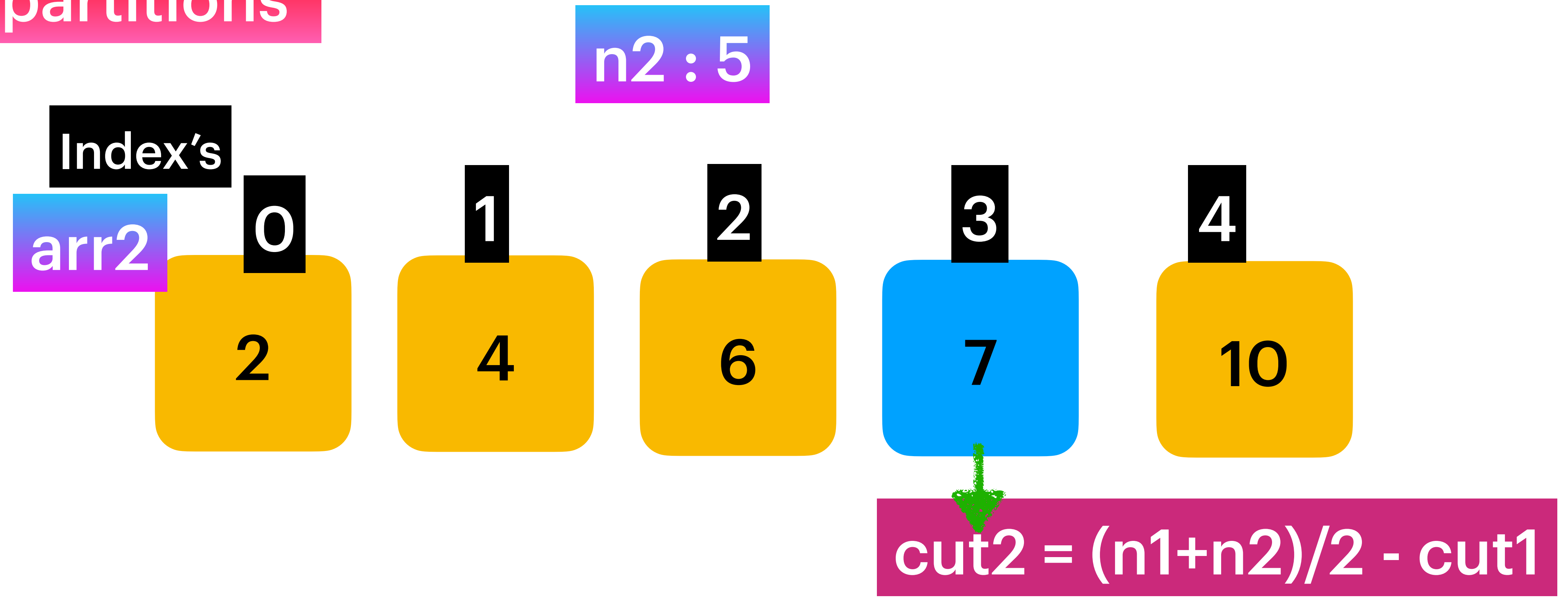
We also know that From Left partition we would need to consider the Max value and from the Right partition we would need consider Min Value.
So that in an even array Median principle is
(Math.max(L1,L2) + Math.min(R1,R2)) / 2


Left Partition
Right Partition
Elements came from First Array
Elements came from Second Array
Elements came from First Array
Elements came from Second Array
L1
L2
R1
R2
1
3
5
2
4
8
9
6
7
10
L1 <= R2 && L2 <= R1

## How can we devide into equal partitions



Lets Apply Math

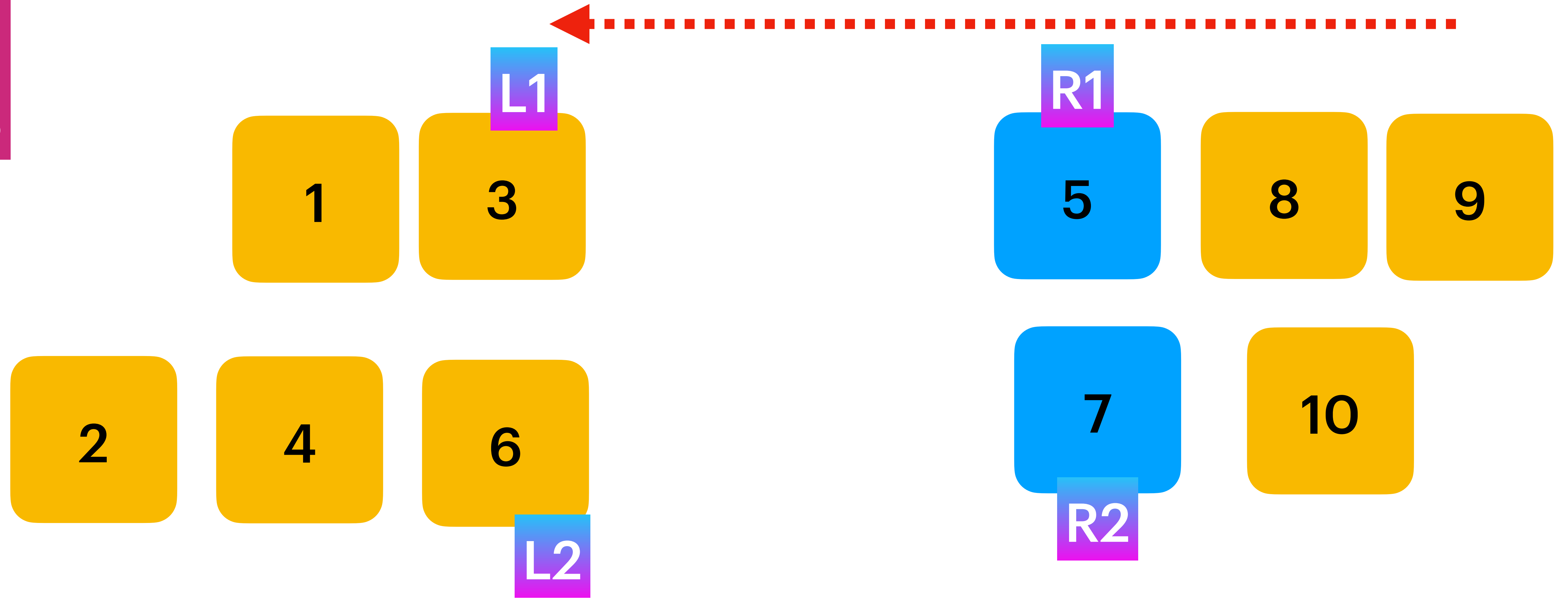


int n = n1+n2 = 10;
lets initialise
low = 0
high = n1 = 5

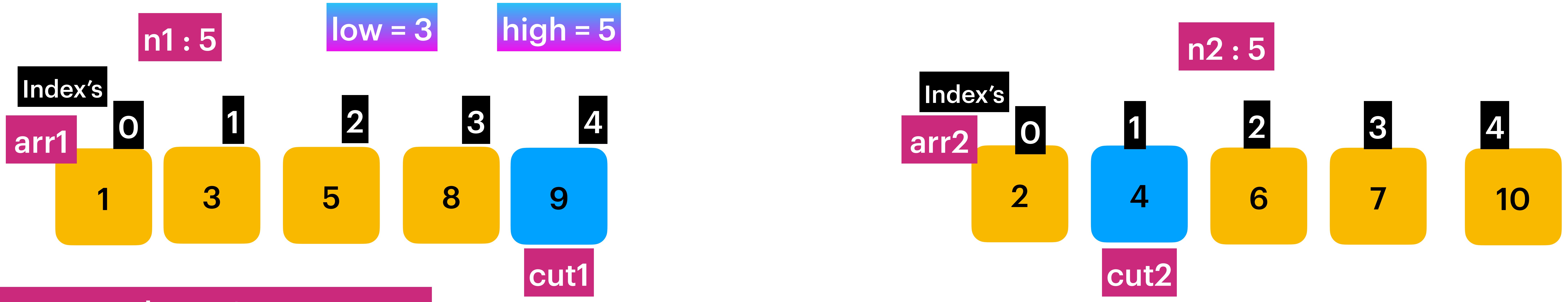
Lets do cut1 = low+high/2 = 2
cut2 = (n1+n2)/2 - cut1 = 5 - 2 = 3

R1 = arr1[cut1] = 5
L1 = arr1[cut1-1] = 3
R2 = arr2[cut2] = 7
L2 = arr2[cut2 - 1] = 6

See now we are able to divide above two arrays into equal partitions.
Based on cut1, cut2



As L2 > R1 we would need to increase the left portion scope of arr1 :
low = cut1 + 1 = 3
No change in high pointer
i.e high = 5

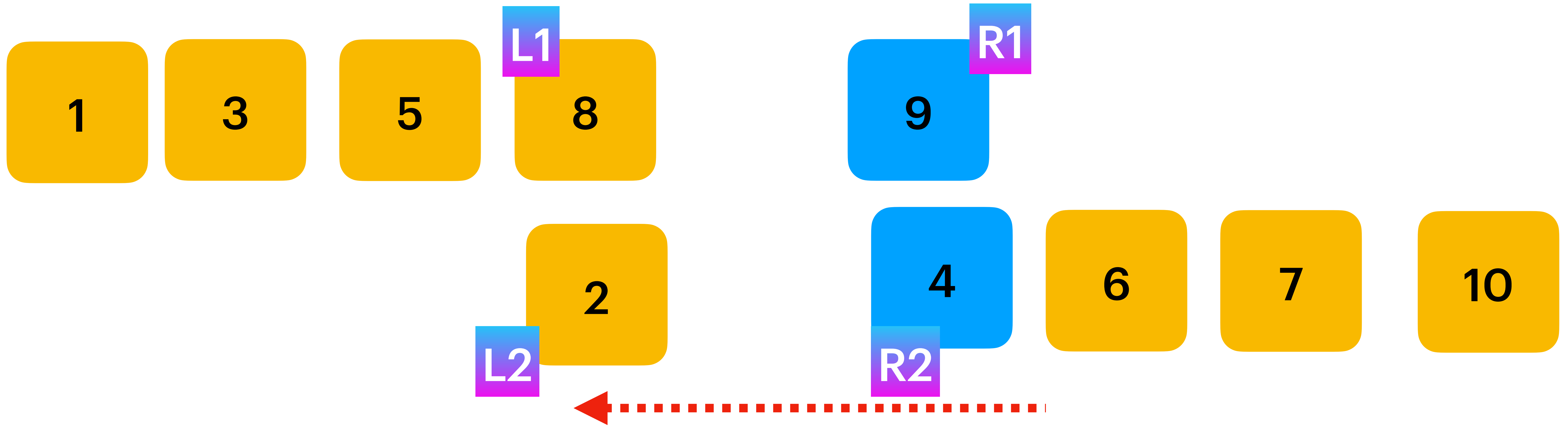


low = 3
high = 5

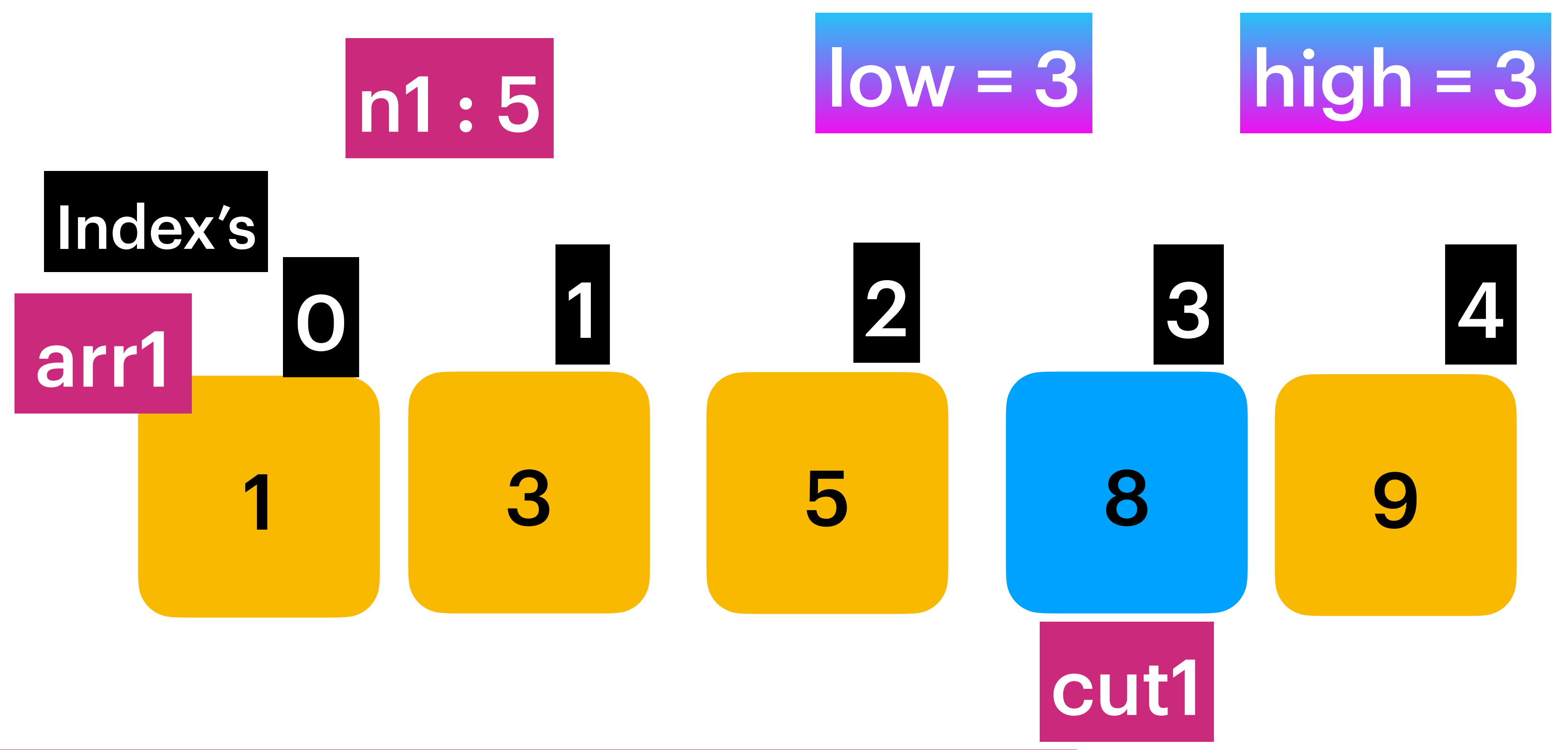
Lets do cut1 = low+high/2 = 4
cut2 = (n1+n2)/2 - cut1 = 5 - 4 = 1

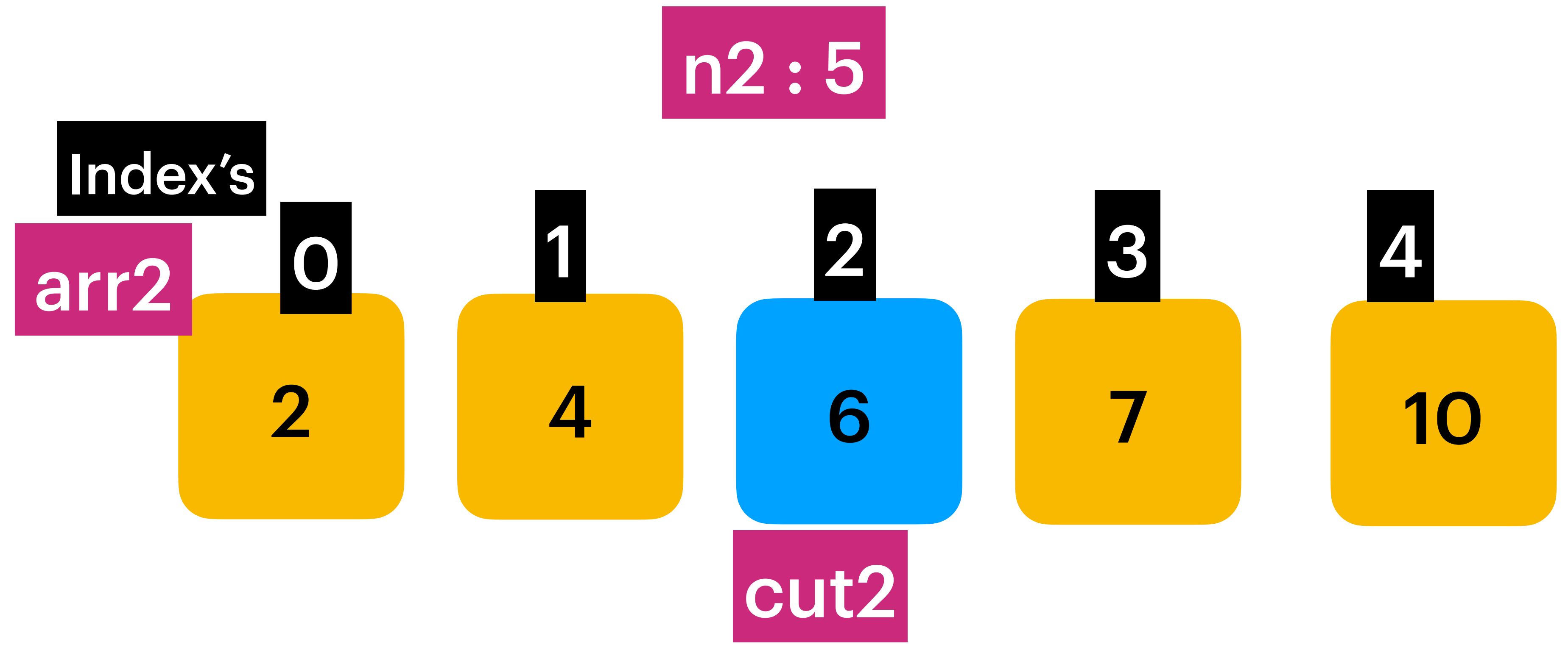
See now we are able to divide above two arrays into equal partitions.
Based on cut1, cut2

R1 = arr1[cut1] = 4
L1 = arr1[cut1-1] = 8
R2 = arr2[cut2] = 4
L2 = arr2[cut2 - 1] = 2



As L1 > R2 we would need to increase the left portion scope of arr2 :
high = cut1 - 1 = 3
No change in low pointer
i.e low = 3



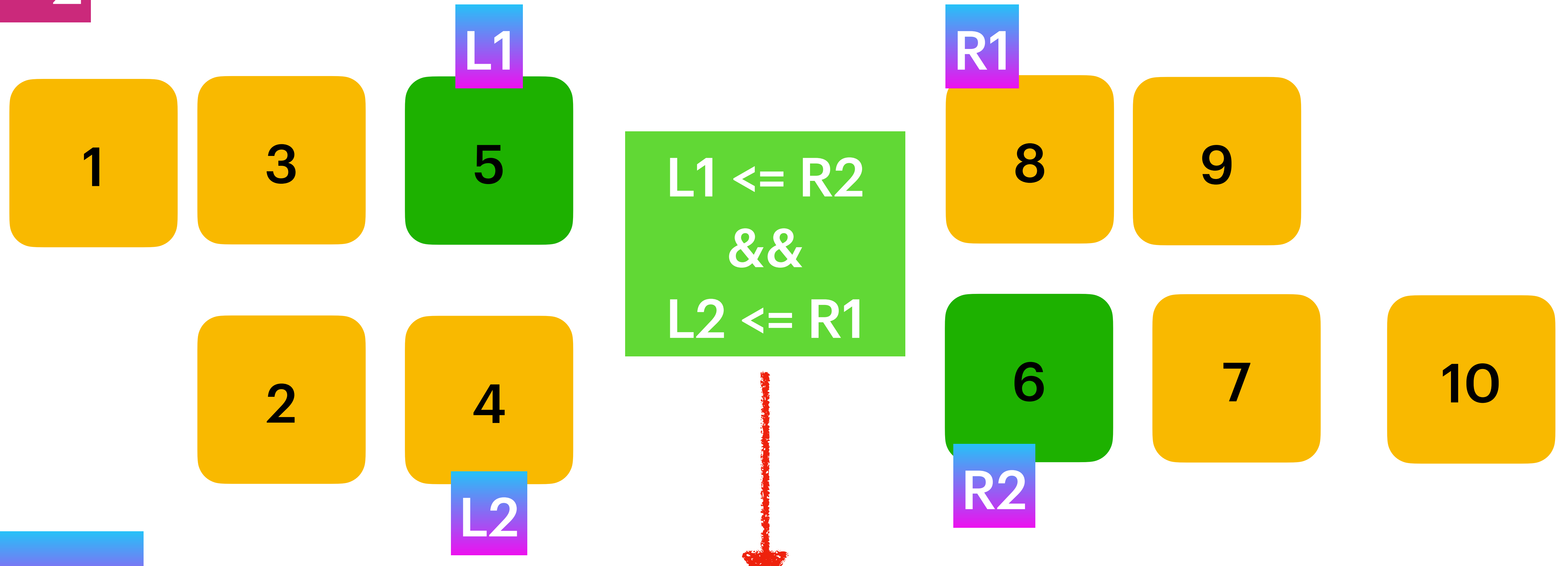


low = 3
high = 3

Lets do cut1 = low+high/2 = 3
cut2 = (n1+n2)/2 - cut1 = 5 - 3 = 2

See now we are able to divide above two arrays into equal partitions.

R1 = arr1[cut1] = 8
L1 = arr1[cut1-1] = 5
R2 = arr2[cut2] = 6
L2 = arr2[cut2 - 1] = 4



L1 <= R2
&&
L2 <= R1

As the n1+n2 is even

Median = Math.max(L1,L2) + Math.min(R1,R2) / 2
= (5+6) / 2 = 5.5

If the size of the array is odd

Median = Math.min(R1,R2)

## Edge Cases

We are deriving cut2 from cut1
so that just to avoid ArrayIndexOutOfBoundsException
make sure arr2.length > arr1.length.

If either of the arrays has only one element then
either cut1 or cut2 would be 0 so that there is  no left
Then L1 or L2 is Integer.MIN_VALUE

If either of cut1 or cut2  equals to length n1 or n2 then there is no right then
R1 or R2 is Integer.MAX_VALUE

Time Complexity :  O(log( Math.min(n1,n2) )

Space Complexity :  O(1 )

If there are n elements then we solve this problem in log(n) steps.
In our use case we had 10 elements got the solution in 3 steps.
We can say O (log(n1+n2)).
To be precising We can get the solution log( Math.min(n1,n2) )

Please Exercise Below sorting techniques so that we can move on to QuickSort

Bubble Sort

Selection Sort

Insertion Sort