# Problem Definition

Given an integer array, circularly shift the array left side k times.

**Example:**

*Inputs:*

array = {1, 2, 3, 4, 5} and k = 2

*Output:*

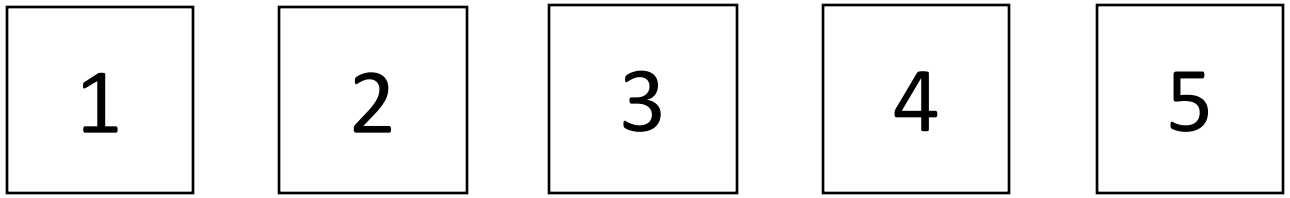
{3, 4, 5, 1, 2}

# Solution 1 (naïve)

```
shiftLeft (array, k):
    if k < 0:
        return -1
    n ← length of the array
    for i from 0 to k:
        temp ← array[0]
        for j from 0 to n-1:
            array[j] ← array[j+1]
        end for
        array[n] ← temp
    end for
    return array
```

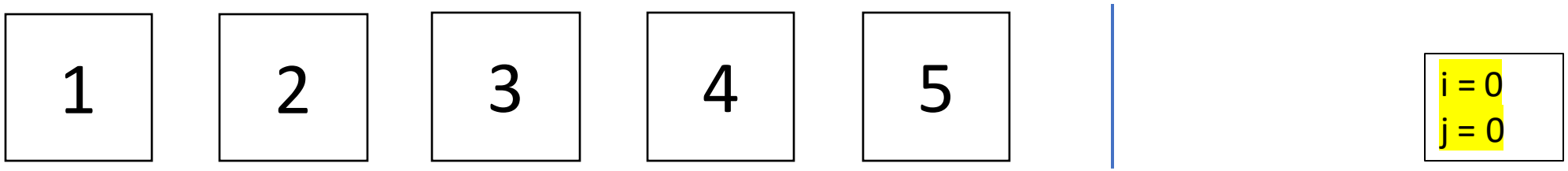
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



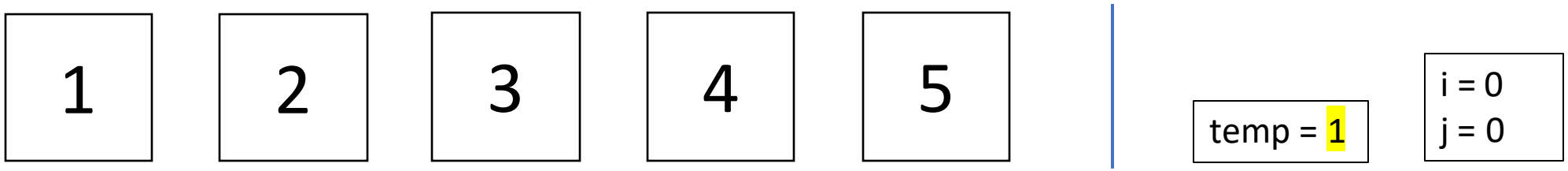
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



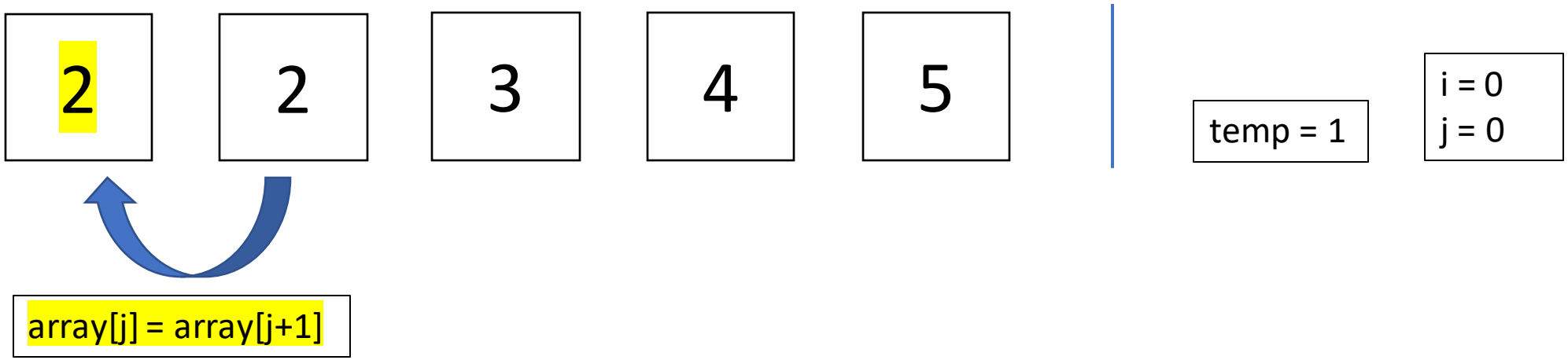
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



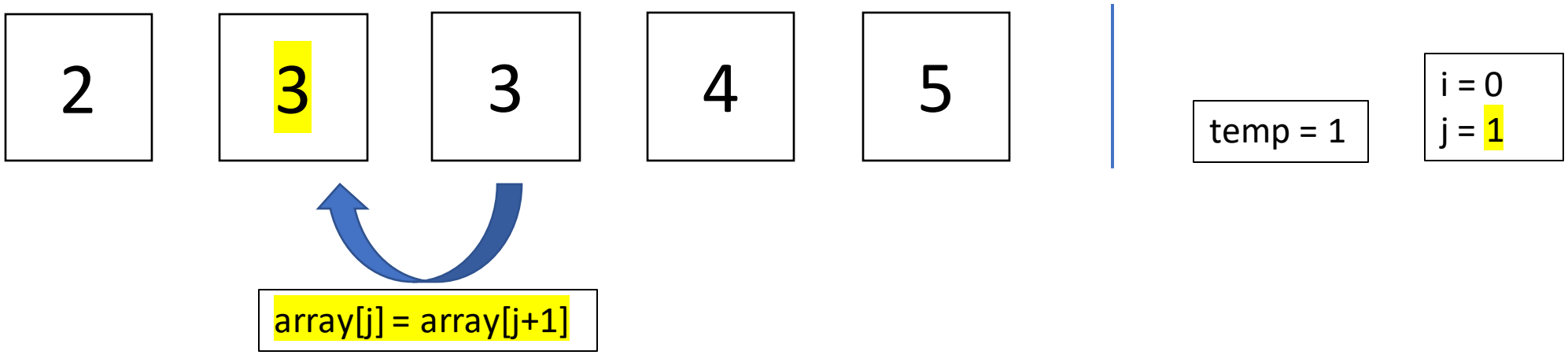
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



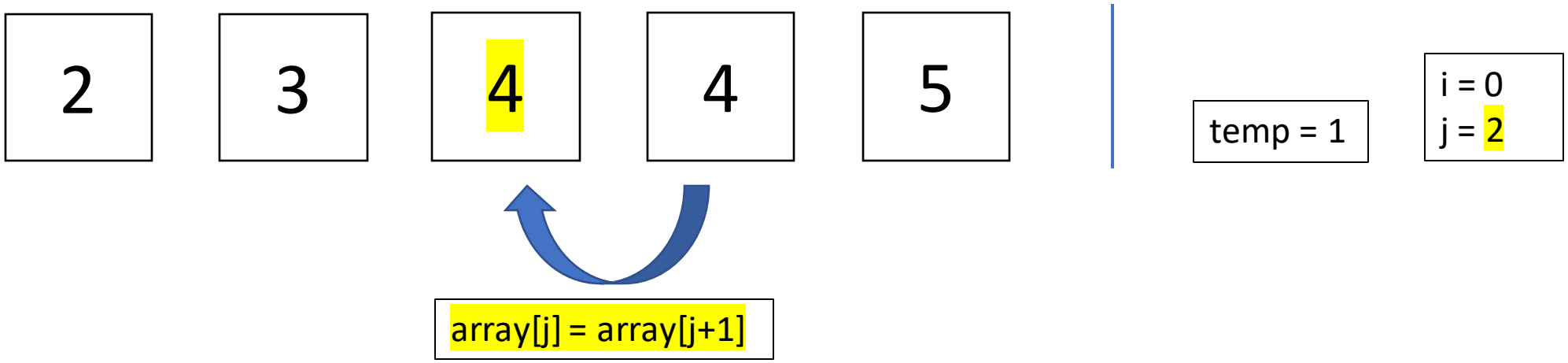
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



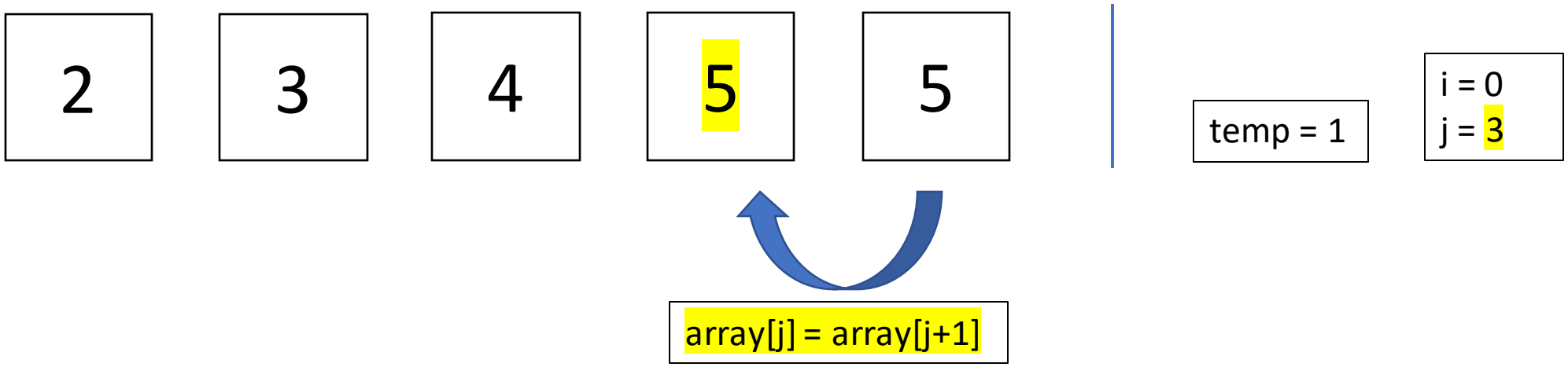
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



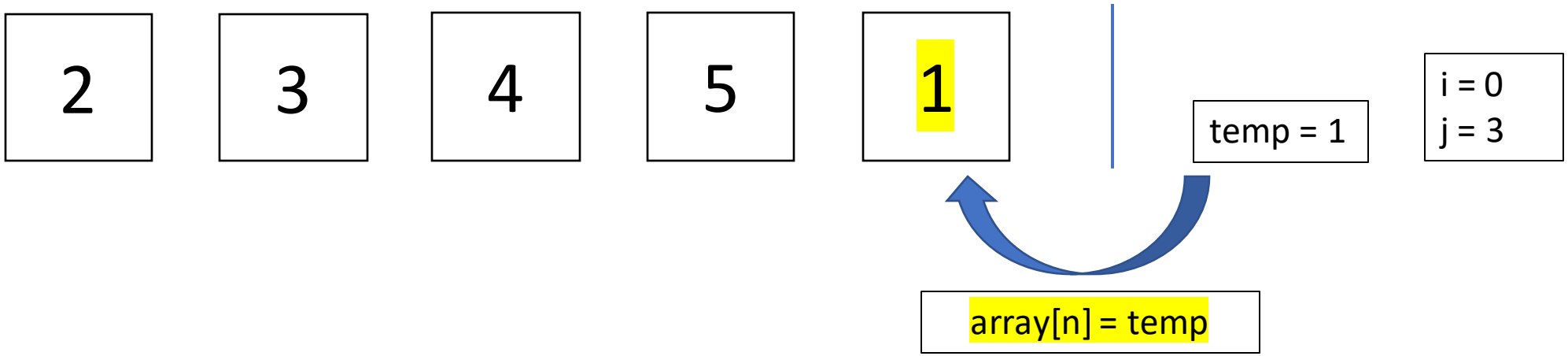
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



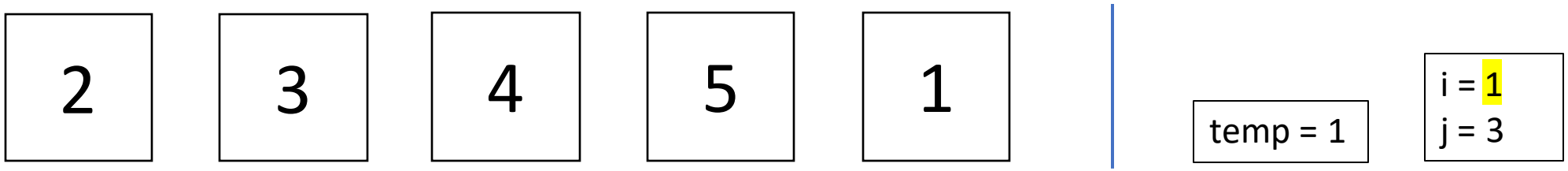
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



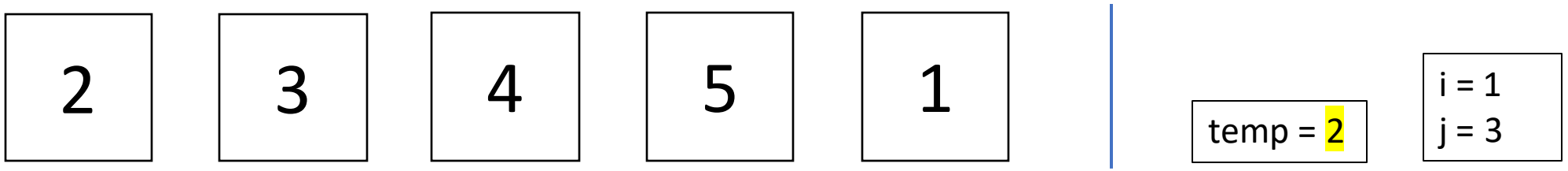
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



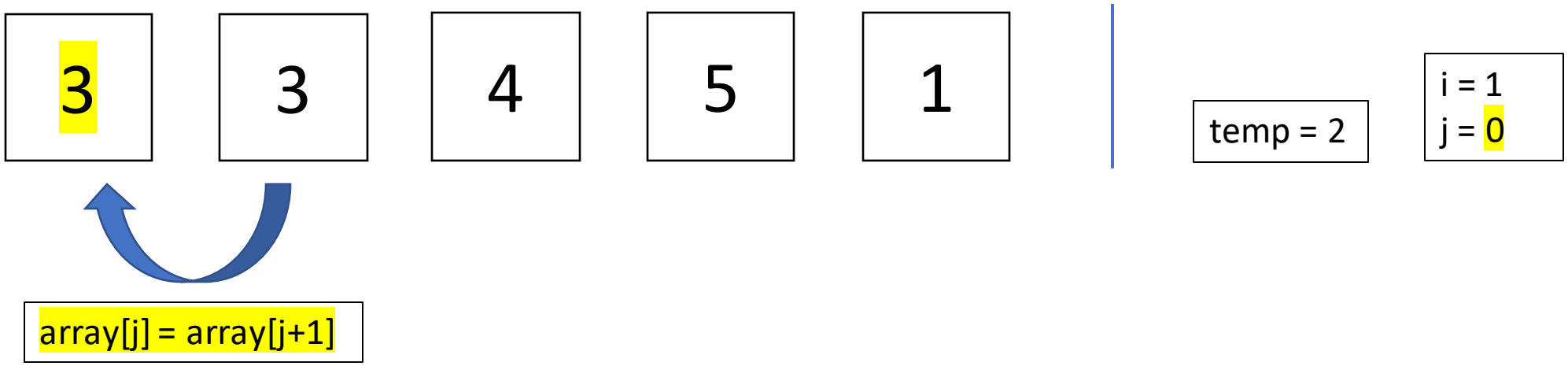
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



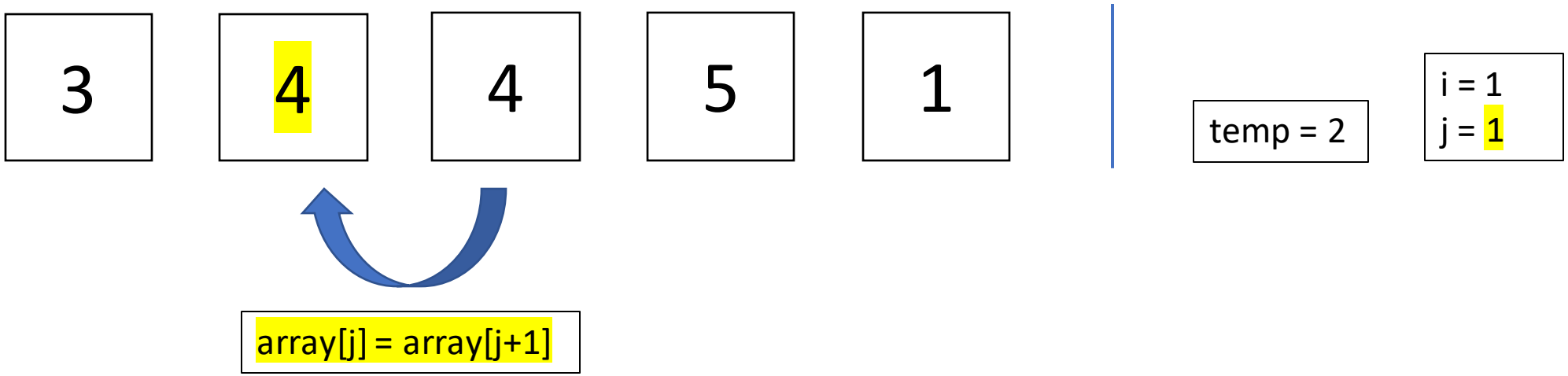
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



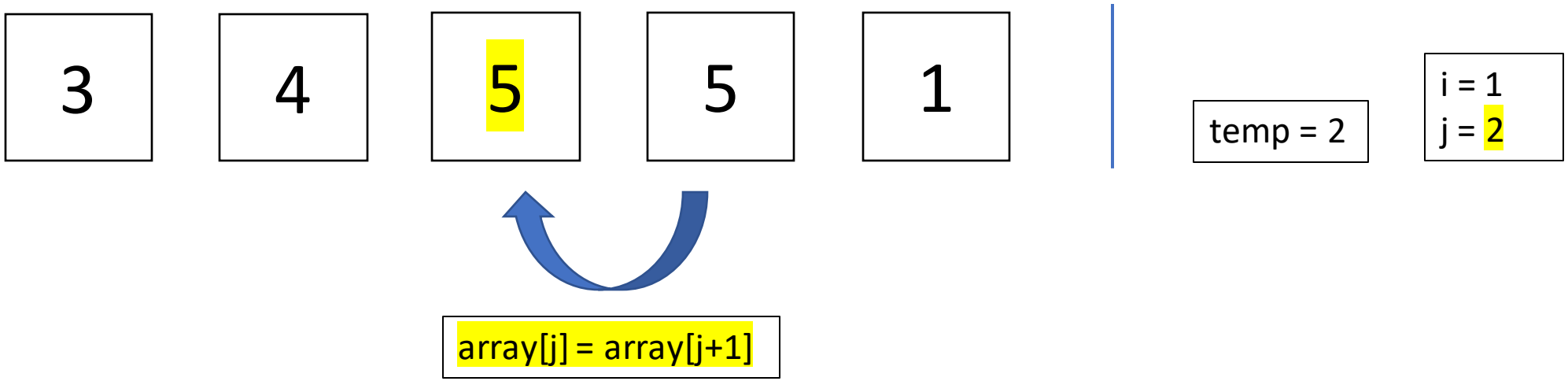
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



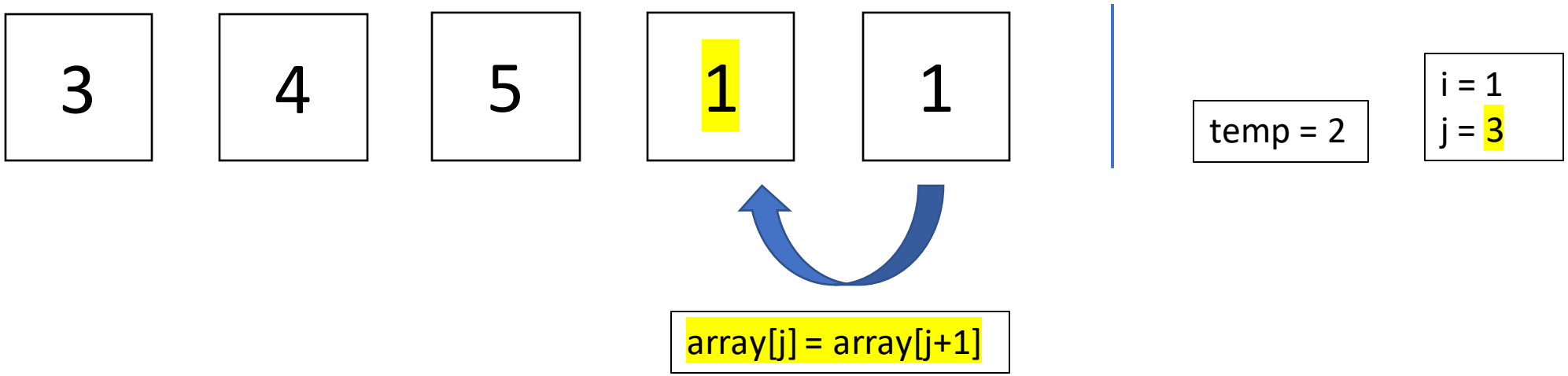
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



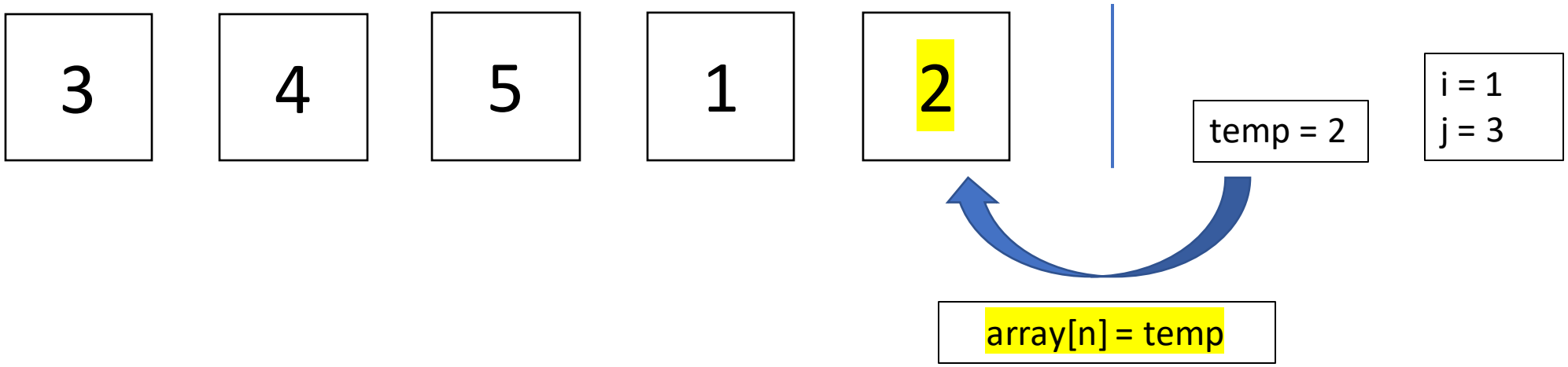
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



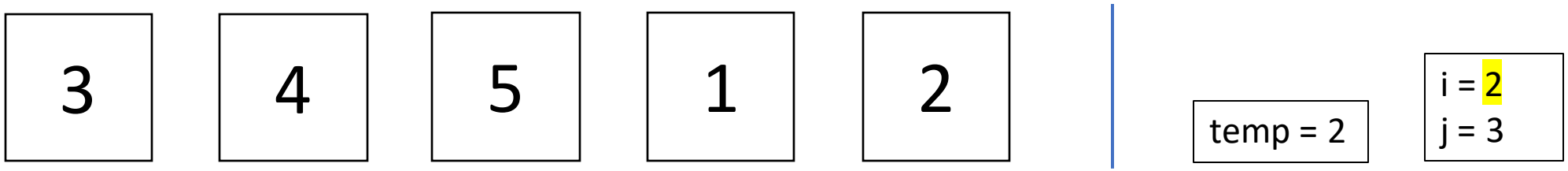
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



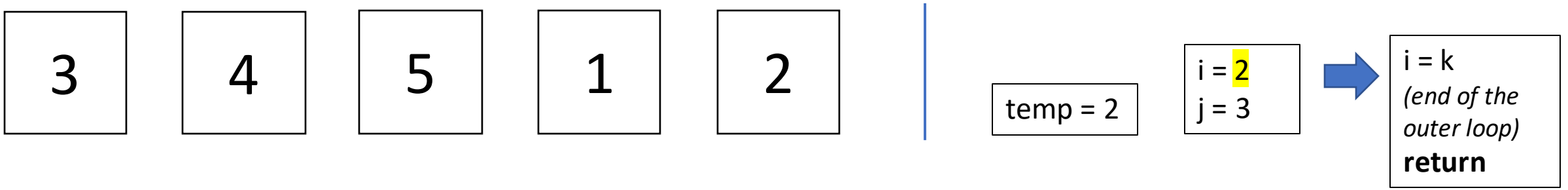
# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



# Solution 1 (naïve)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



# Solution 1 (naïve)

## Time Complexity?

The best and the worst cases of the algorithm depend on **k**. Shifting left once requires n swapping operations. We have to repeat this k times, therefore the time complexity is O(n*k).

Specifically:

- If k = 0, it means no shifting and therefore the best case time complexity becomes Ω(1).
- The array might be shifted to left at most n-1 times. Because *shifting it to left side 3 times or n+3 times forms the same output.* So, if we edit the outer loop's upper limit at k%n, the loop will be iterated at most n-1 times. Which makes the worst case time complexity O(n*(n-1)) = O(n^2). But if we don't edit the outer loop, complexity might grow drastically according to k.

## Space Complexity?

The algorithm uses two iterators and a temporary variable. Space complexity is constant.

# Solution 2 (auxiliary array)

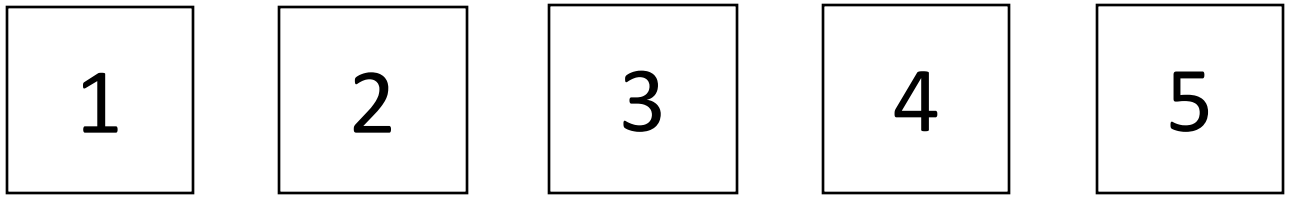
```
shiftLeft (array, k):
    if k ≤ 0:
        return -1
    end if
    k ← k % n
    n ← length of the array
    aux_array ← [ ]
    for i from 0 to k:
        append array[i] to aux_array
    end for
    for i from k to n:
        array[i-k] ← array[i]
    end for
    for i from n-k to n:
        array[i] ← aux_array [i - (n - k)]
    end for
    return array
```

# Solution 2 (auxiliary array)

```
shiftLeft (array, k):
    if k ≤ 0:
        return -1
    end if
    k ← k % n
    n ← length of the array
    aux_array ← [ ]
    for i from 0 to k:                    ← 1st loop
        append array[i] to aux_array
    end for
    for i from k to n:                    ← 2nd loop
        array[i-k] ← array[i]
    end for
    for i from n-k to n:                  ← 3rd loop
        array[i] ← aux_array [i - (n - k)]
    end for
    return array
```

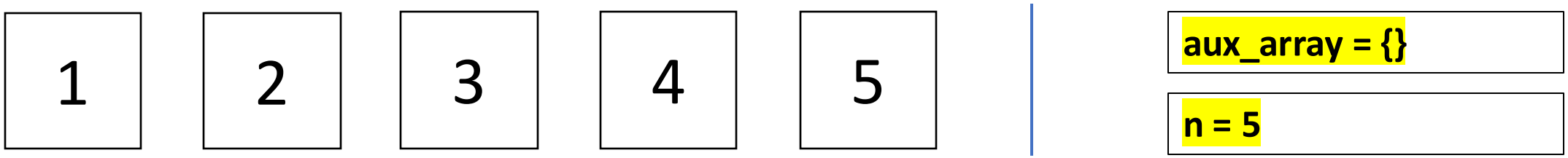
# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



# Solution 2 (auxiliary array)

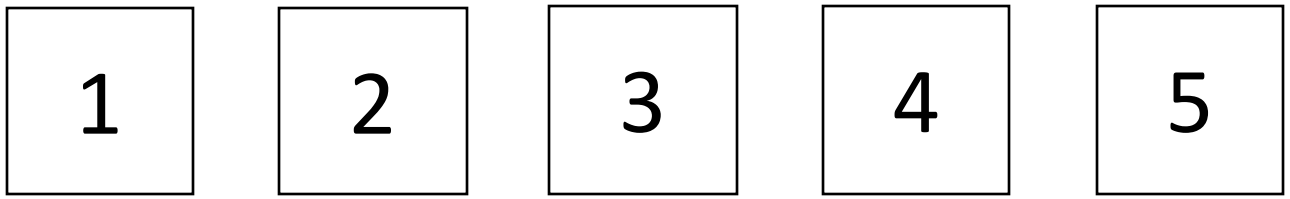
*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st loop:**




**aux_array = {}**

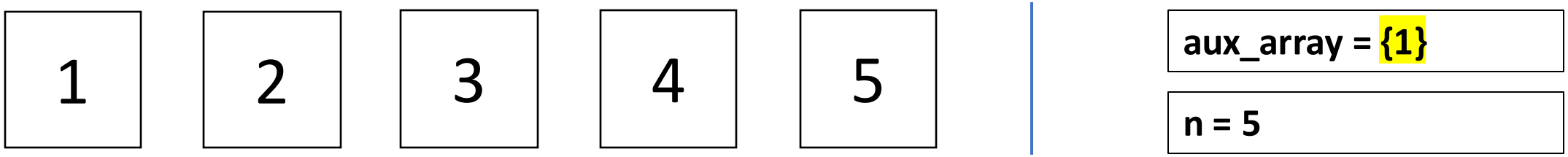
**n = 5**

**1st loop:**
for i from 0 to k
i= 0

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st loop:**

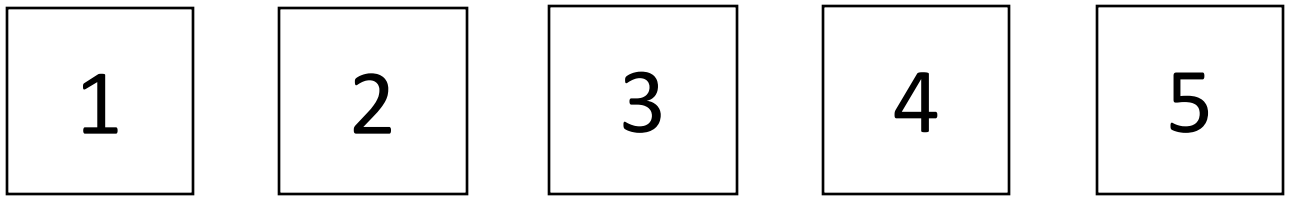



**1st loop:**
for i from 0 to k
i= 0 → append array[0] to aux_array

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st loop:**




**aux_array = {1}**

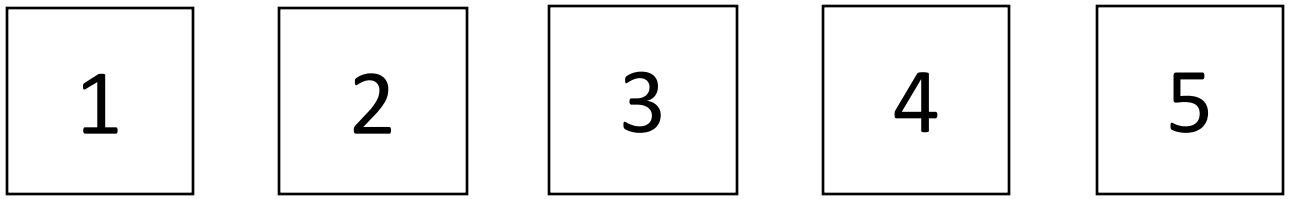
**n = 5**

**1st loop:**
for i from 0 to k
i= 1

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st loop:**




**aux_array = {1, 2}**

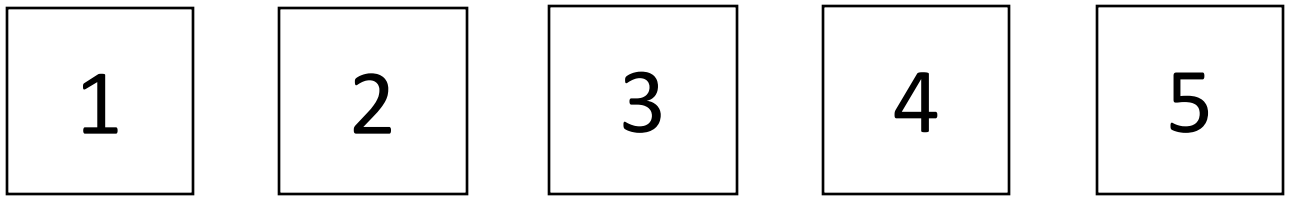
**n = 5**

**1st loop:**
for i from 0 to k
i= 1 → append array[1] to aux_array

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st loop:**




**aux_array = {1, 2}**

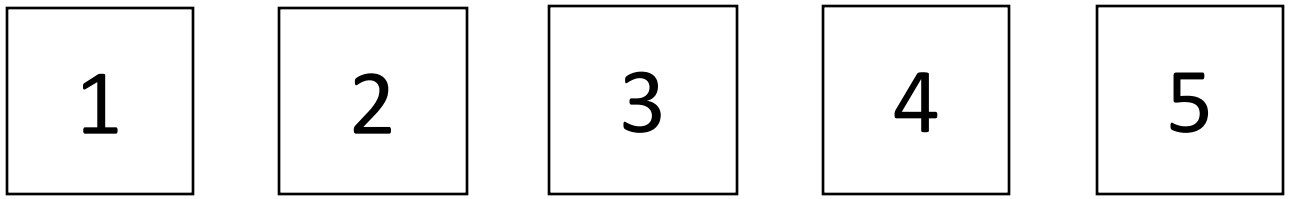
**n = 5**

**1st loop:**
for i from 0 to k
i= 2

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st loop:**




**aux_array = {1, 2}**

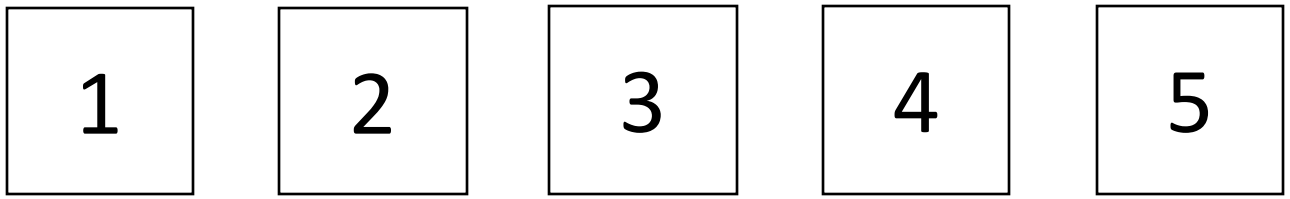
**n = 5**

**1st loop:**
for i from 0 to k
i= 2 → i = k → end of the loop

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd loop:**




**aux_array = {1, 2}**

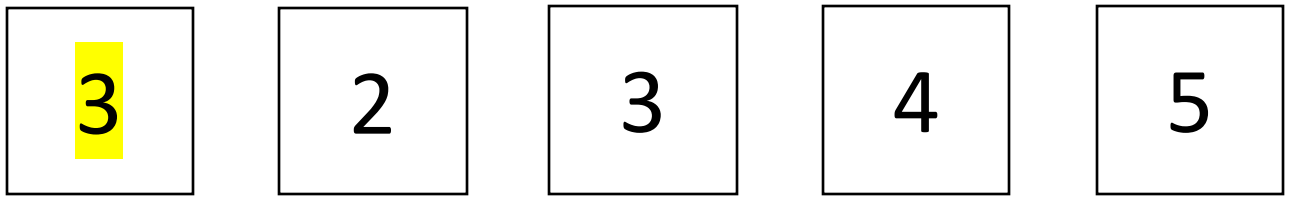
**n = 5**

**2nd loop:**
for i from k to n
i= 2

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd loop:**




**aux_array = {1, 2}**

**n = 5**

**2nd loop:**
for i from k to n
i= 2 → update array[i-k] as array[i]
(array[0] = array[2])

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd loop:**

| 3 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

**aux_array = {1, 2}**

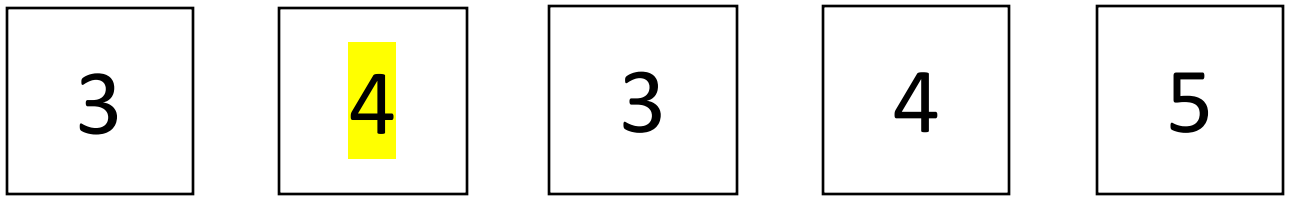
**n = 5**

**2nd loop:**
for i from k to n
i= 3

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd loop:**




**aux_array = {1, 2}**

**n = 5**

**2nd loop:**
for i from k to n
i= 3 → update array[i-k] as array[i]
(array[1] = array[3])

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd loop:**

| 3 | 4 | 3 | 4 | 5 |
|---|---|---|---|---|

**aux_array = {1, 2}**

**n = 5**

**2nd loop:**
for i from k to n
i= 4

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd loop:**


**aux_array = {1, 2}**

**n = 5**

**2nd loop:**
for i from k to n
i= 4 → update array[i-k] as array[i]
(array[2] = array[4])

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd loop:**

| 3 | 4 | 5 | 4 | 5 |
|---|---|---|---|---|

**aux_array = {1, 2}**

**n = 5**

**2nd loop:**
for i from k to n
i= 5

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd loop:**

| 3 | 4 | 5 | 4 | 5 |
|---|---|---|---|---|

**aux_array = {1, 2}**

**n = 5**

**2nd loop:**
for i from k to n
i= 5 → i = n → end of the 2nd loop

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd loop:**

| 3 | 4 | 5 | 4 | 5 |
|---|---|---|---|---|

**aux_array = {1, 2}**

**n = 5**

**3rd loop:**
for i from n-k to n
i= 3

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd loop:**


**aux_array = {1, 2}**

**n = 5**

**3rd loop:**
for i from n-k to n
i= 3 → update array[i] as aux_array[i - (n – k)]
(array[3] = aux_arary[0])

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd loop:**

| 3 | 4 | 5 | 1 | 5 |
|---|---|---|---|---|

**aux_array = {1, 2}**

**n = 5**

**3rd loop:**
for i from n-k to n
i= 4

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd loop:**


**aux_array = {1, 2}**

**n = 5**

**3rd loop:**
for i from n-k to n
i= 4 → update array[i] as aux_array[i - (n – k)]
(array[4] = aux_arary[1])

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd loop:**

| 3 | 4 | 5 | 1 | 2 |
|---|---|---|---|---|

**aux_array = {1, 2}**

**n = 5**

**3rd loop:**
for i from n-k to n
i= 5

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd loop:**

| 3 | 4 | 5 | 1 | 2 |
|---|---|---|---|---|

**aux_array = {1, 2}**

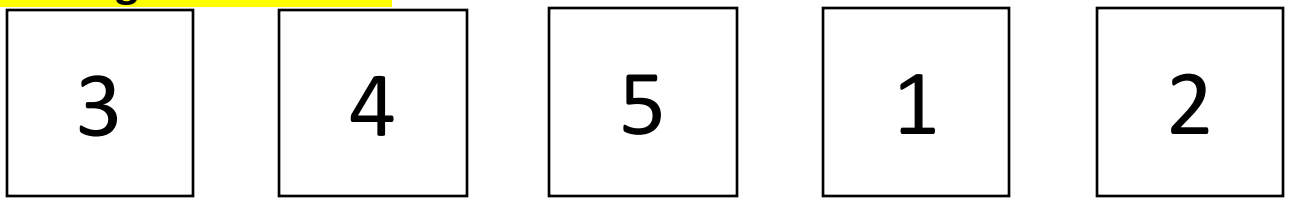
**n = 5**

**3rd loop:**
for i from n-k to n
i= 5 → i = n → end of the 3rd loop

# Solution 2 (auxiliary array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**The algorithm ends.**



# Solution 2 (auxiliary array)

## Time Complexity?

The algorithm consists of 3 consecutive loops, each can have at most n iterations. If we analyze the algorithm we see that the array is divided 2 distinct parts and first 2 loops work on these parts separately.

1st loop works on the first part which is array[0:k] # array[0:k] indicates the elements of the array between indexes 0 and k
2nd loop works on the second part which is array[k:n]

If we minimize the number of iterations of 1st loop, the 2nd array will get closer to n iterations. So, in the best case, we can make them both work with n/2 iterations. In this case the number of iterations of the 3rd loop becomes n/2 too. As a result, we end up with $\Omega(3*(n/2)) = \Omega(n)$.

Obviously, none of the loops takes more than O(n) time. Therefore, the worst case is the same as the best case: O(n).

## Space Complexity?

The algorithm uses an additional array and an iterator. The array's size is k and it dominates the size of the iterator. Therefore, the space complexity is O(k).

# Solution 3 (reversing the array)

```
shiftLeft (array, k):
    if k ≤ 0:
        return -1
    end if
    k ← k % n
    n ← length of the array
    array ← reverse(array, 0, k-1)
    array ← reverse(array, k, n-1)
    array ← reverse(array, 0, n-1)
    return array
```

```
reverse(array, left, right)
    while left < right:
        swap (array[left] and array[right])
        left  ← left  + 1
        right ← right – 1
    end while
    return array
```

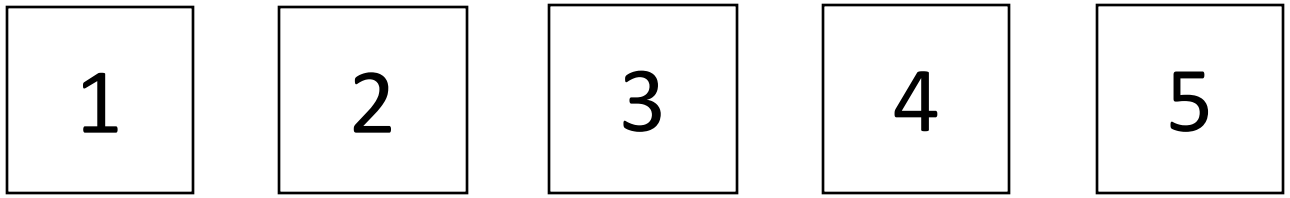
# Solution 3 (reversing the array)

```
shiftLeft (array, k):
    if k ≤ 0:
        return -1
    end if
    k ← k % n
    n ← length of the array
    array ← reverse(array, 0, k-1)   } 1st reverse operation
    array ← reverse(array, k, n-1)  } 2nd reverse operation
    array ← reverse(array, 0, n-1)  } 3rd reverse operation
    return array
```

```
reverse(array, left, right)
    while left < right:
        swap (array[left] and array[right])
        left  ← left  + 1
        right ← right – 1
    end while
    return array
```

# Solution 3 (reversing the array)

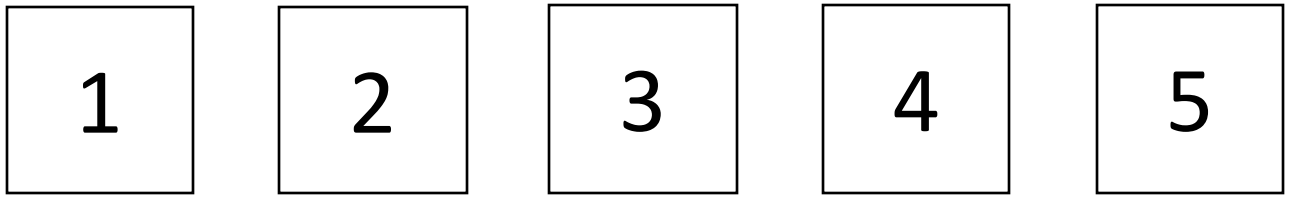
*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



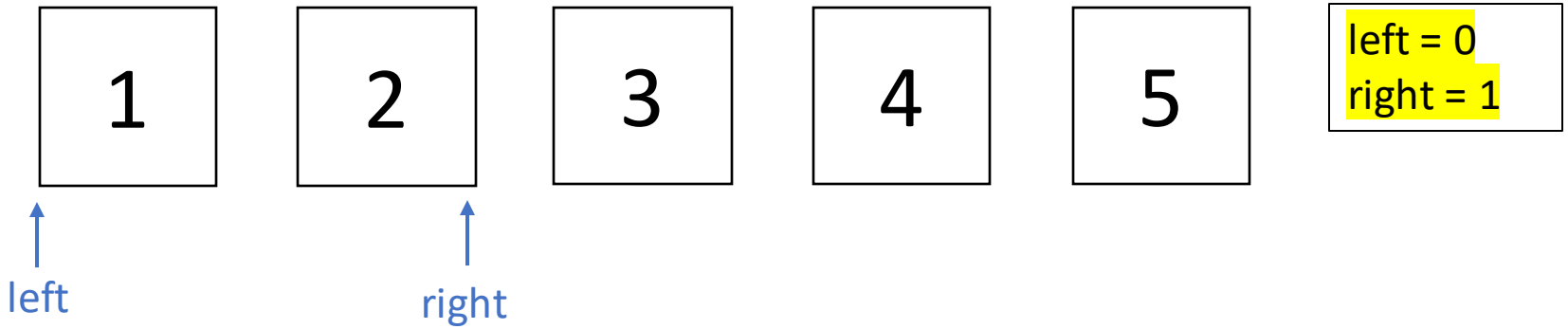
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st reverse operation:**



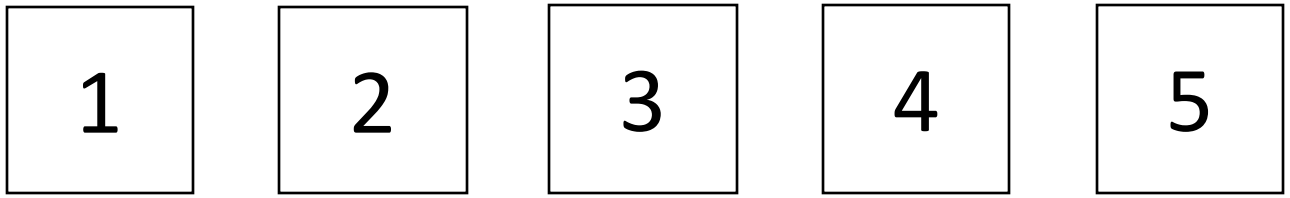

**function call:**
↓ reverse(array, 0 , k-1) ↓



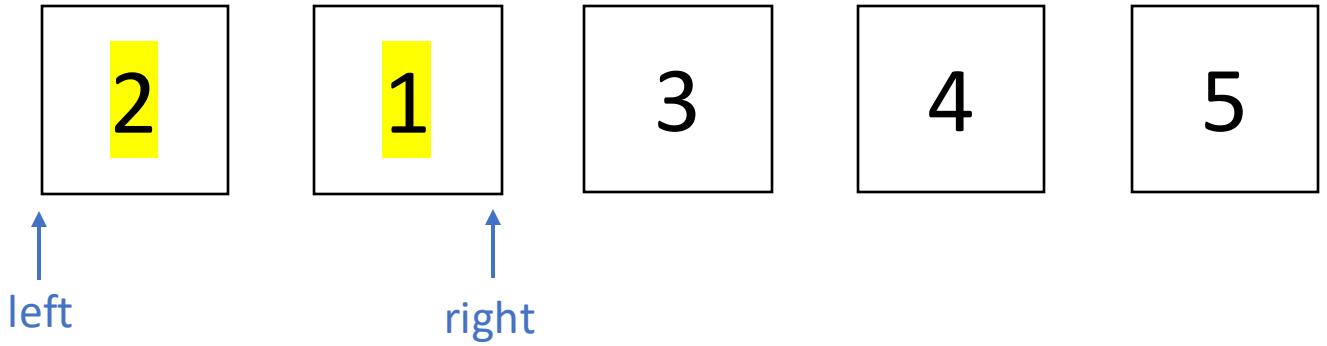
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st reverse operation:**




**function call:**
↓ reverse(array, 0 , k-1) ↓



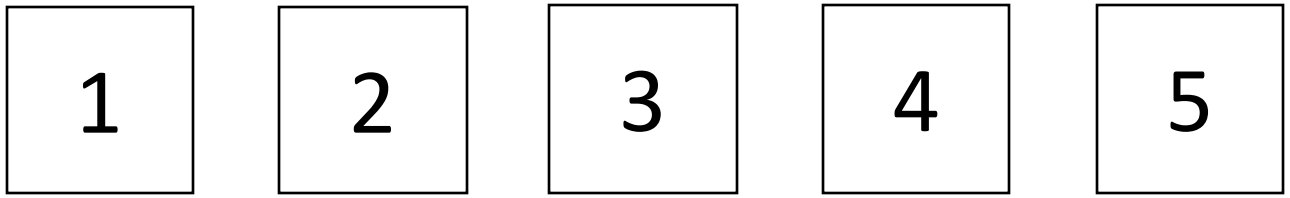

left = 0
right = 1

swap array[left] and array[right]

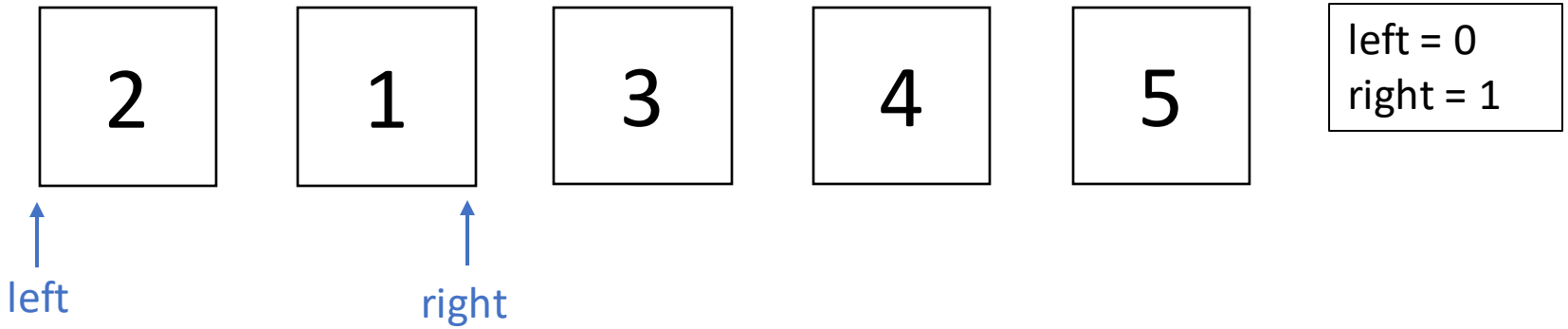
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st reverse operation:**



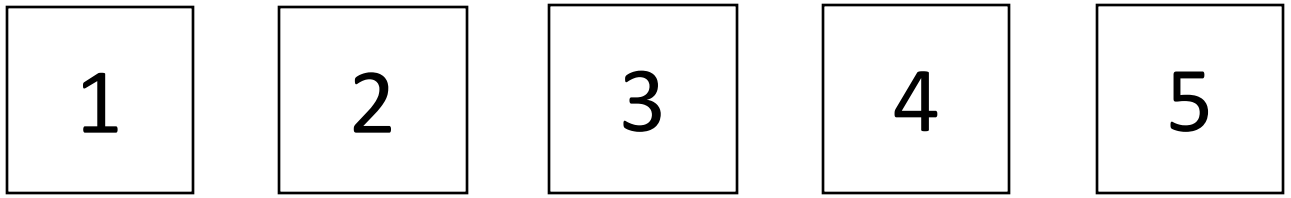

**function call:**
↓ reverse(array, 0 , k-1) ↓



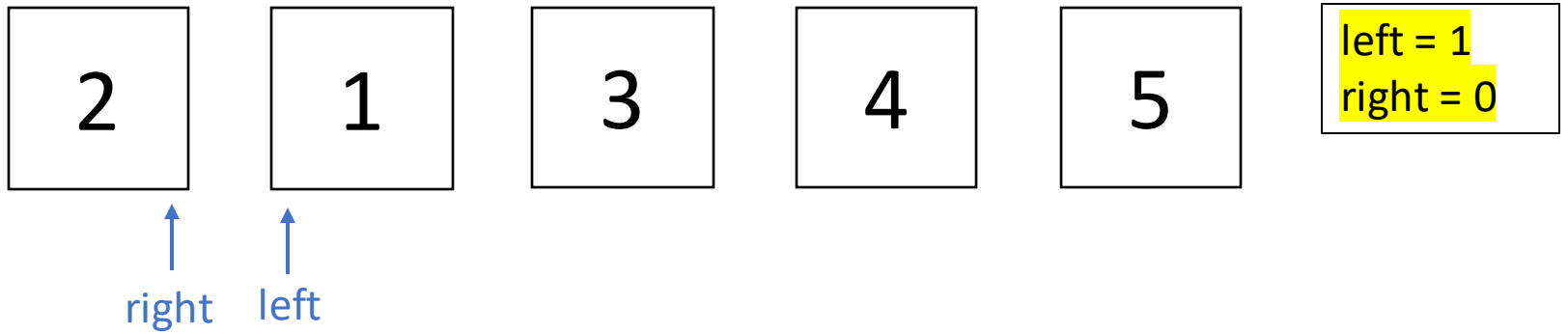
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 1st reverse operation:**




**function call:**
↓ reverse(array, 0 , k-1) ↓



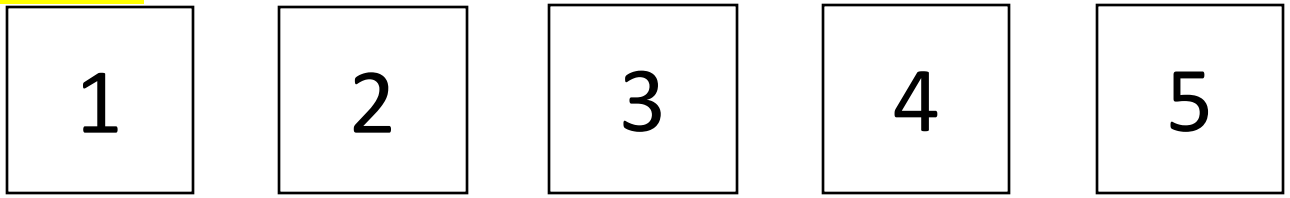

left = 1
right = 0

→ left <? right
**False** → end of the while loop
return array

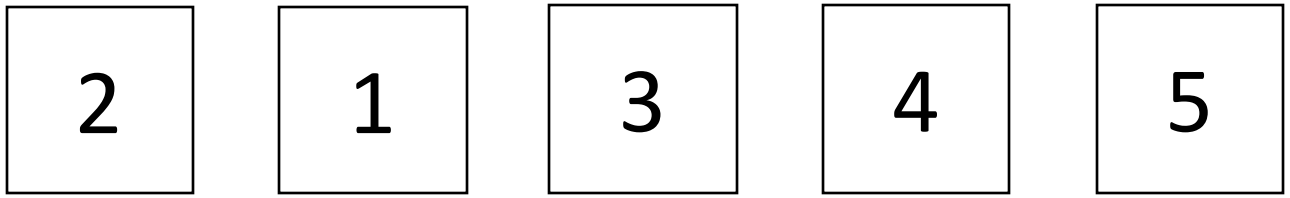
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Array is updated after the 1st reverse operation:**
**Before:**

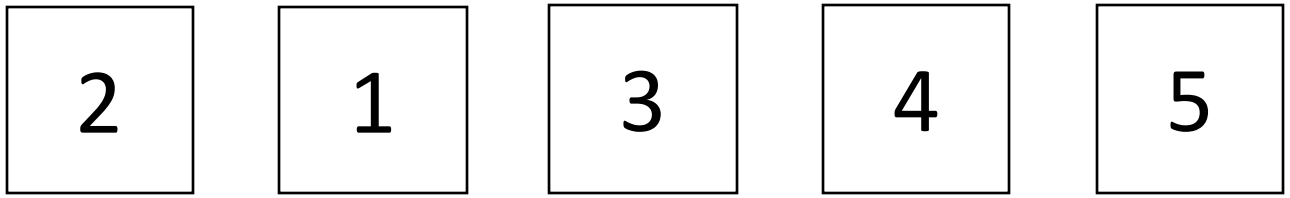



**After:**



# Solution 3 (reversing the array)

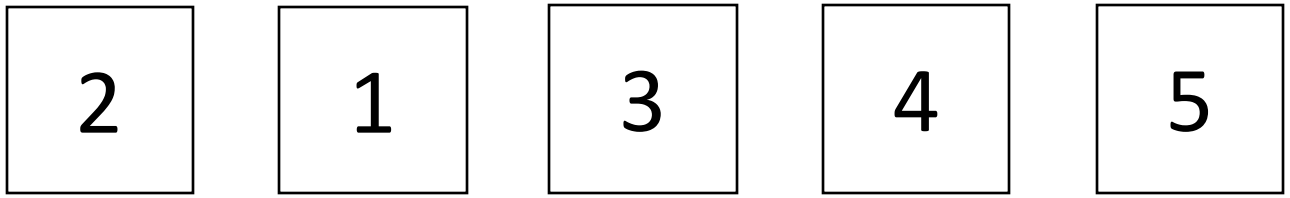
*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2



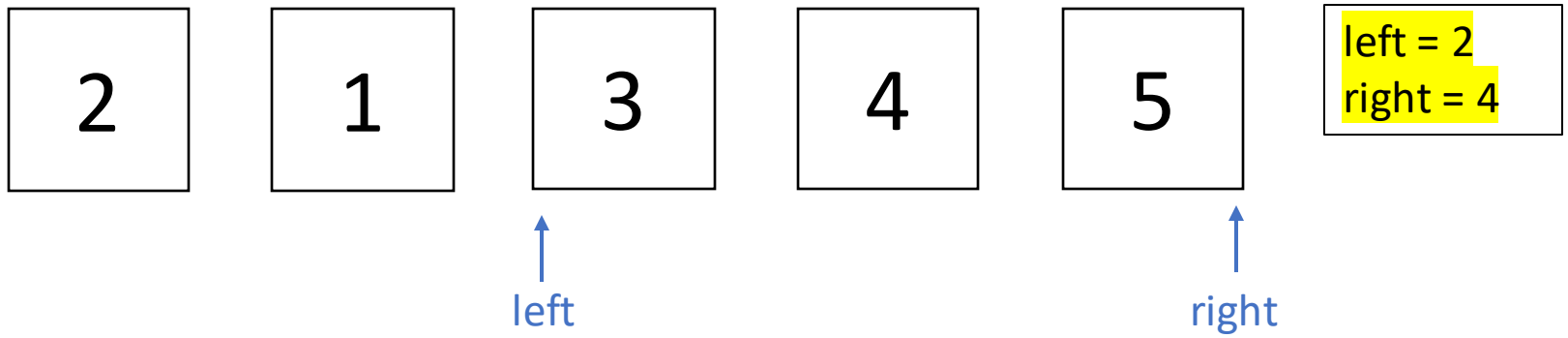
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd reverse operation:**



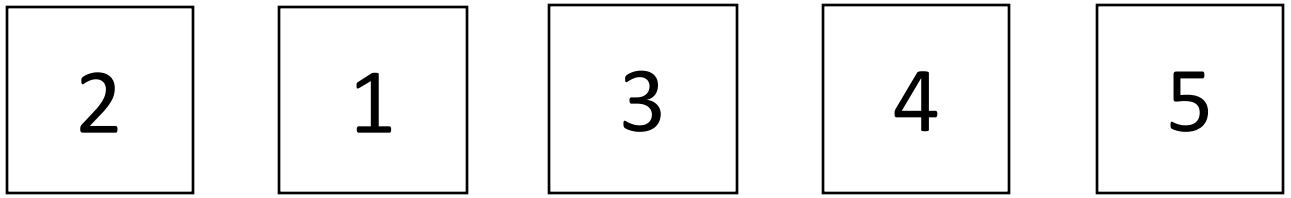

**function call:**
↓ reverse(array, k , n-1) ↓ ( n = 5 )



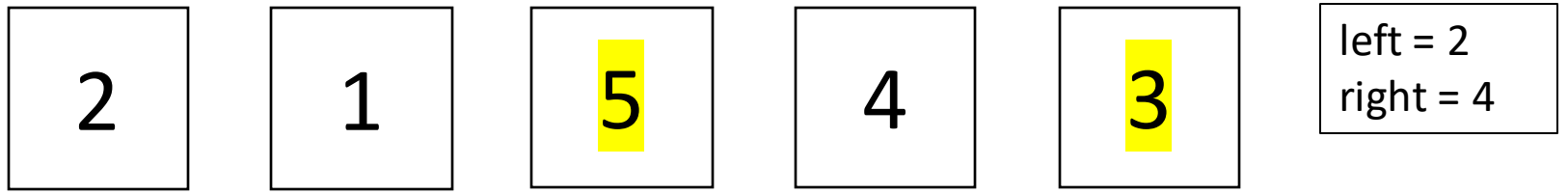
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd reverse operation:**




**function call:**
↓ reverse(array, k , n-1) ↓ ( n = 5 )




left = 2
right = 4

left

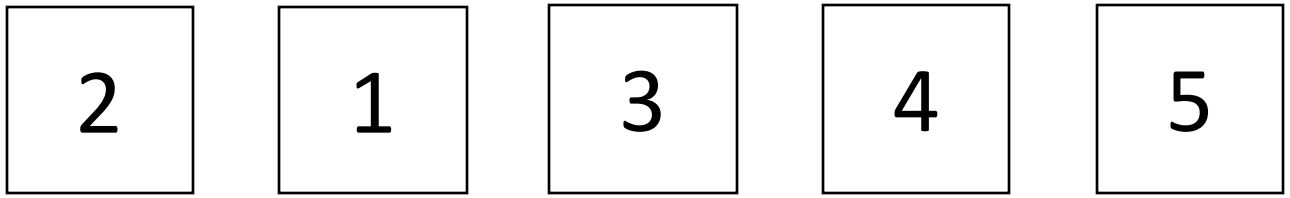
right

swap array[left] and array[right]

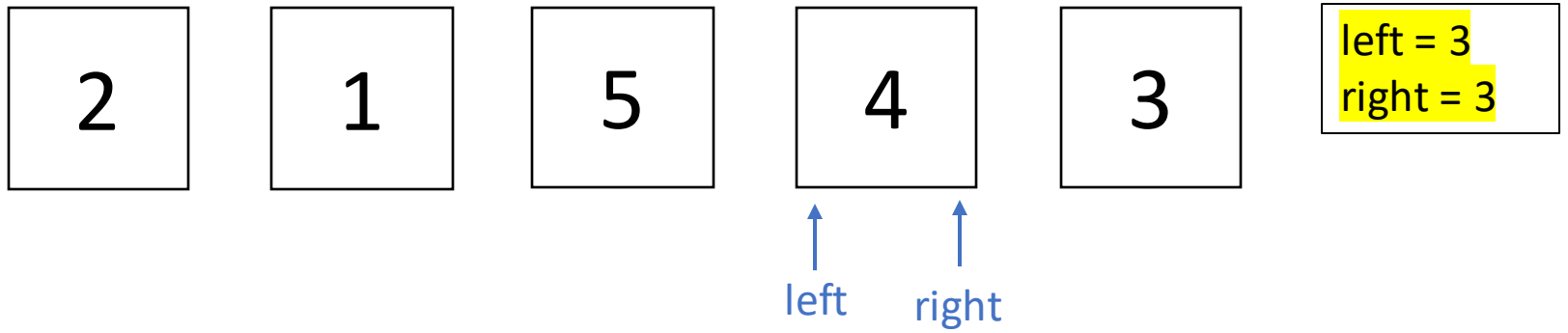
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd reverse operation:**



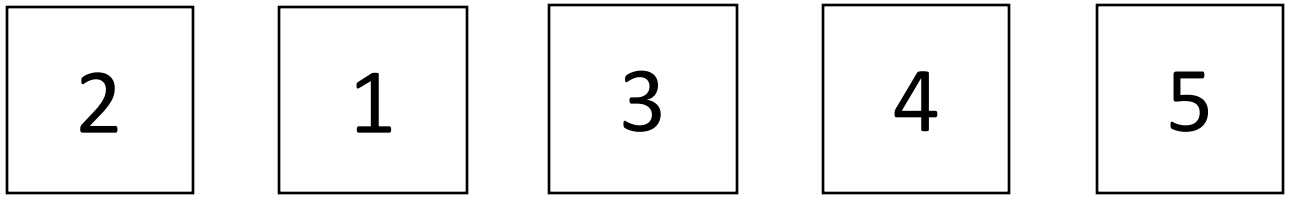

**function call:**
↓ reverse(array, k , n-1) ↓ ( n = 5 )



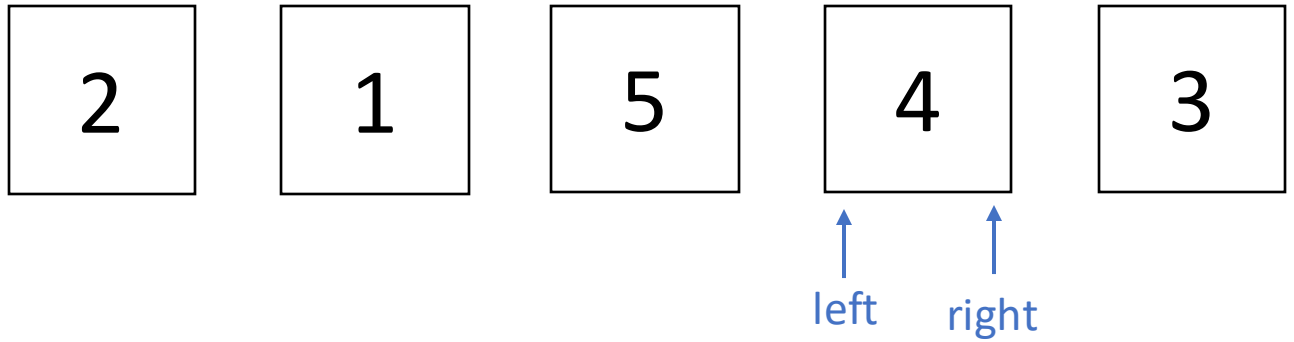
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 2nd reverse operation:**




**function call:**
↓ reverse(array, k , n-1) ↓ ( n = 5 )



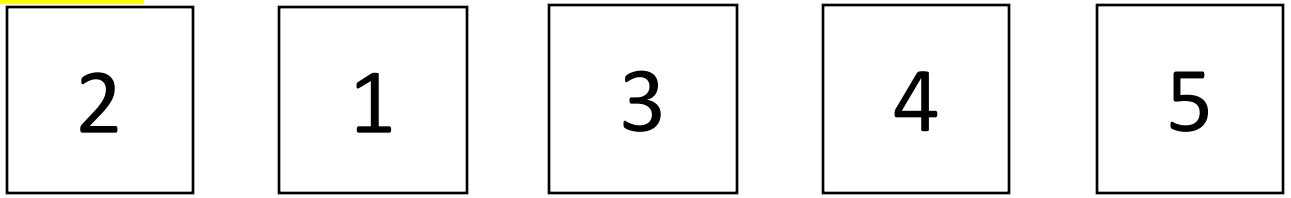

left = 3
right = 3

→ left <? right
**False** → end of the while loop
return array

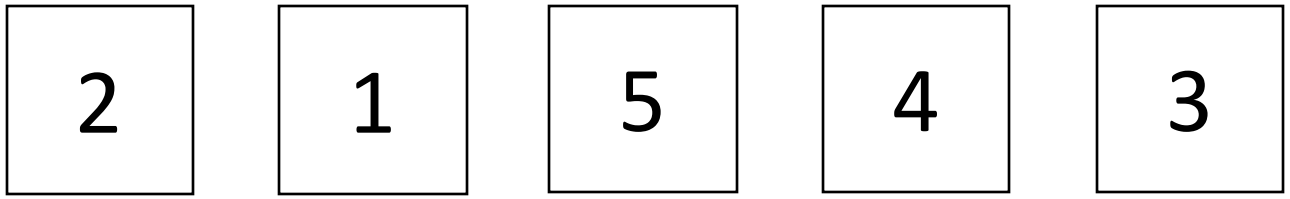
# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Array is updated after the 2nd reverse operation:**
**Before:**




**After:**



# Solution 3 (reversing the array)

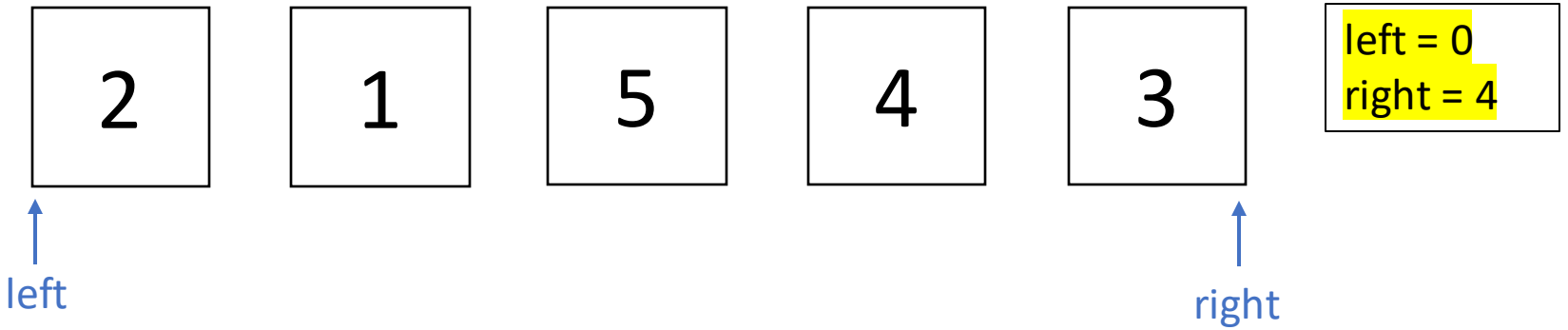
*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd reverse operation:**


**function call:**
↓ reverse(array, 0 , n-1) ↓ ( n = 5 )

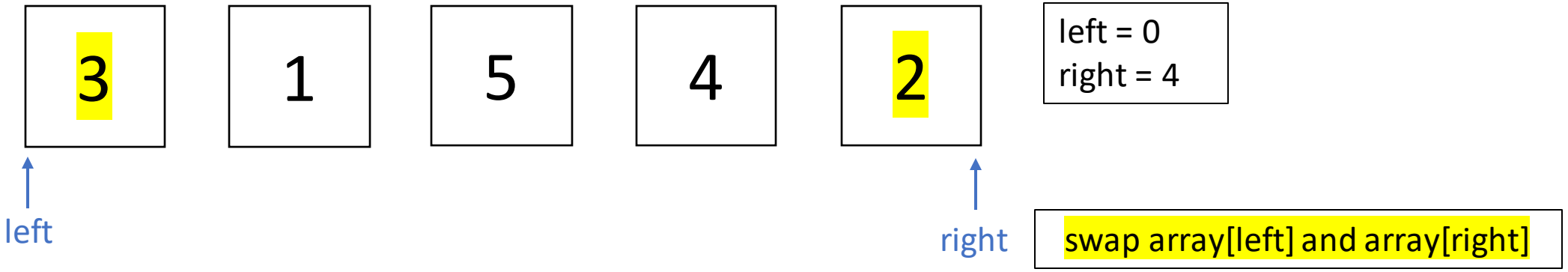


# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd reverse operation:**


**function call:**
↓ reverse(array, 0 , n-1) ↓ ( n = 5 )

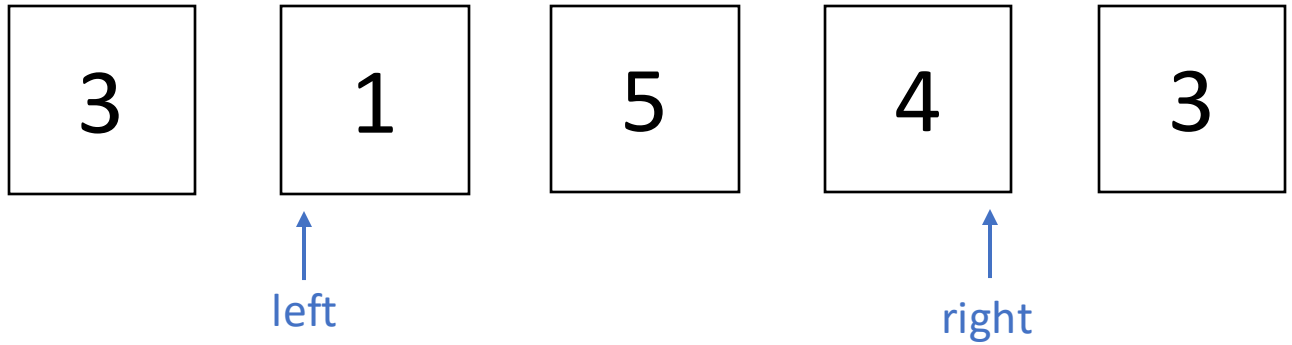


# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd reverse operation:**


**function call:**
↓ reverse(array, 0 , n-1) ↓ ( n = 5 )




left = 1
right = 3

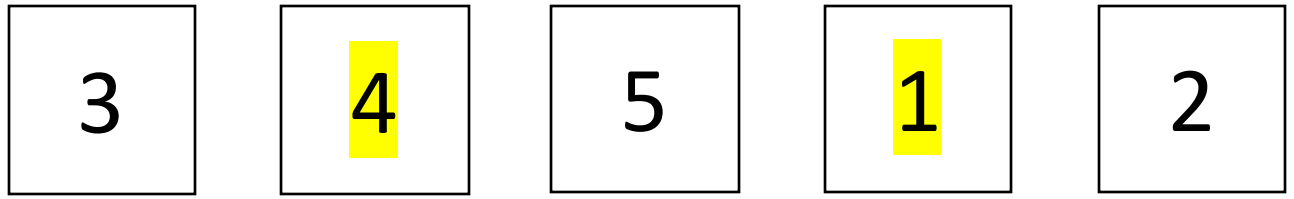
→ left <? right
**True** → continue

# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd reverse operation:**


**function call:**
↓ reverse(array, 0 , n-1) ↓ ( n = 5 )




left = 1
right = 3

left

right

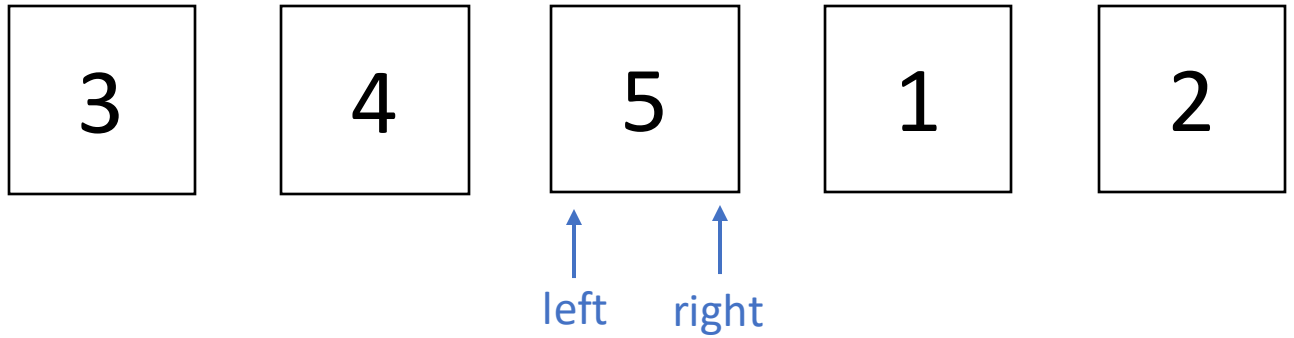
swap array[left] and array[right]

# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Implementing 3rd reverse operation:**


**function call:**
↓ reverse(array, 0 , n-1) ↓ ( n = 5 )




left = 2
right = 3

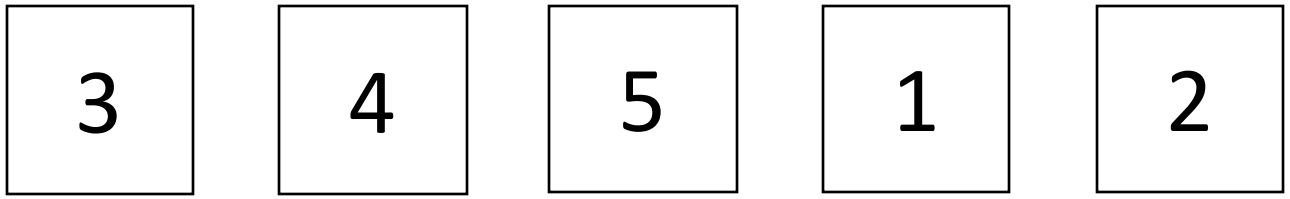
→ left <? right
**False** → end of the while loop
return array

# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**Array is updated after the 3rd reverse operation:**
**Before:**

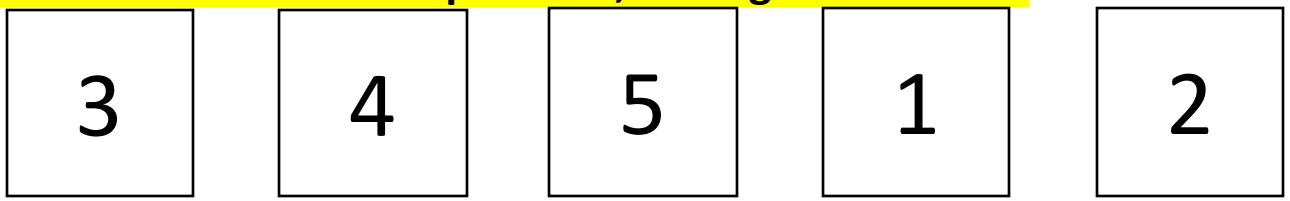

**After:**



# Solution 3 (reversing the array)

*Inputs:*
array = {1, 2, 3, 4, 5} and k = 2

**After the 3rd reverse operation, the algorithm ends.**



# Solution 3 (reversing the array)

## Time Complexity?

The algorithm implements an operation called **reverse** 3 times. This operation uses a while loop.
As in Solution 2, this operation takes at most O(n) time. And if we try to minimize one of these 3 operations, the other one will get closer to n. So, best case and worst case the time complexities are the same: O(n).

## Space Complexity?

The algorithm doesn't use an additional array as in Solution 2. It only uses a temporary variable for swapping operations and iterators. Which makes the space complexity constant.

# Comparison of the solutions

| Solution | Best Case Time Complexity | Worst Case Time Complexity | Space Complexity |
|---|---|---|---|
| Solution 1 (naïve) | Ω(1) | O(n^2) | O(1) |
| Solution 2 (auxiliary array) | Ω(n) | O(n) | O(k) |
| Solution 3 (reverse array) | Ω(n) | O(n) | O(1) |