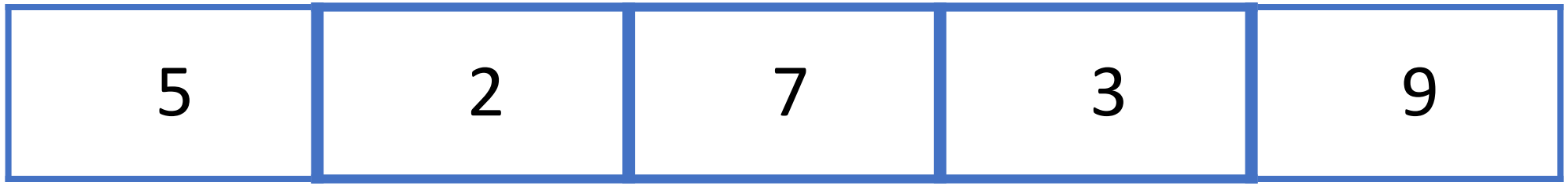
Selection sort & Insertion sort

Cristóbal González

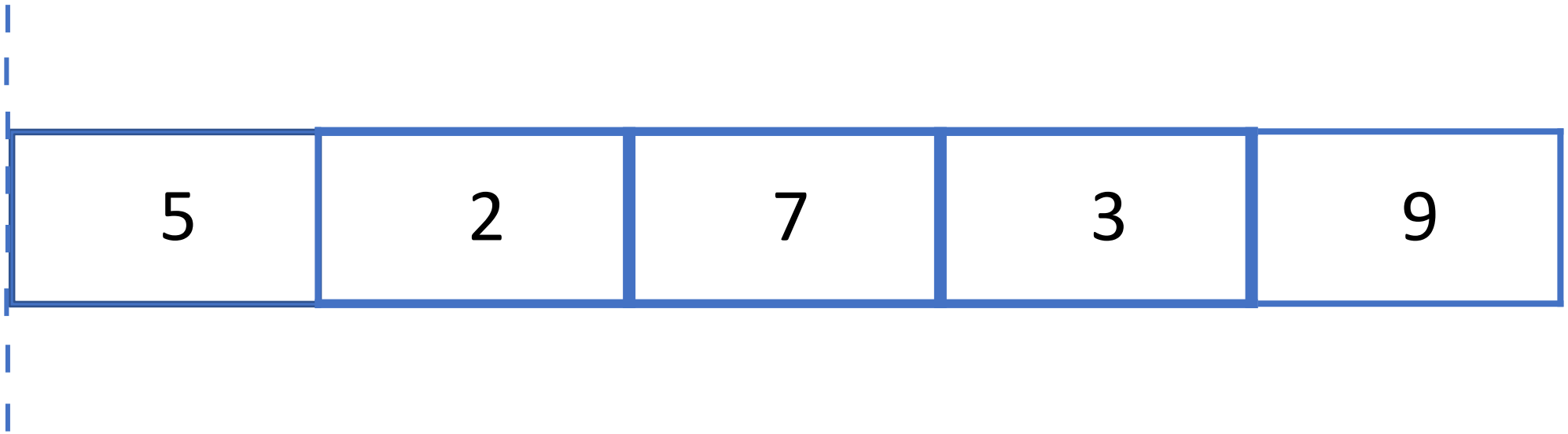
Selection Sort

1. Tenemos una secuencia desordenada
2. Buscar el menor dato ‘x’ en la secuencia
3. Intercambiar ese elemento ‘x’ con el primer elemento de la secuencia
4. Avanzar uno en la secuencia
5. Si aún queda secuencia, volver a 2

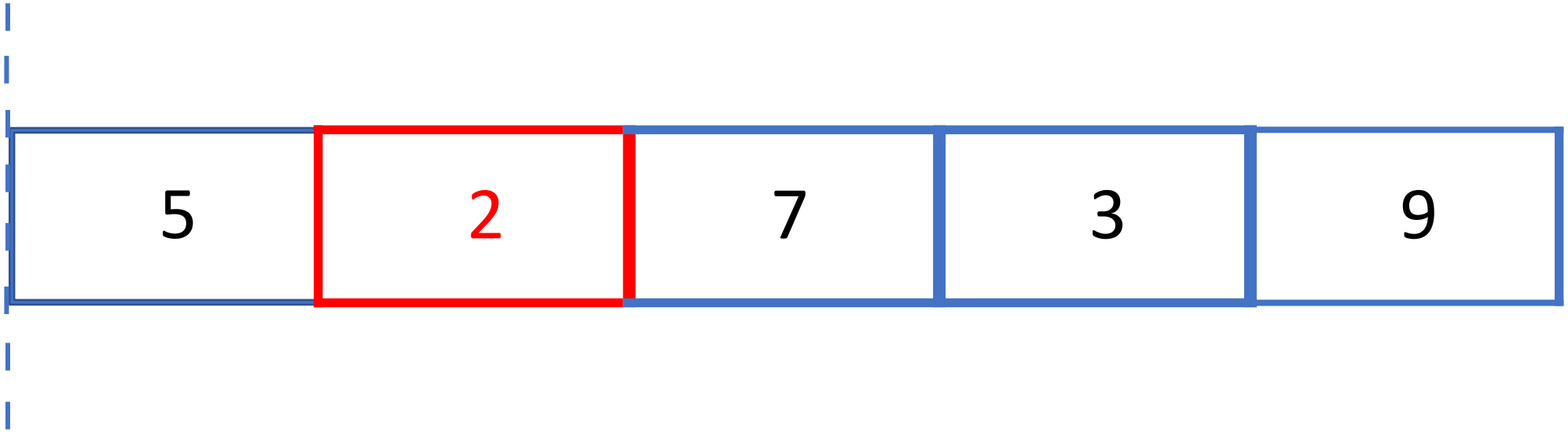
Veamos un ejemplo…



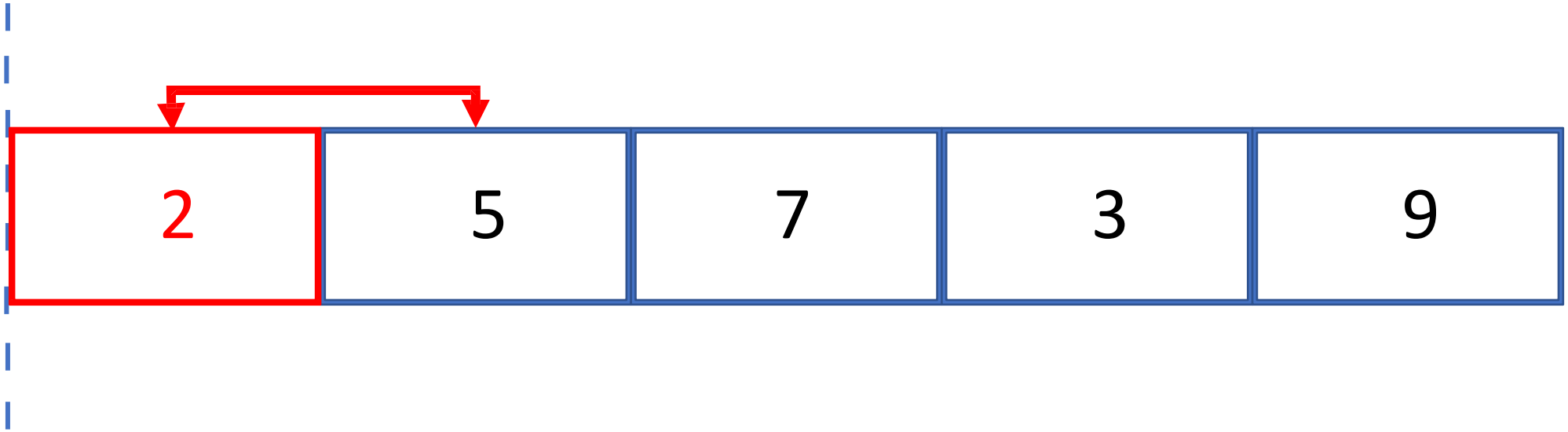
Partimos al principio



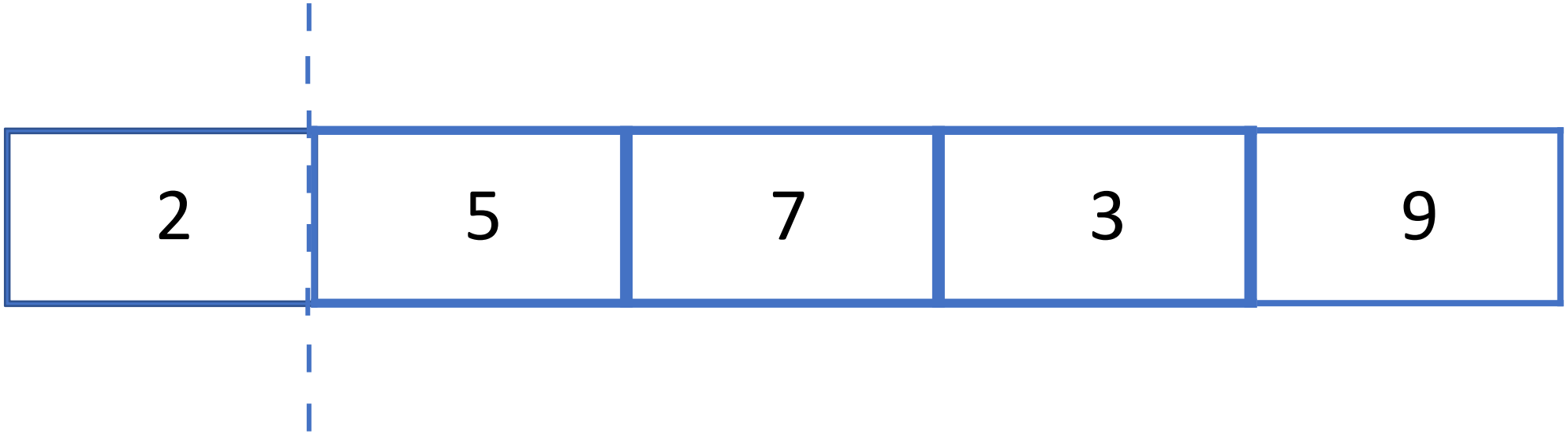
Seleccionamos el menor



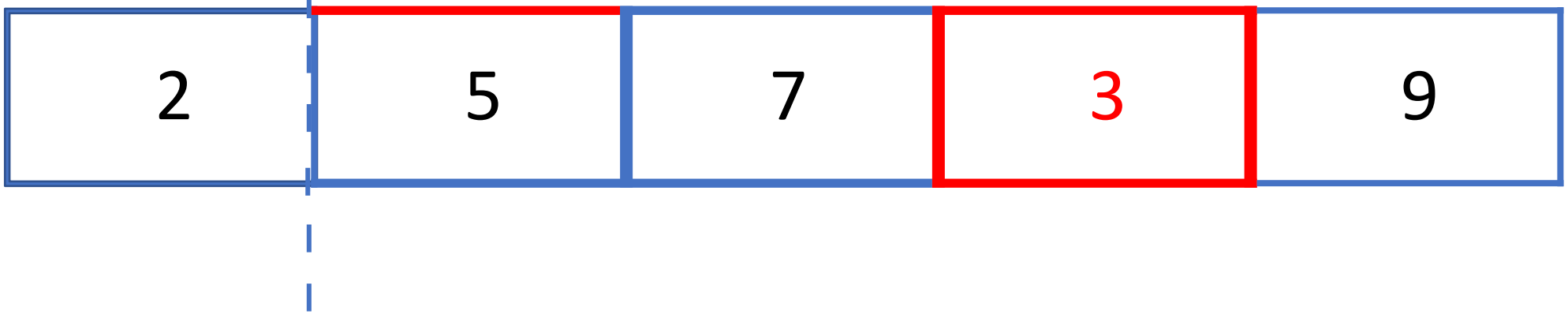
Intercambiamos a la primera posición



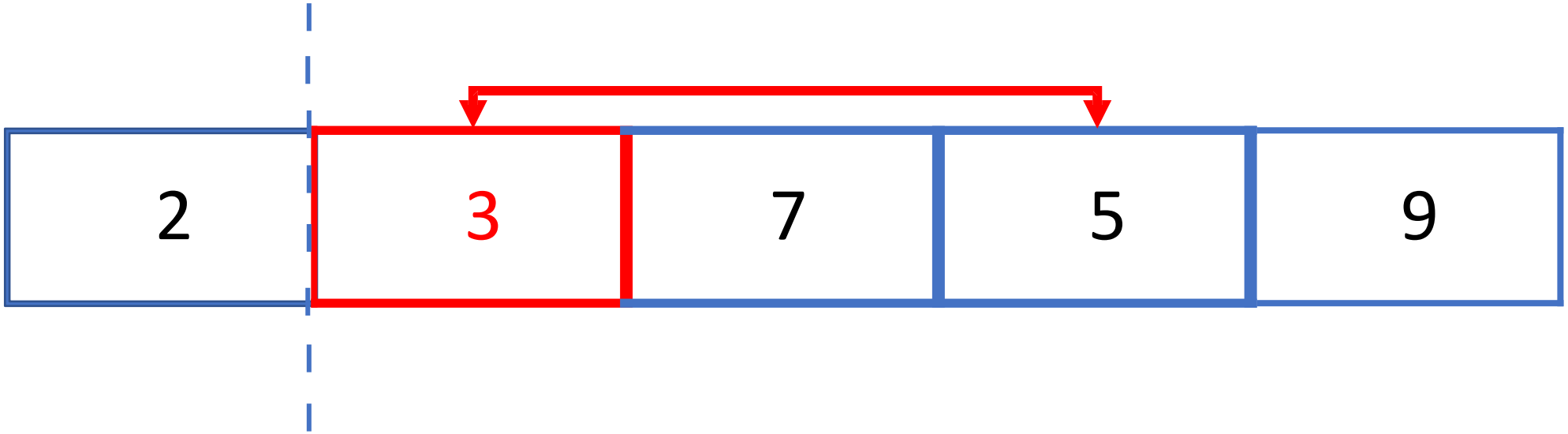
Nos movemos a la segunda posición



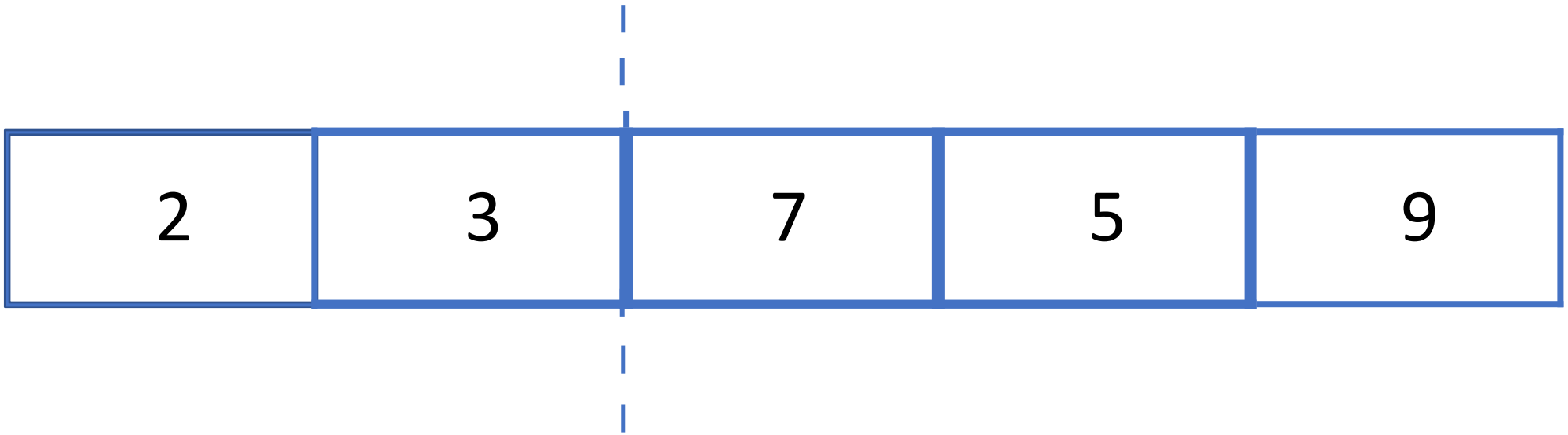
Seleccionamos el menor…



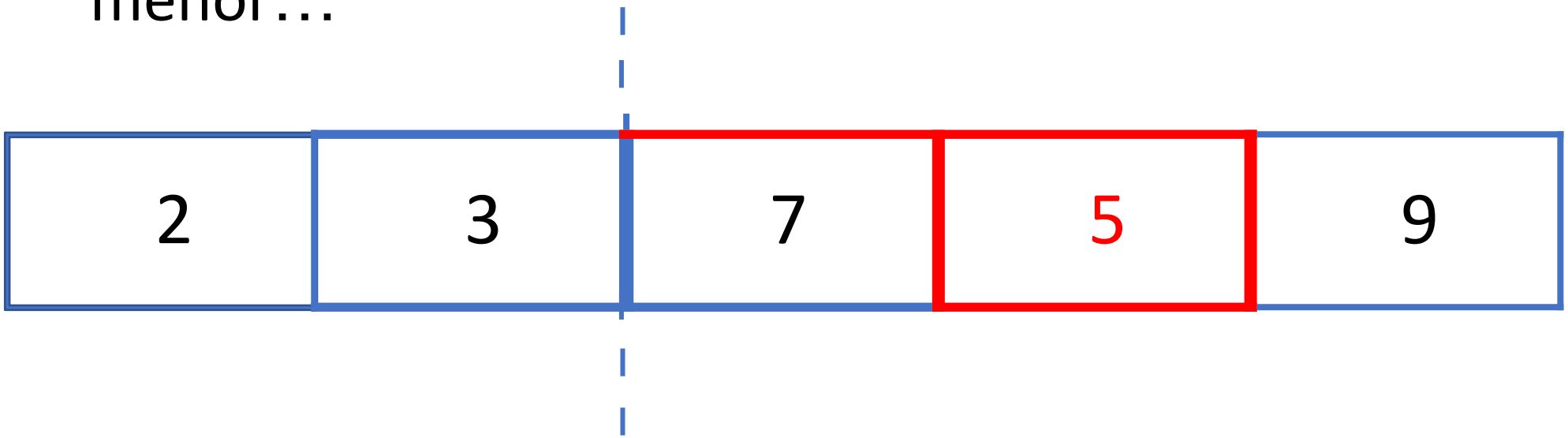
Intercambiamos con el primero



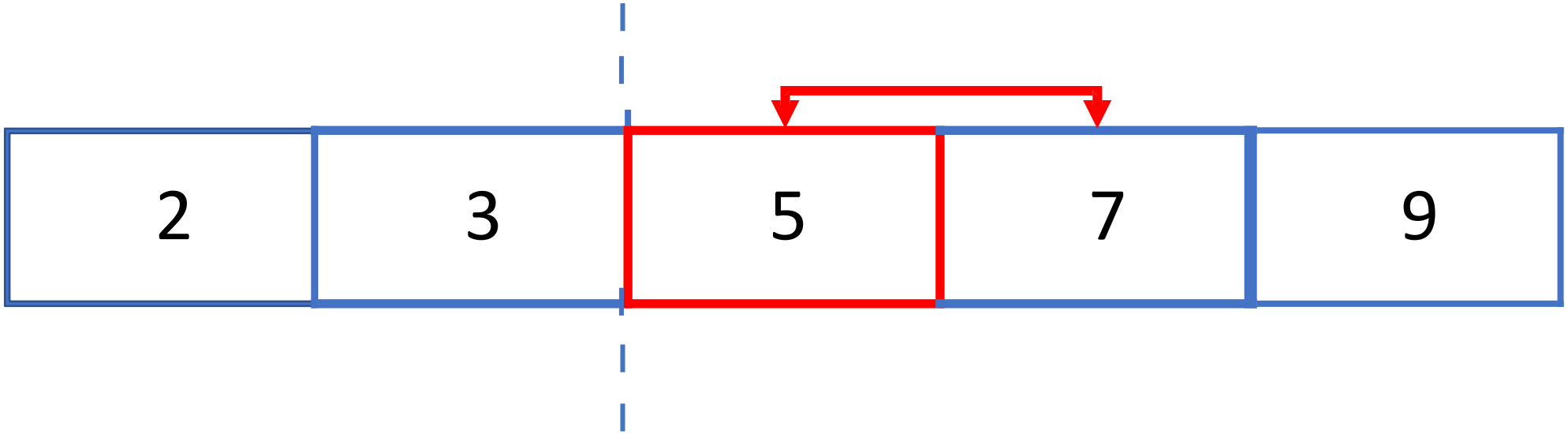
Nos movemos a la tercera posición



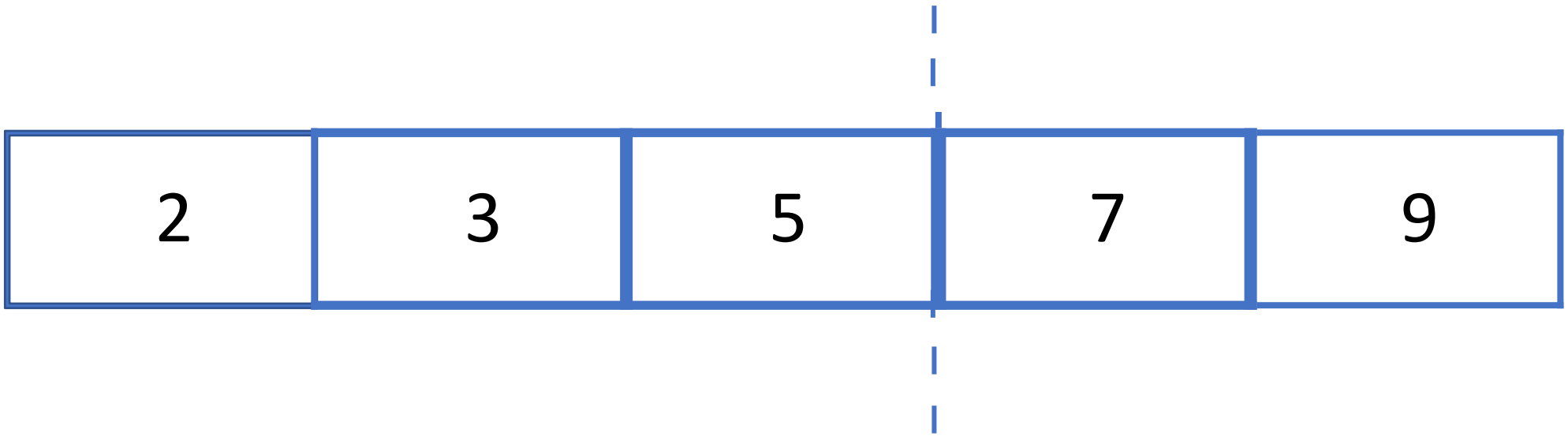
Seleccionamos el menor…



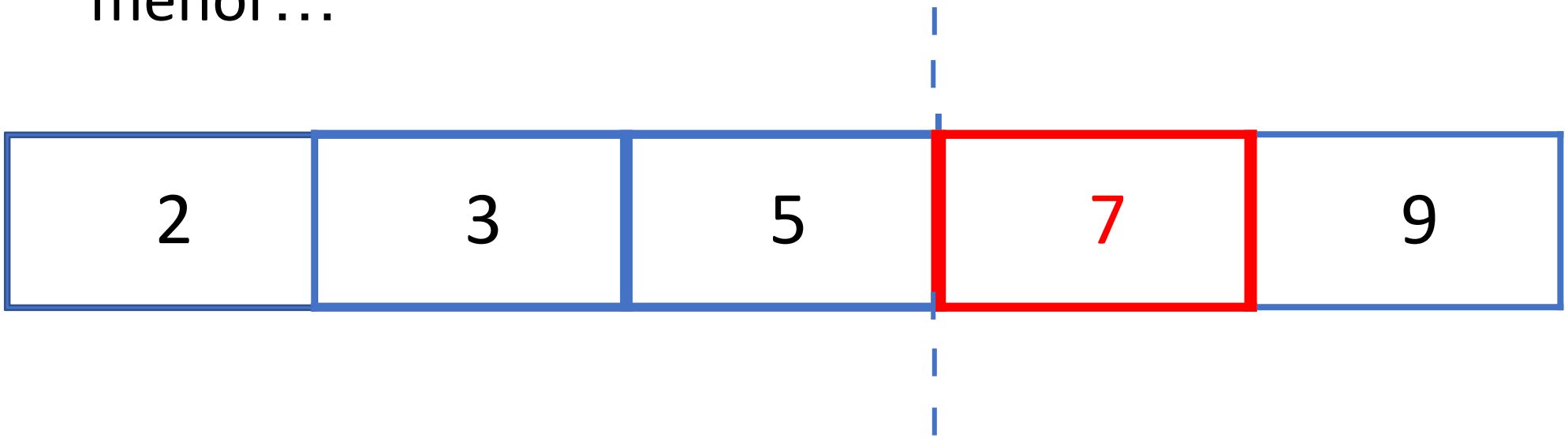
Intercambiamos con el primero



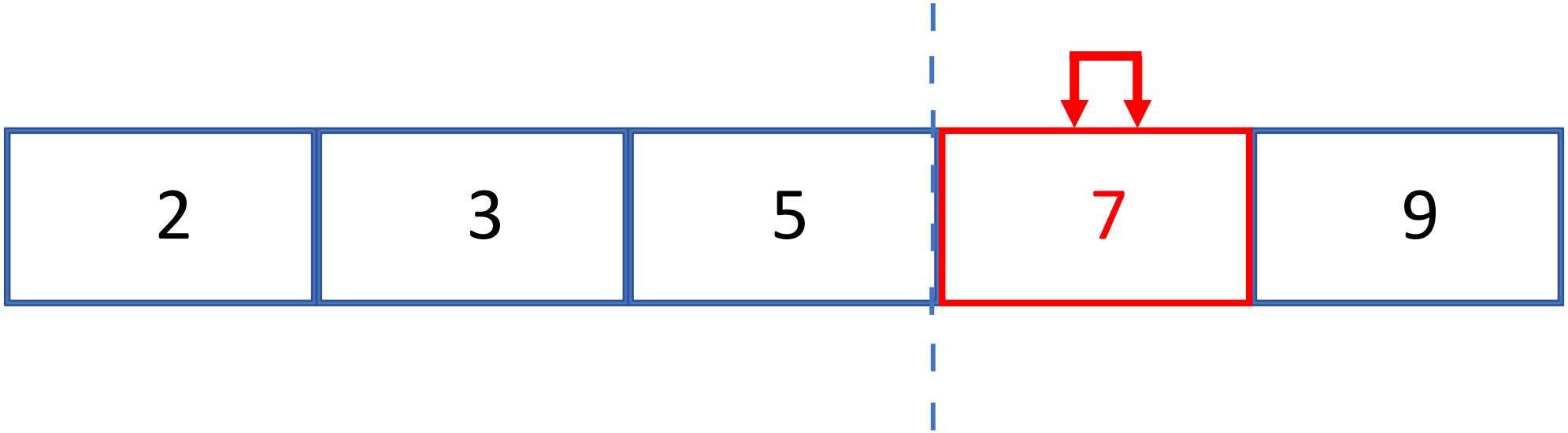
Nos movemos a la cuarta posición



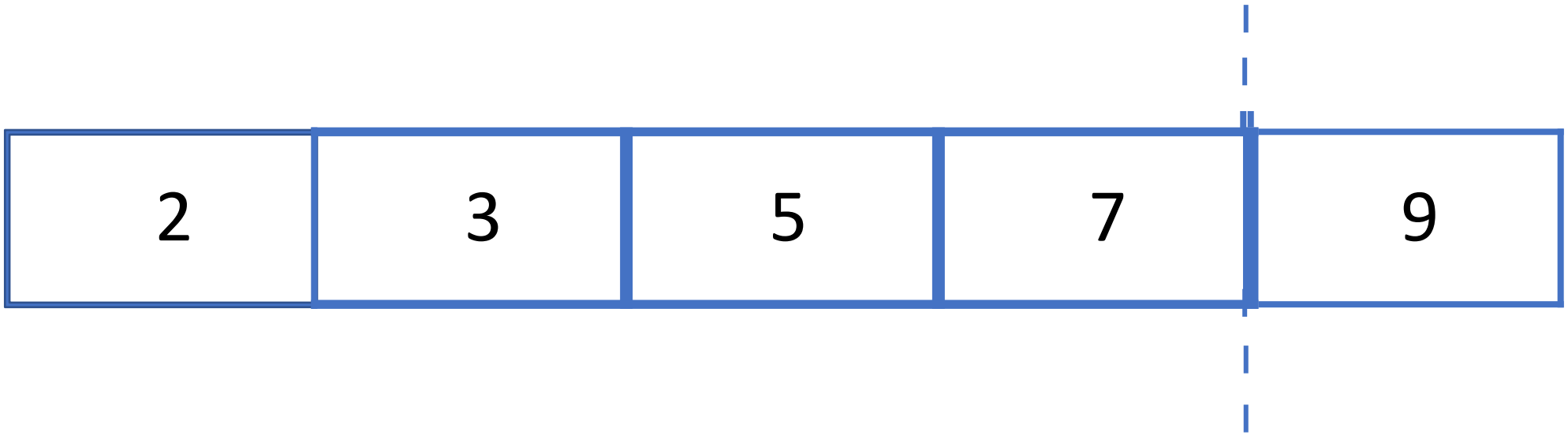
Seleccionamos el menor…



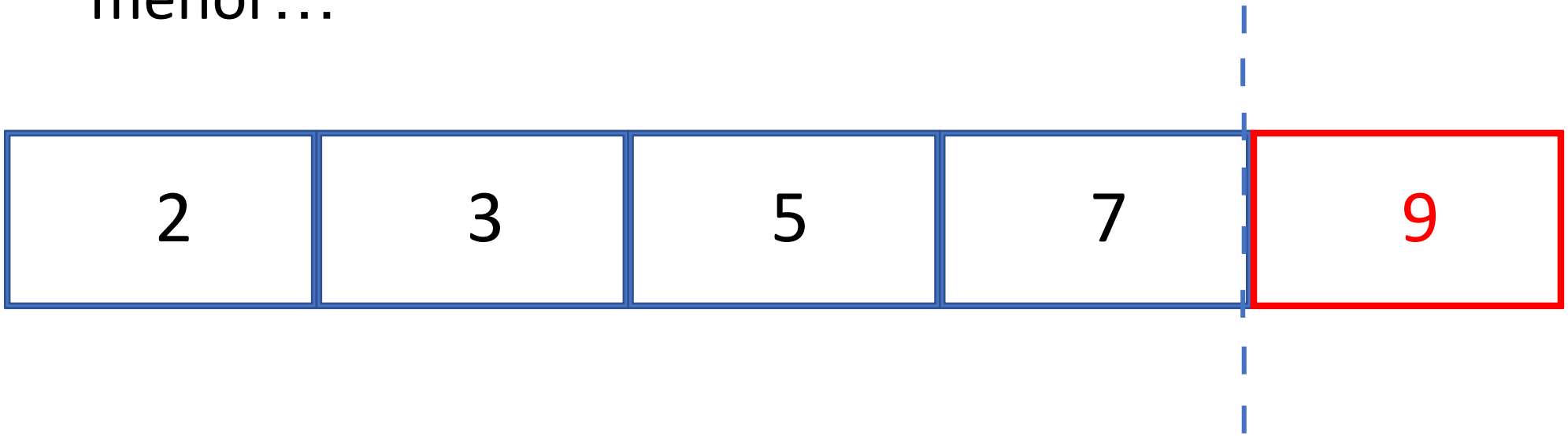
Intercambiamos con el primero



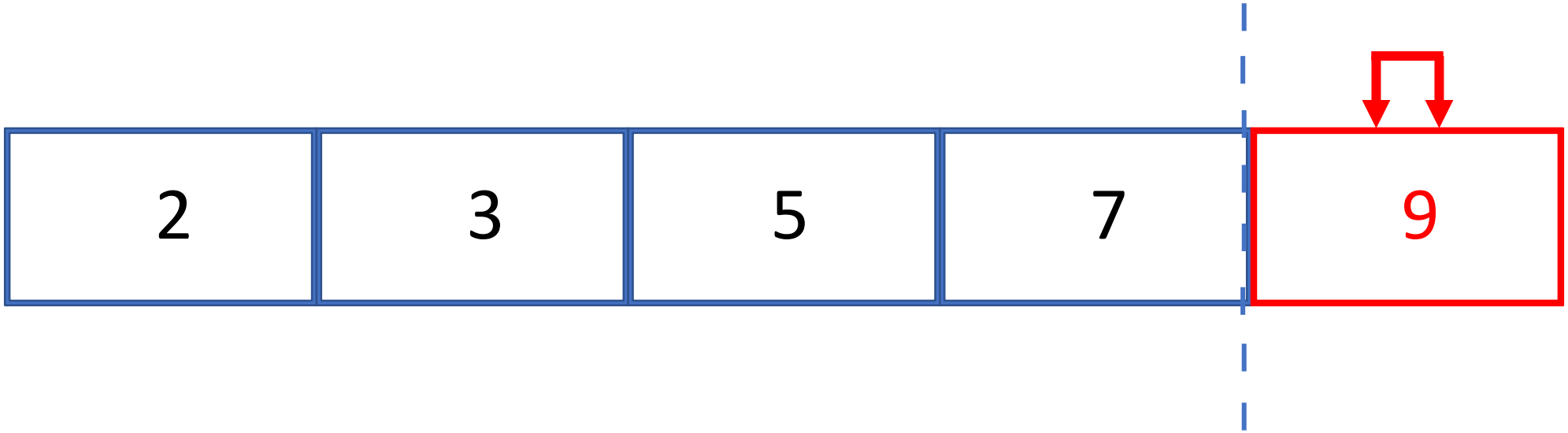
Nos movemos a la quinta posición



Seleccionamos el menor…



Intercambiamos con el primero



Quedo ordenado!!

Como se programaría en C?

```c
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
```

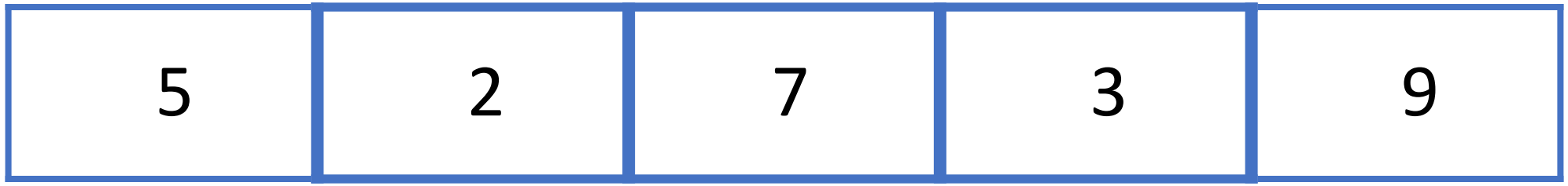
Funcion para cambiar contenido entre 2 punteros

```c
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}
```

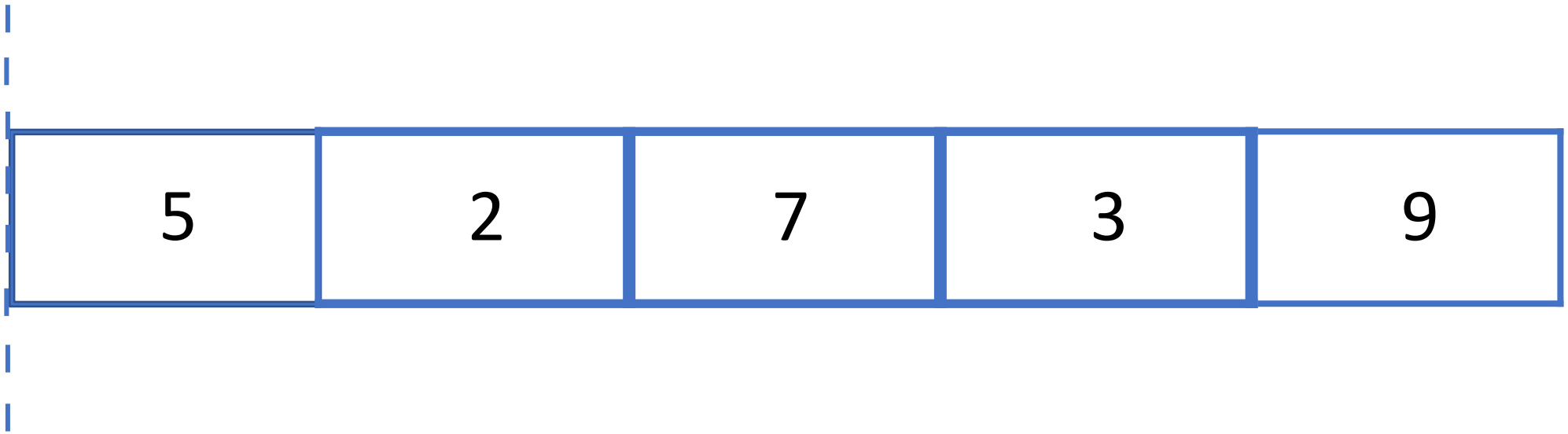
Insertion Sort

1. Tenemos una secuencia desordenada
2. Tomar el primer dato ‘x’ de la secuencia
3. Insertar ‘x’ en los elementos anteriores de manera que quede ordenado
4. Avanzar uno en la secuencia
5. Si quedan elementos en la secuencia, volver a 2

Veamos el mismo ejemplo anterior…

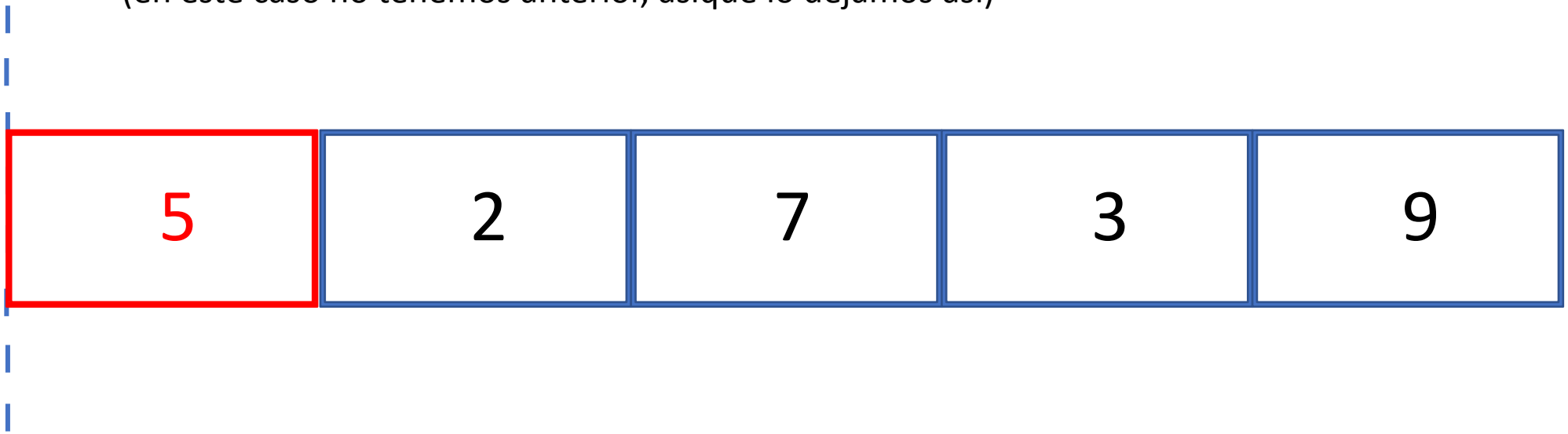


Partimos al principio

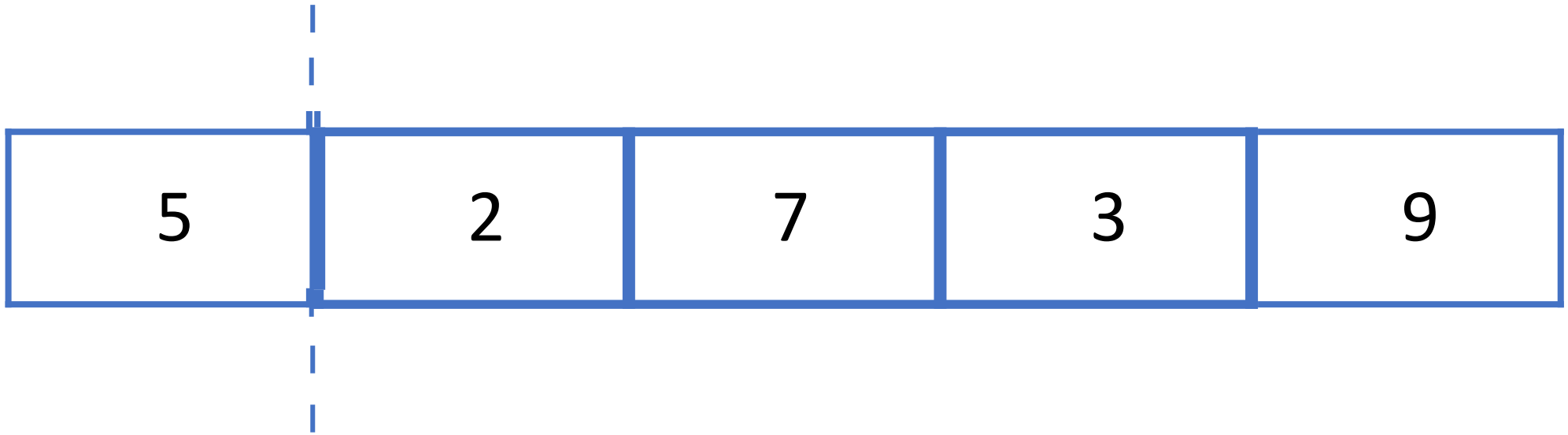


Comparamos el primero con el anterior

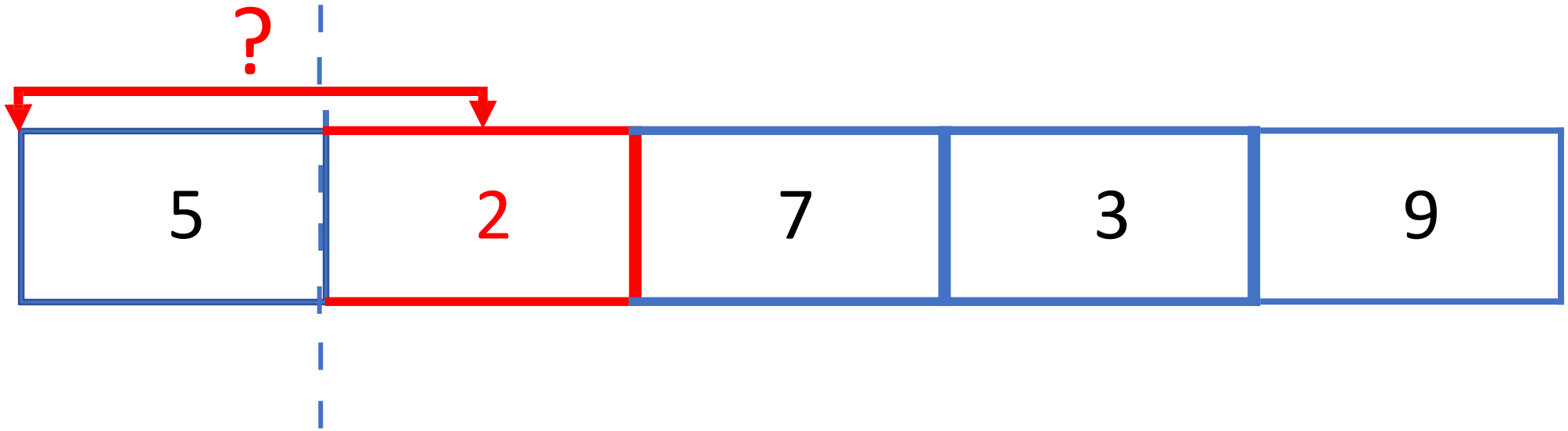
(en este caso no tenemos anterior, asique lo dejamos así)



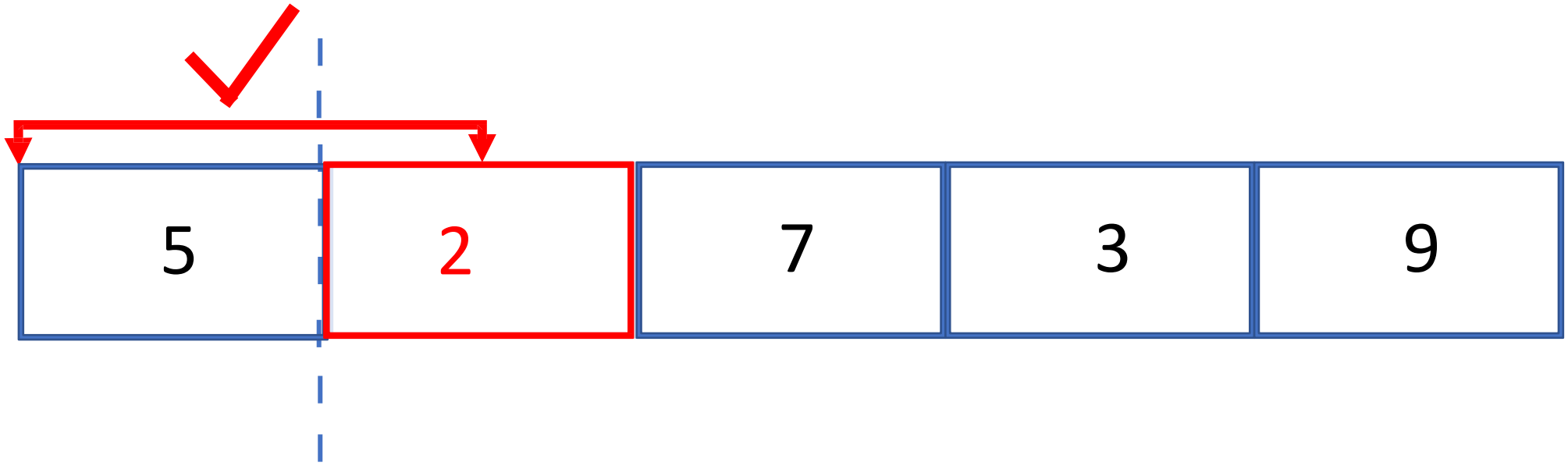
Nos movemos a la segunda posición



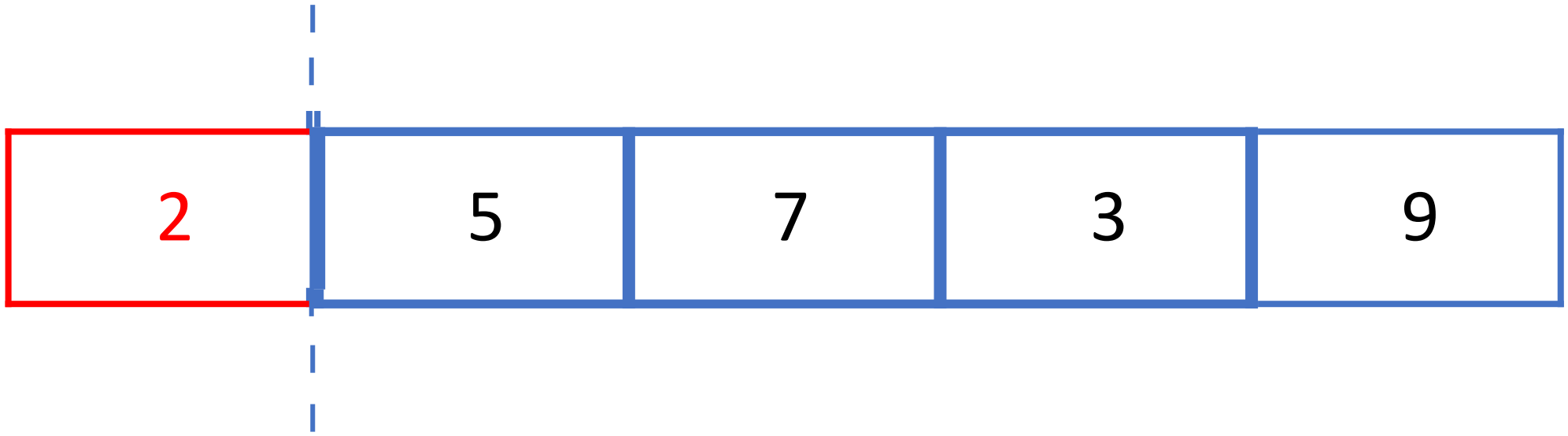
Vemos en qué posición debería quedar el primero



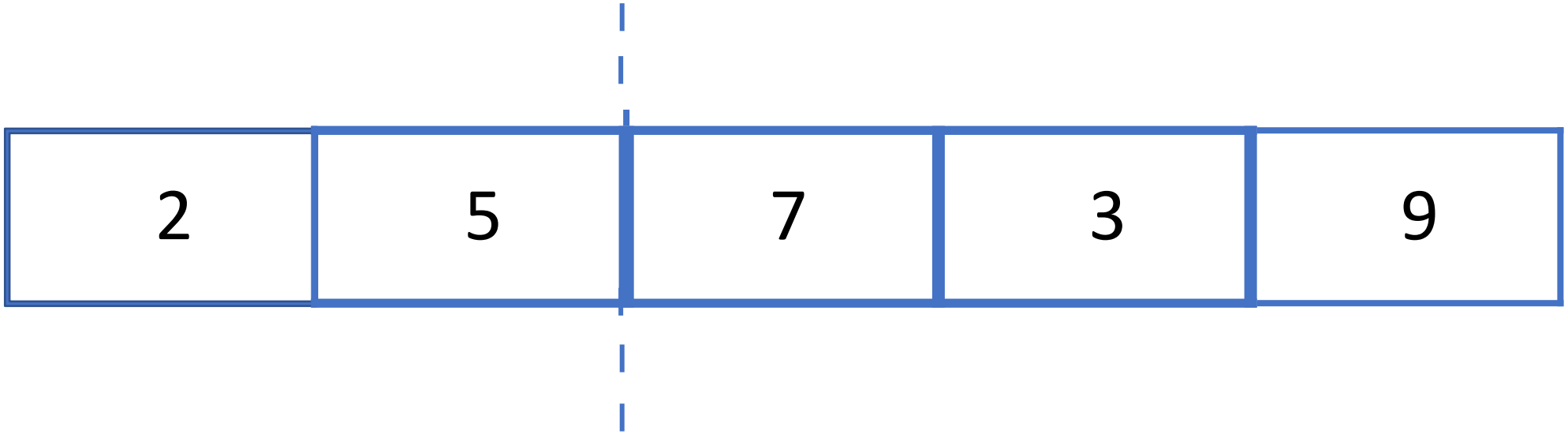
Como sabemos que va atrás, movemos la casilla.



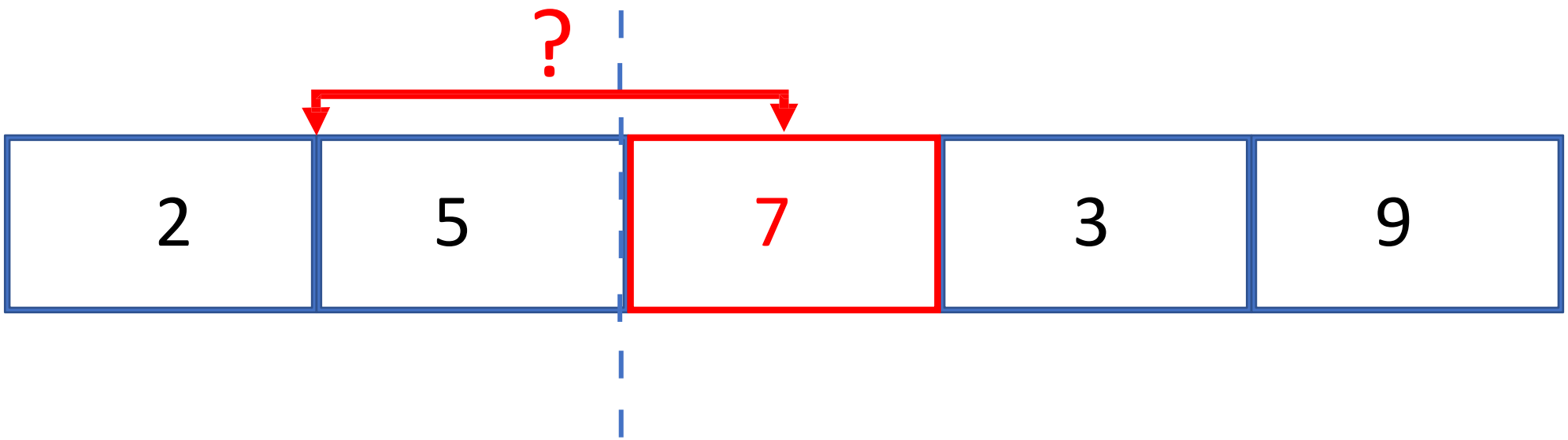
Reposicionamos



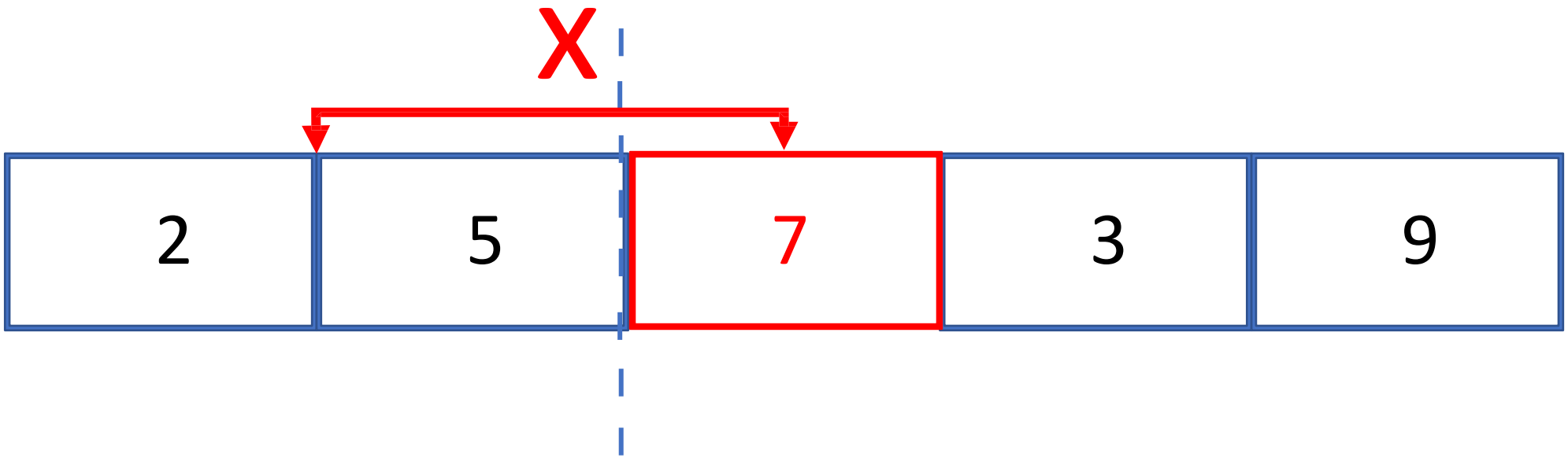
Nos movemos a la tercera posición



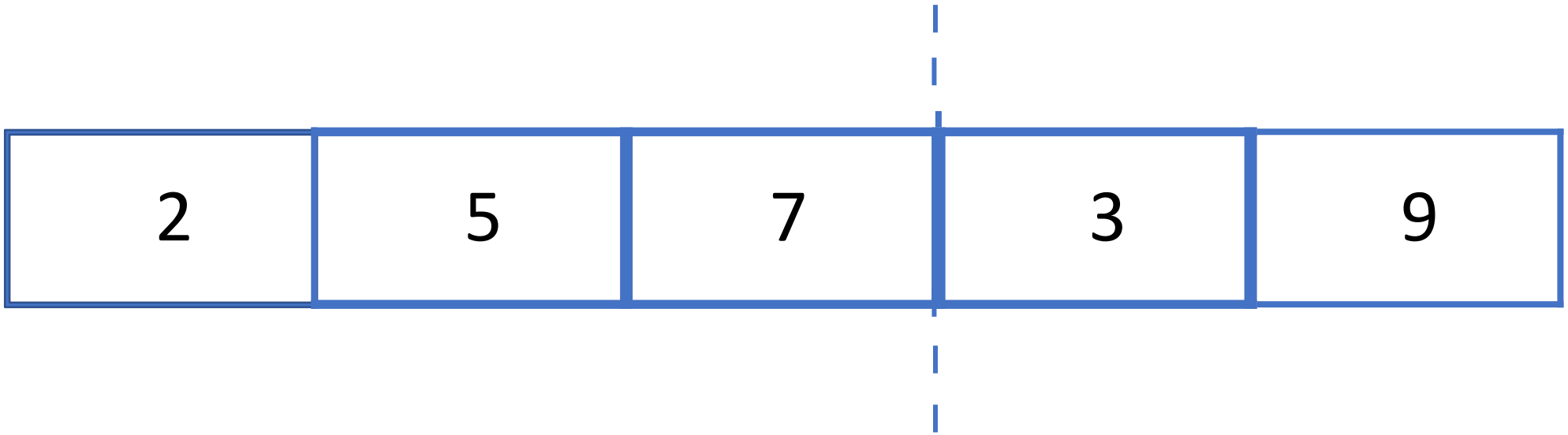
Vemos en qué posición debería quedar el primero



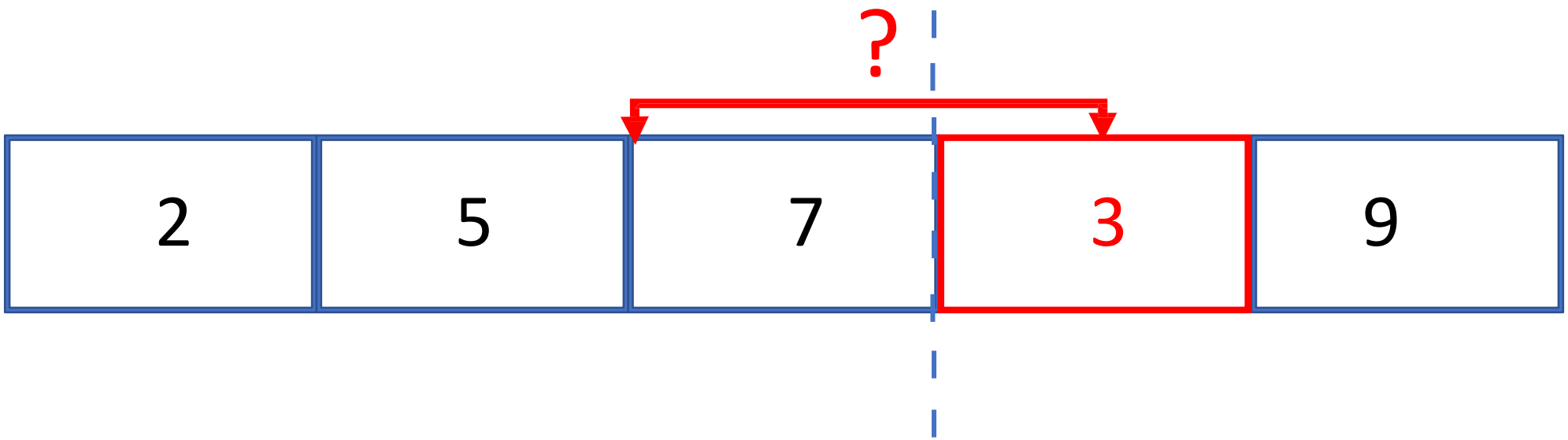
No pasa nada, lo dejamos así



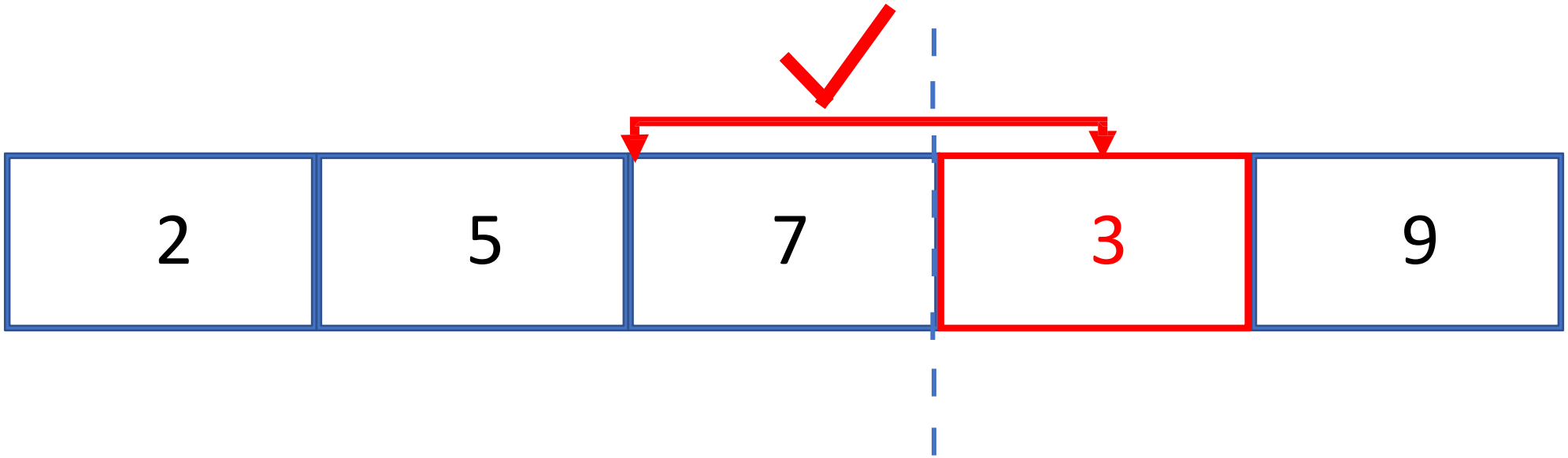
Nos movemos a la cuarta posición



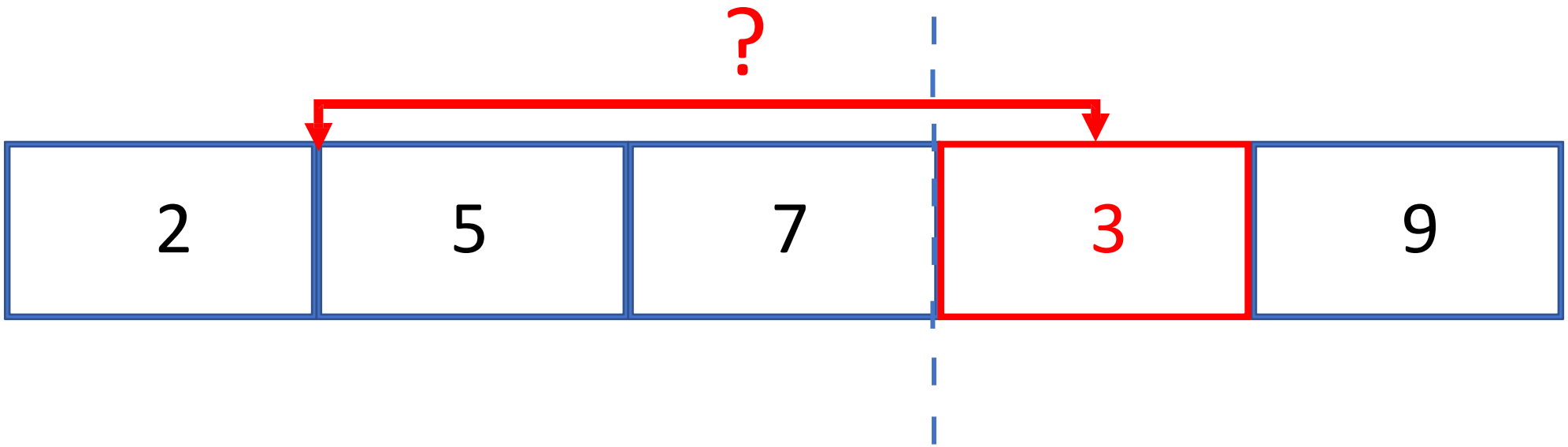
Vemos en qué posición debería quedar el primero



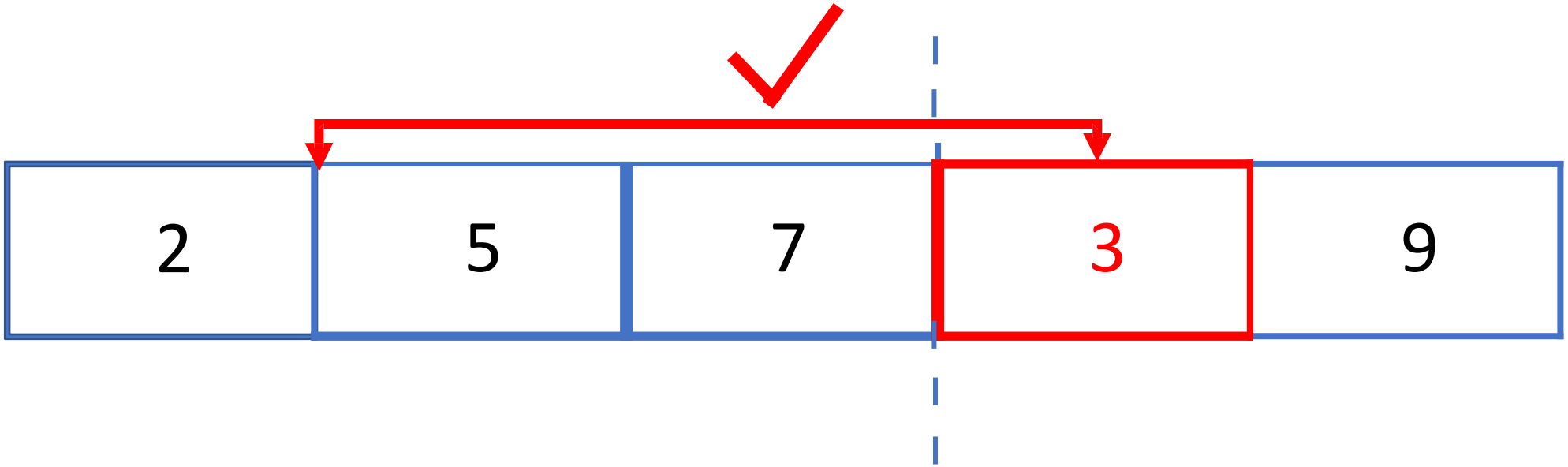
Sabemos que es menor, pero no necesariamente va ahí



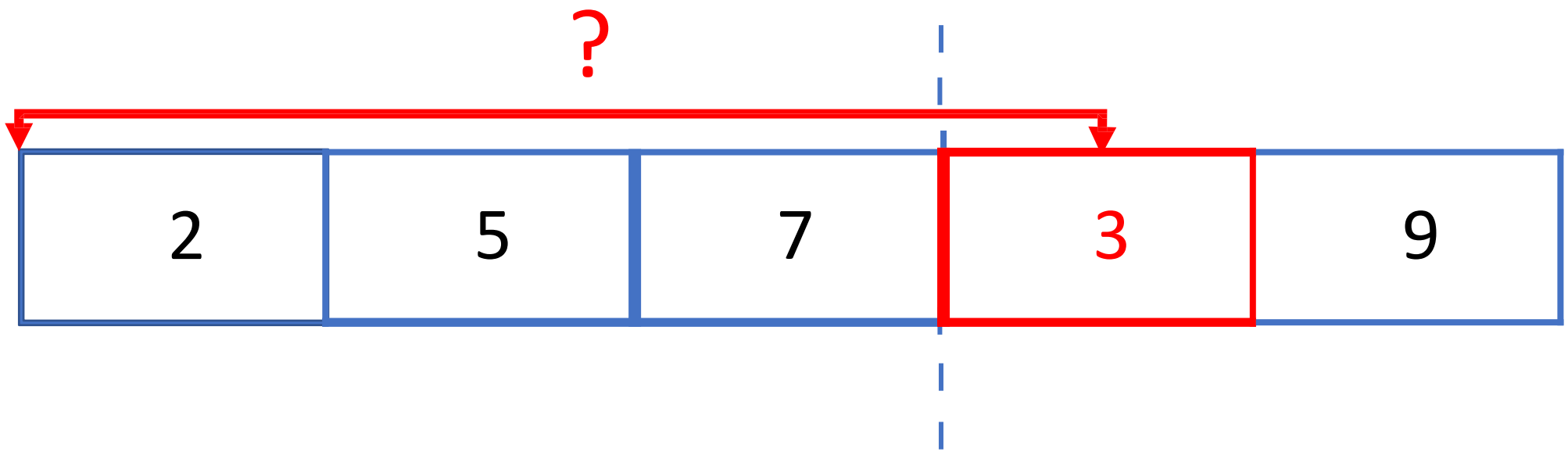
Vemos en otra posición



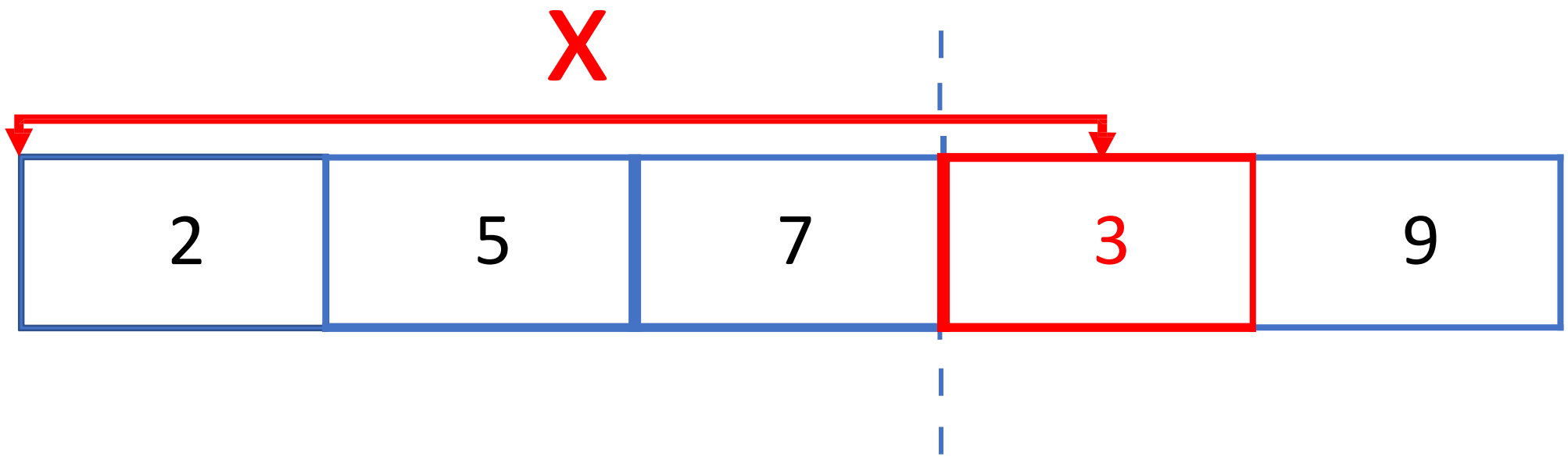
Sabemos que es menor, pero no necesariamente va ahí



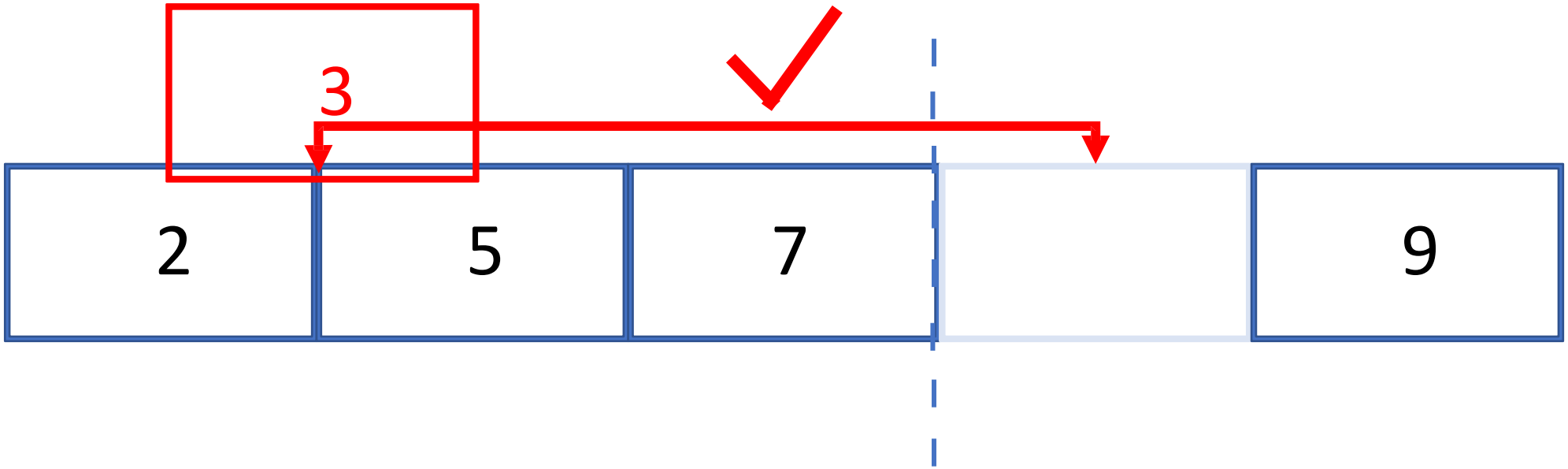
Sabemos que es menor, pero no necesariamente va ahí



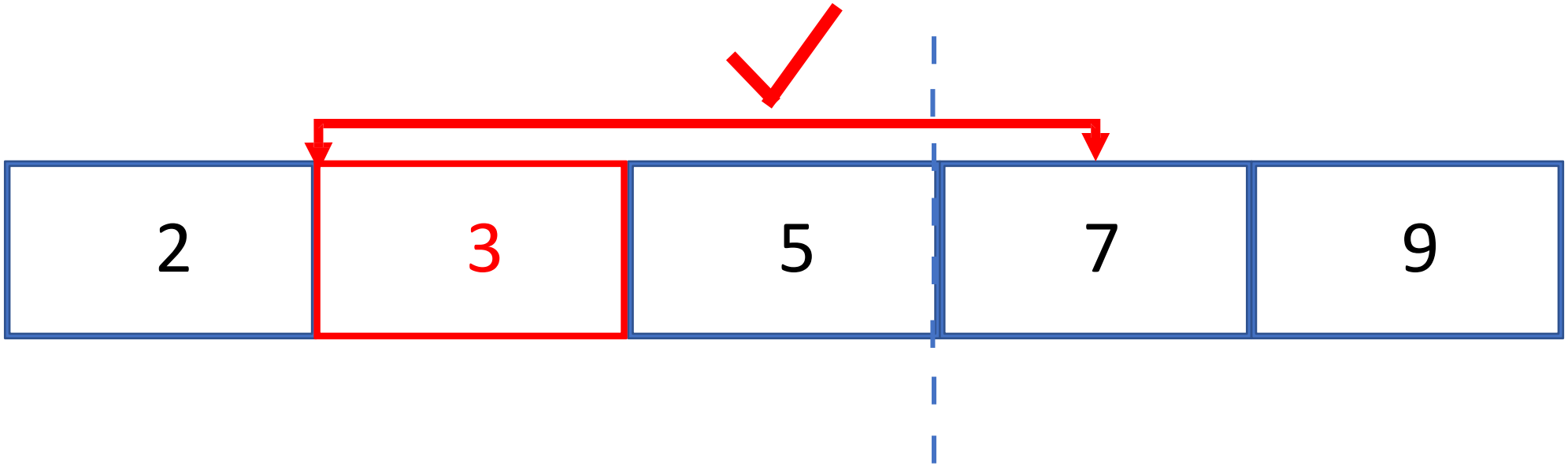
Entonces era el de antes



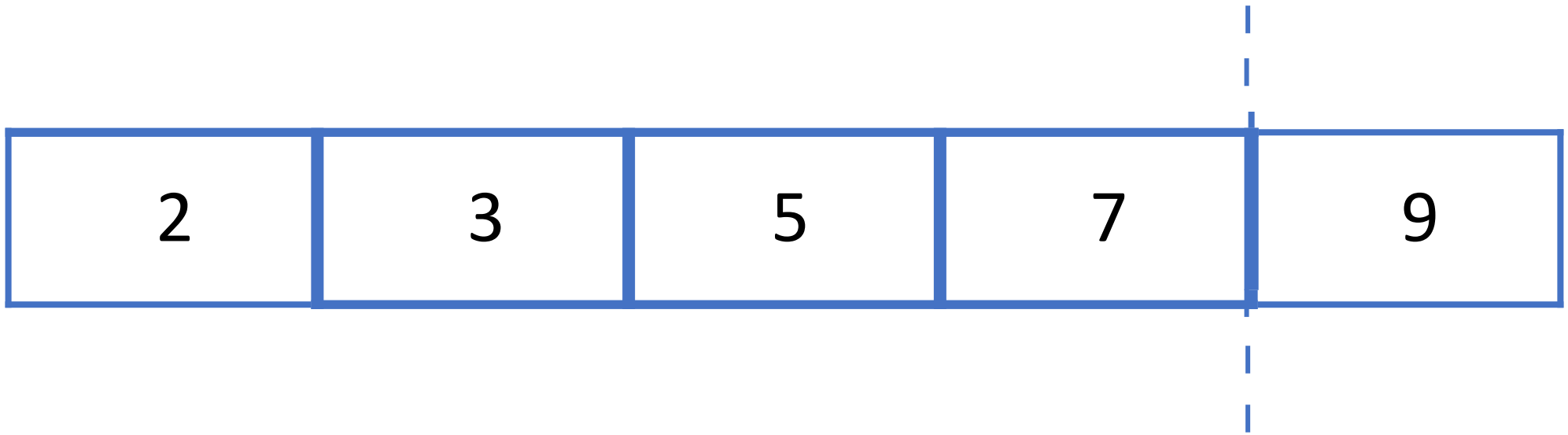
Colocamos donde corresponde y movemos los otros.



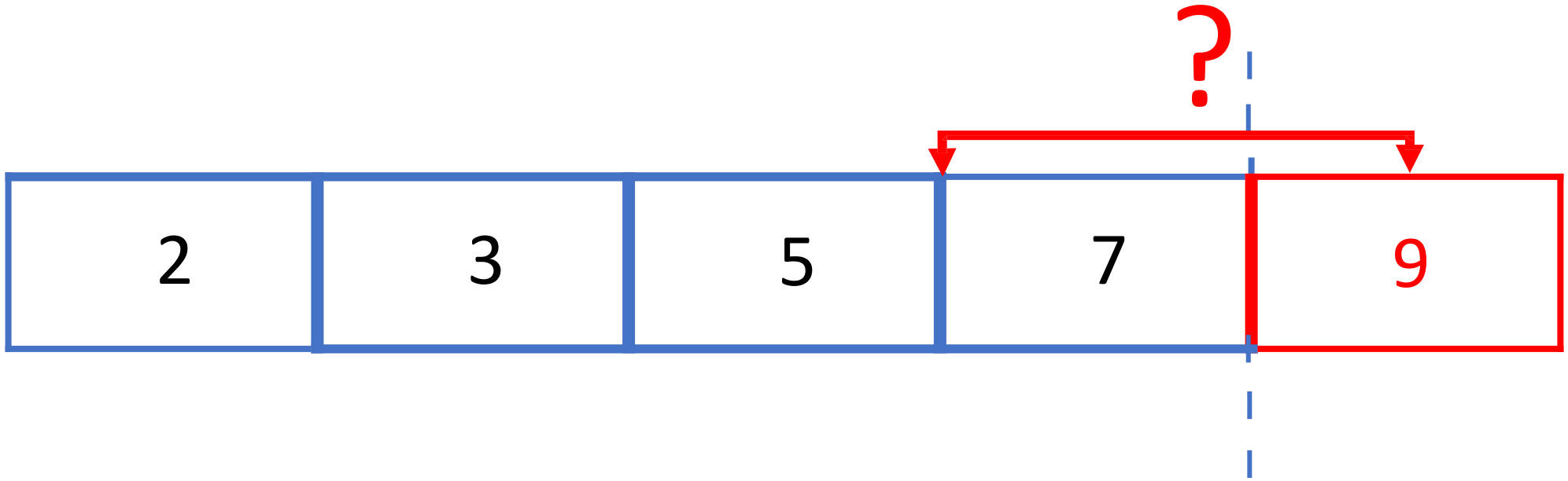
Sabemos que es menor, pero no necesariamente va ahí



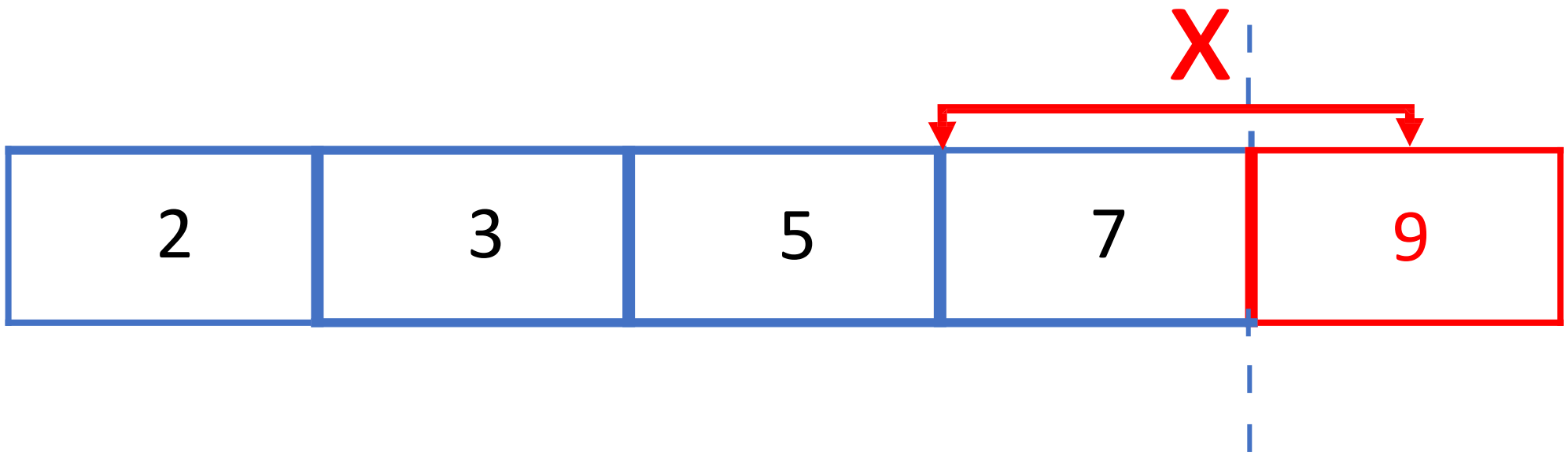
Nos movemos a la quinta posición



Vemos en qué posición debería quedar el primero



No pasa nada, lo dejamos así



Esta ordenado!!

Como se programaría en C?

```
1   /* Función que ordena un array utilizando insertion sort*/
2   void insertionSort(int arr[], int n)
3   {
4       int i, key, j;
5       for (i = 1; i < n; i++)
6       {
7           key = arr[i];
8           j = i - 1;
9
10          /* Mueve los elementos arr[0..i-1],
11          mayores que key 1 espacio hacia
12          adelante */
13          while (j >= 0 && arr[j] > key)
14          {
15              arr[j + 1] = arr[j];
16              j = j - 1;
17          }
18          arr[j + 1] = key;
19      }
20  }
```

Correctitud

- Un algoritmo se dice que es **correcto** si, para todo input válido, se cumple que:
  - El algoritmo termina en tiempo finito.
  - Se obtiene el resultado esperado.

- Para demostrar la correctitud de un algoritmo se suele utilizar **inducción**, debido a su buena compatibilidad con problemas de tamaño creciente.

Inducción

1. Se comienza con una **base de inducción**, la cual es un input inicial que cumple con el objetivo.

2. Luego, se plantea una **hipótesis de inducción**, que es una declaración que asumimos como verdadera. Generalmente se asume que el objetivo se cumple con un input ***N***.

3. Y finalmente, se propone la **tesis de inducción**, la cual se demuestra utilizando la hipótesis. Se demuestra para un input ***N + 1***.

Ejemplo Inducción:

Por demostrar: **6 divide a $n^3 - n$** para todo **n** natural.

1. **BI:** Para **n** = 1;

   $1^3 - 1 = 0 = 0 * 6$ 6 divide a $1^3 - 1$

2. **HI:** Asumimos que la afirmación se cumple para un número **N**.

   6 divide a $\mathbf{N^3 - N}$

3. **TI:** Demostramos que la afirmación se cumple para **N + 1**.

$$\begin{aligned}(N+1)^3 - (N+1) &= (N+1)(N^2+2N)\\ &= N^3 + 3N^2 + 2N\\ &= (N^3 - N) + (3N^2 + 3N)\\ &= 6k + 3N(N+1)\\ &= 6k + 6k'\end{aligned}$$

$$(N+1)^3 - (N+1) = 6k''$$

Ejemplo Correctitud: Bubble Sort

Bubble Sort, es un algoritmo que ordena un arreglo realizando intercambios entre pares vecinos desordenados hasta que el arreglo esté ordenado.

1. Se compara una posición con la siguiente, y si el número siguiente es menor, entonces se intercambian. Se repite esta acción desde la primera hasta la penúltima posición.

2. Se repite el paso 1. hasta que se realice una iteración sin ningún intercambio.

Ejemplo Correctitud: Bubble Sort

1° Iteración

2	1	8	6

1	2	8	6

1	2	8	6

1	2	6	8

Ejemplo Correctitud: Bubble Sort

2° Iteración

1	2	6	8

1	2	6	8

1	2	6	8

1	2	6	8

Ejemplo Correctitud: Bubble Sort

```
1   void bubbleSort(int* lista, int largo)
2   {
3       int swapped = 1;
4
5       // Mientras se produzcan intercambios
6       while(swapped)
7       {
8           swapped = 0;
9           for(int i = 0; i < largo - 1; i++)
10          {
11              // Si el siguiente es menor,
12              // se intercambian
13              if (lista[i] > lista[i+1])
14              {
15                  swap(&lista[i], &lista[i+1]);
16                  swapped = 1;
17              }
18          }
19      }
20  }
```

Ejemplo Correctitud: Bubble Sort

Inducción:

1. **BI:** Como base de inducción se elegirá a un arreglo de largo 1. Como tiene un solo elemento, está ordenado.

2. **HI:** Bubble Sort ordena todo arreglo de largo **n**.

3. **TI:** Bubble Sort ordena todo arreglo de largo **n + 1**.

La tesis se demuestra utilizando la hipótesis.

Ejemplo Correctitud: Bubble Sort

TI: Se demuestra que Bubble Sort es correcto para todo arreglo de largo **n + 1**, utilizando la hipótesis.

Se observa que en Bubble sort, luego de la primera iteración de intercambios, el máximo del arreglo quedará en la última posición, y se mantendrá en esa posición, ya que, al ser el mayor, nunca se cumplirá la condición de intercambio con otro número en la posición anterior.

Por lo que, siendo **A** un arreglo de largo **n + 1**, **max(A)** el máximo del arreglo **A**, y **A'** el mismo arreglo pero sacando a **max(A)**. Se tiene que:

BS(A) = BS(A') + [max(A)]

Luego, se observa que **A'** será de largo **n**, porque se removió el máximo, entonces por **HI** se deduce que **BS(A')** entregará un arreglo ordenado. Y como el agregarle **max(A)** al final del arreglo mantendrá el orden correcto, se demuestra que **BS(A)** entrega un arreglo ordenado.

Ejemplo Correctitud: Bubble Sort

- Se demostró que Bubble Sort ordena arreglos de cualquier tamaño, pero.
  ¿Se demostró que Bubble Sort siempre termina?

- La condición de término de Bubble Sort es que no se produzcan intercambios en una iteración, y esto ocurre si y solo si el arreglo está ordenado.
  Como para cualquier input válido se consiguen arreglos ordenados, Bubble Sort siempre termina. Y como los inputs válidos siempre serán arreglos finitos, se termina en tiempo finito.