

# METODO DE ORDENAMIENTO

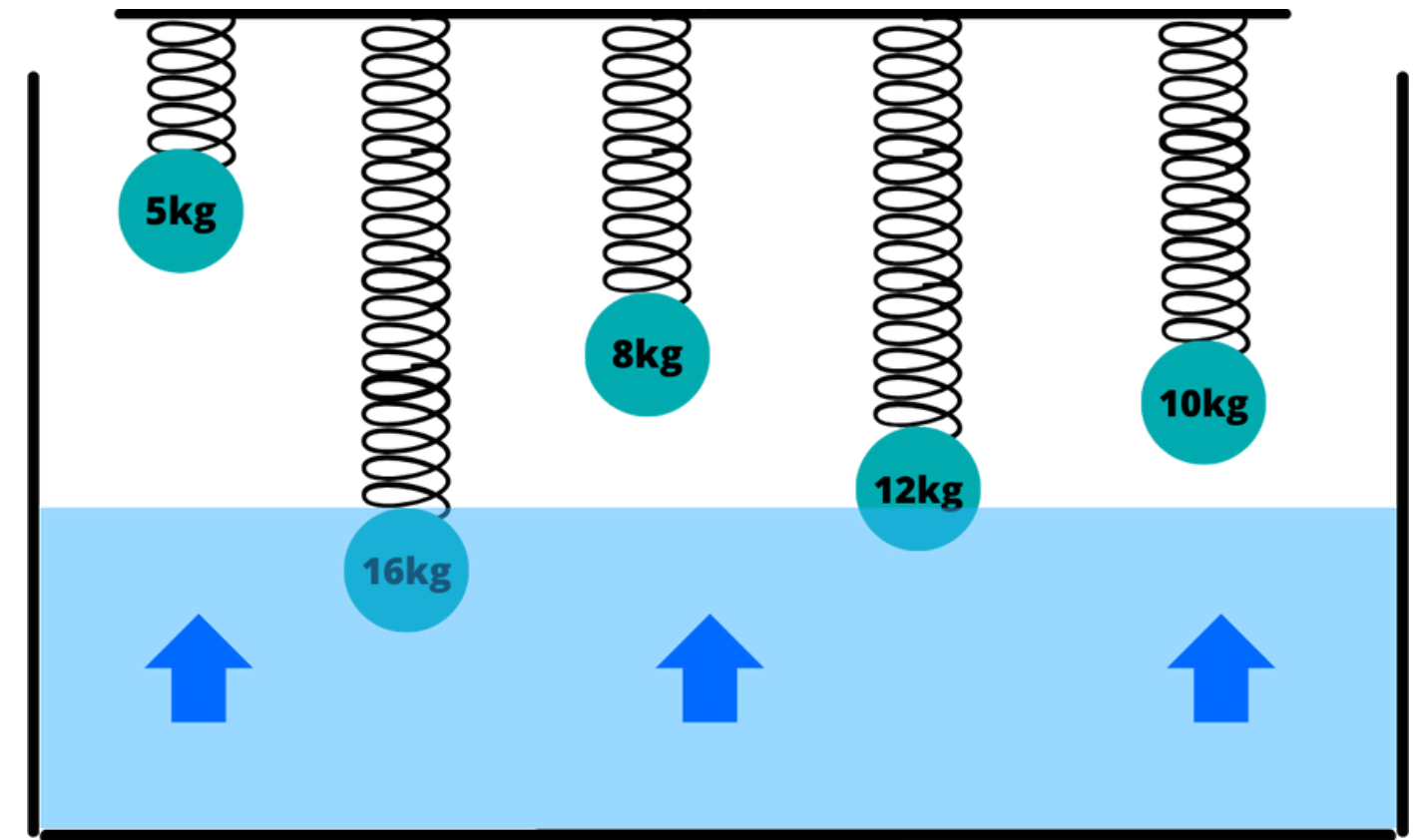
Algoritmo de ordenamiento, diseñado por Kevin Tupac Agüero, basándose en un modelo de la vida real.



# COMO SURGIÓ LA IDEA

Estaba en la clase de Estructura de Datos cuando mi profesor nos mencionó que existían diversos algoritmos de ordenamiento. Nos propuso un reto interesante: si alguno de nosotros creaba un algoritmo original, él lo enseñaría en clase y llevaría nuestro nombre. Desde ese momento, no pude dejar de pensar en ideas para ordenar objetos.

Pensé en utilizar algo de la vida real como base para el ordenamiento. Imaginé una cuerda con resortes amarrados a ella, y de estos resortes colgaban bolas de distintos tamaños. Si agregamos agua, esta irá subiendo y chocará primero con las bolas de mayor peso, y luego con las de menor peso, hasta llegar al tope. Esa fue mi idea inicial, y ahora necesito convertirla en código.



# PROCESO DE CREACIÓN



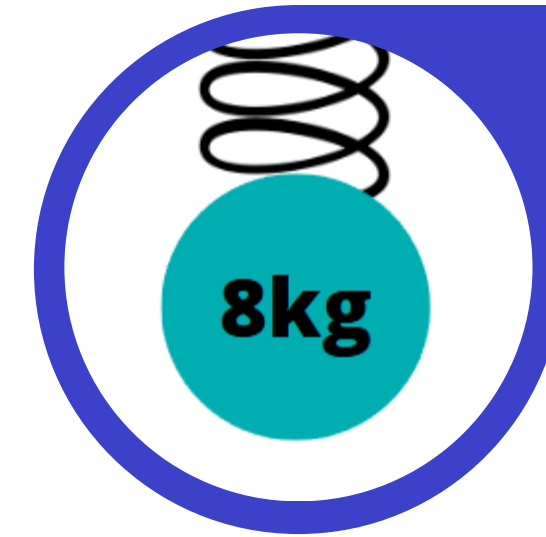
## CONVERTIR LOS PESOS

Mi primer desafío fue encontrar una manera de representar los pesos de las bolas de una manera que el código pudiera entender. Decidí convertir los pesos a bits. En términos binarios, los números con mayor valor tienen una representación más larga, lo que permite comparaciones directas.



## SIMULACION Y MEJORA

Realicé una simulación para visualizar cómo quería que el algoritmo funcionara. Realice un paso a paso de lo que sucedería y como podría mejorarse durante ese proceso.



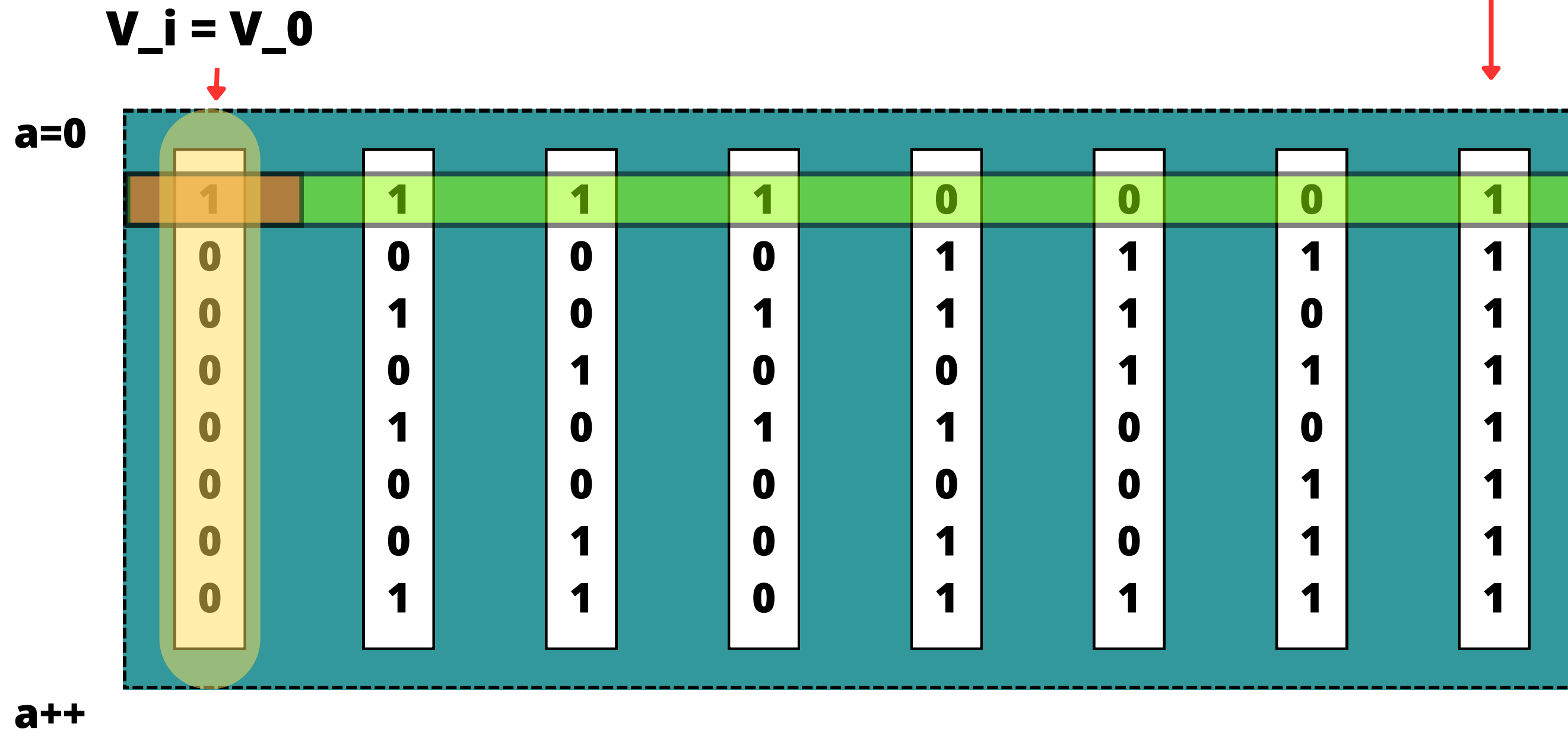
## REALIZACIÓN DEL ALGORITMO

Una vez que tuve claro lo que quería, pasé a realizar el algoritmo mediante ensayo y error, logrando terminarlo. Comprobé que su complejidad temporal era  $O(n \log n)$ , lo cual es eficiente para algoritmos de ordenamiento.

Si se encuentra un 1:  $a++$ ,  $a = 0$

$V_a = V_0$

$V_{n-1}$



Si se encuentra un 1:  $a++$ ,  $a = 0$

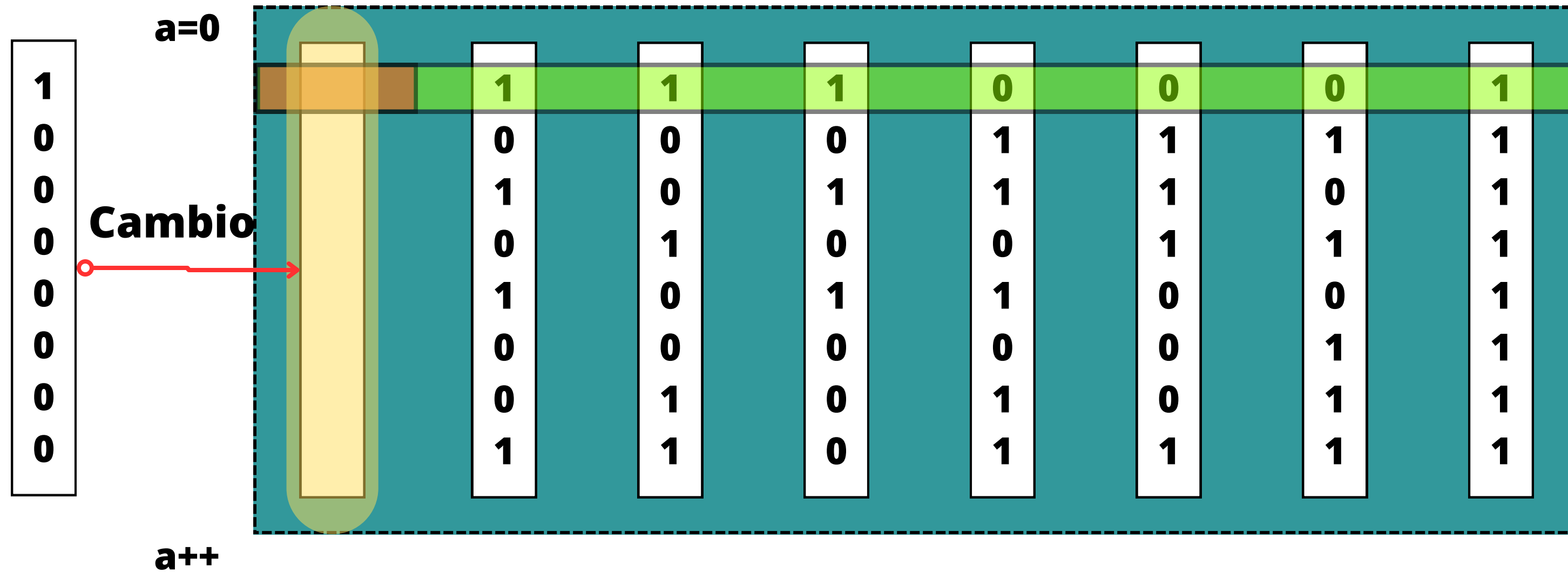
FuncionCambio( $V_i$ ,  $V_a$ ) {

$aux = V_i$

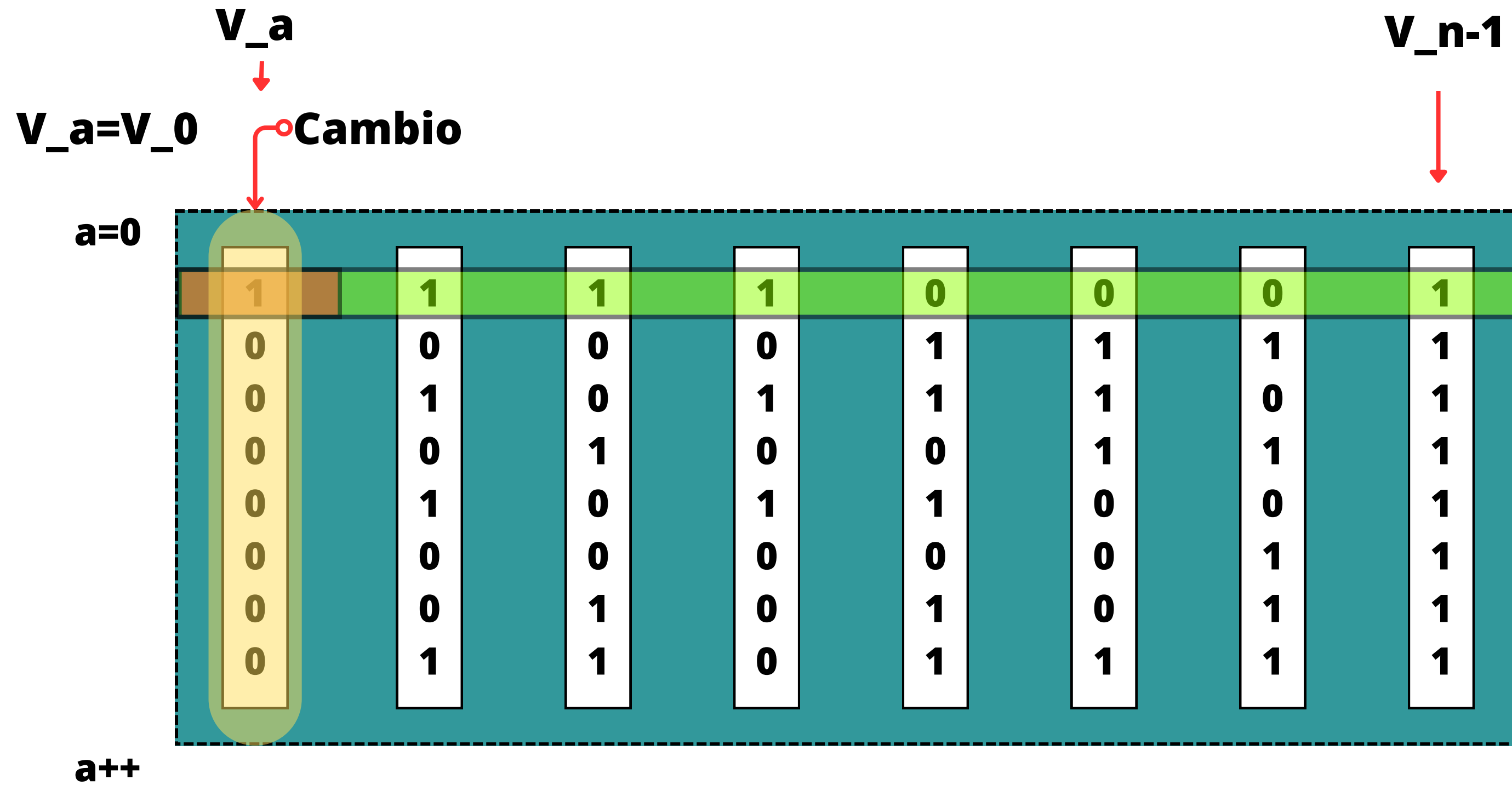
$V_i = V_a$

$V_a = aux$

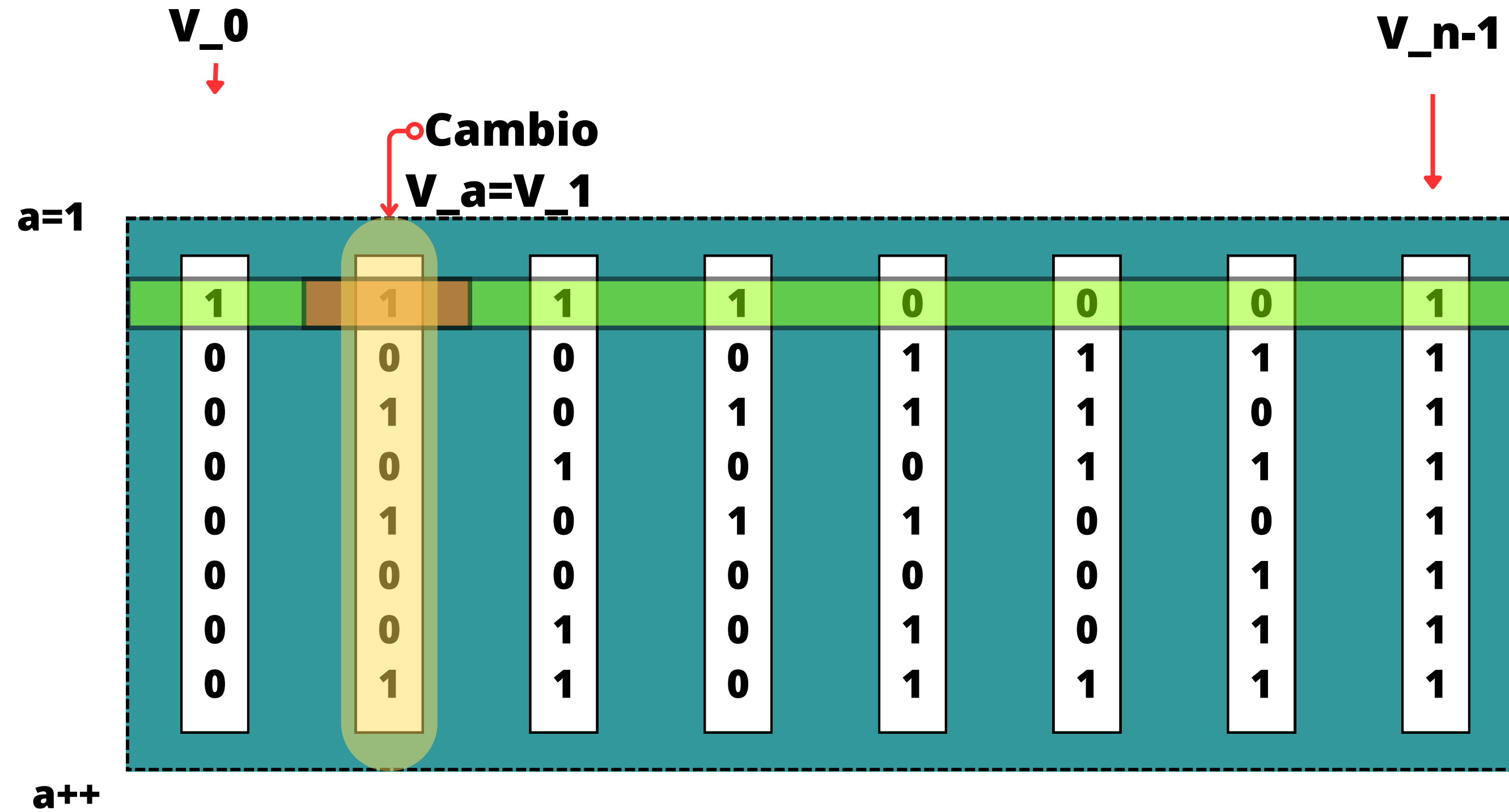
$V_{n-1}$



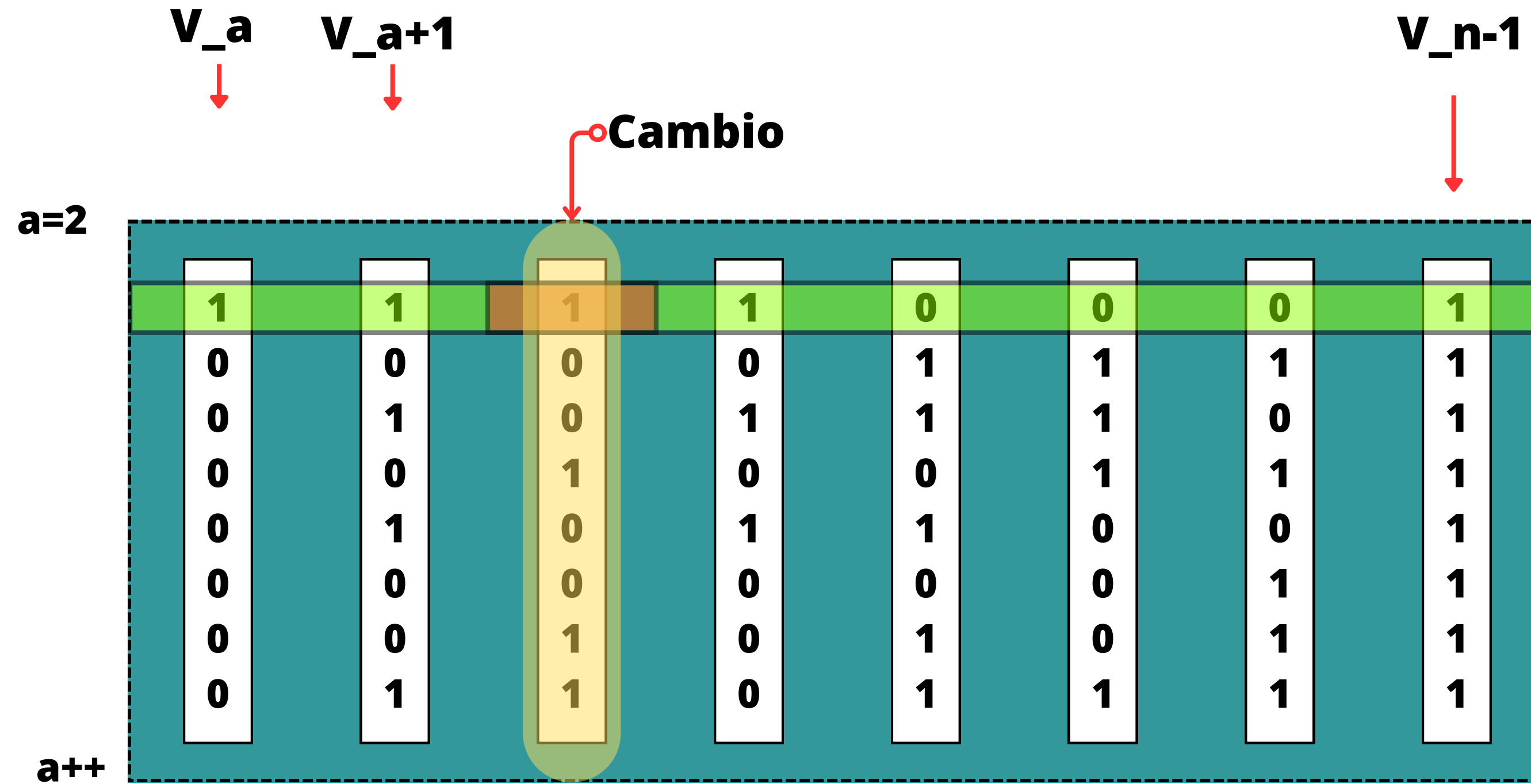
**Si se encuentra un 1: a++, a = 0**



Si se encuentra un 1:  $a++$ ,  $a = 0$

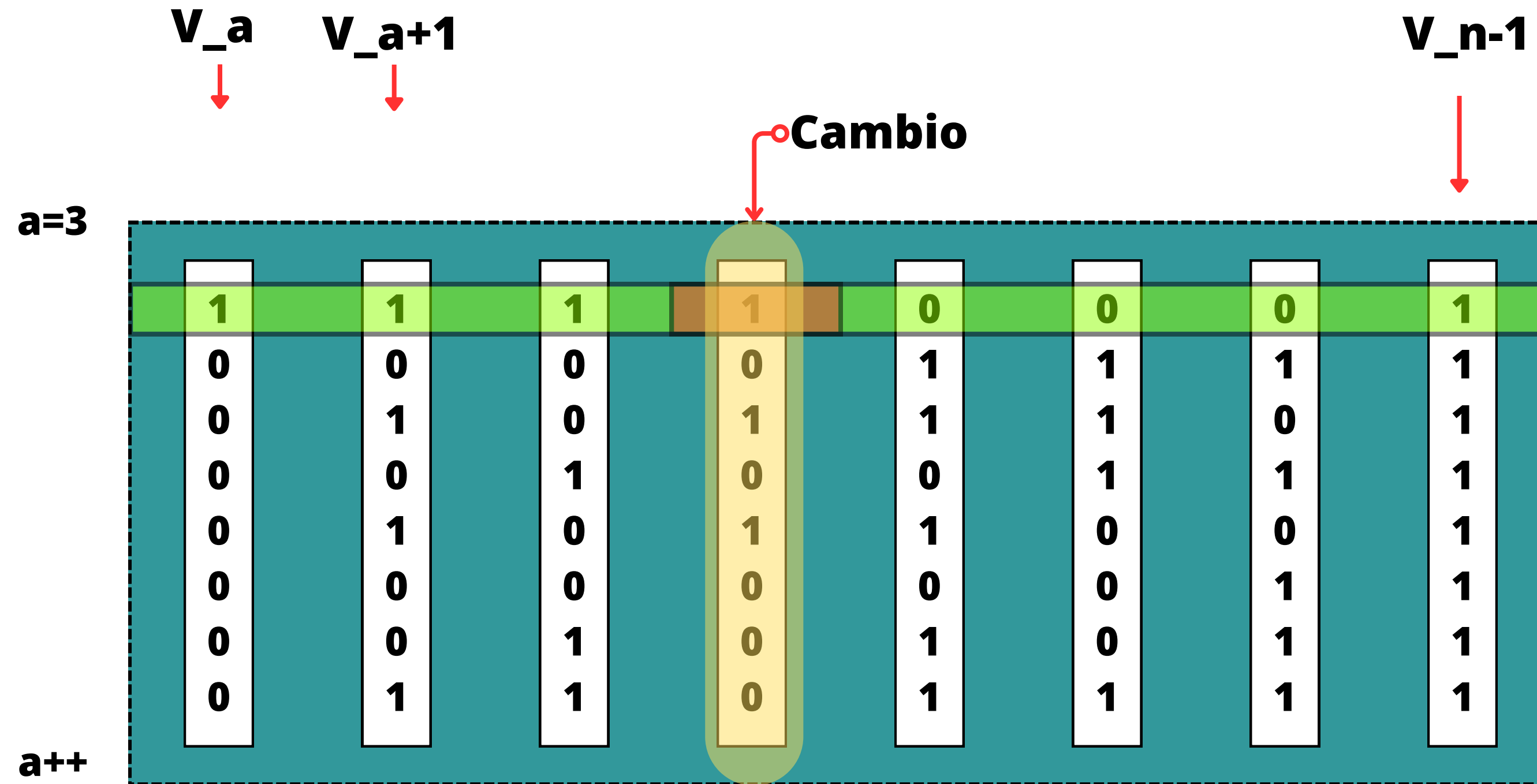


Si se encuentra un 1:  $a++$ ,  $a = 0$





## Si se encuentra un 1: $a++$ , $a = 0$



## Si se encuentra un 1: $a++$ , $a = 0$

**V\_a**



**V\_a+1**



**V<sub>n-1</sub>**




**a=4**

1	1	1	1	0	0	0	1
0	0	0	0	1	1	1	1
0	1	0	1	1	1	0	1
0	0	1	0	0	1	1	1
0	1	0	1	1	0	0	1
0	0	0	0	0	0	1	1
0	0	1	0	1	0	1	1
0	1	1	0	1	1	1	1


# a++

## Si se encuentra un 1: $a++$ , $a = 0$


**V\_a**



**V<sub>a</sub>+1**

A red arrow pointing downwards from the text 'V<sub>a</sub>+1' to the text 'V<sub>a</sub> = 1000' in the next block.

**V<sub>n-1</sub>**

A red arrow pointing downwards from the text 'V<sub>n-1</sub>' to the text 'V<sub>n</sub>'.


**a=4**

1	1	1	1	0	0	0	1
0	0	0	0	1	1	1	1
0	1	0	1	1	1	0	1
0	0	1	0	0	0	1	1
0	1	0	1	1	0	0	1
0	0	0	0	0	0	1	1
0	0	1	0	1	0	1	1

# a++


## Si se encuentra un 1: $a++$ , $a = 0$

**V\_a**



**V<sub>a</sub>+1**

**V<sub>n-1</sub>**




**a=4**

1	1	1	1	0	0	0	1
0	0	0	0	1	1	1	1
0	1	0	1	1	1	0	1
0	0	1	0	0	1	0	1
0	1	0	1	1	0	1	1
0	0	0	0	0	0	1	1
0	0	1	0	1	0	1	1
0	1	1	0	1	1	1	1

# a++

## Si se encuentra un 1: $a++$ , $a = 0$

**V\_a**



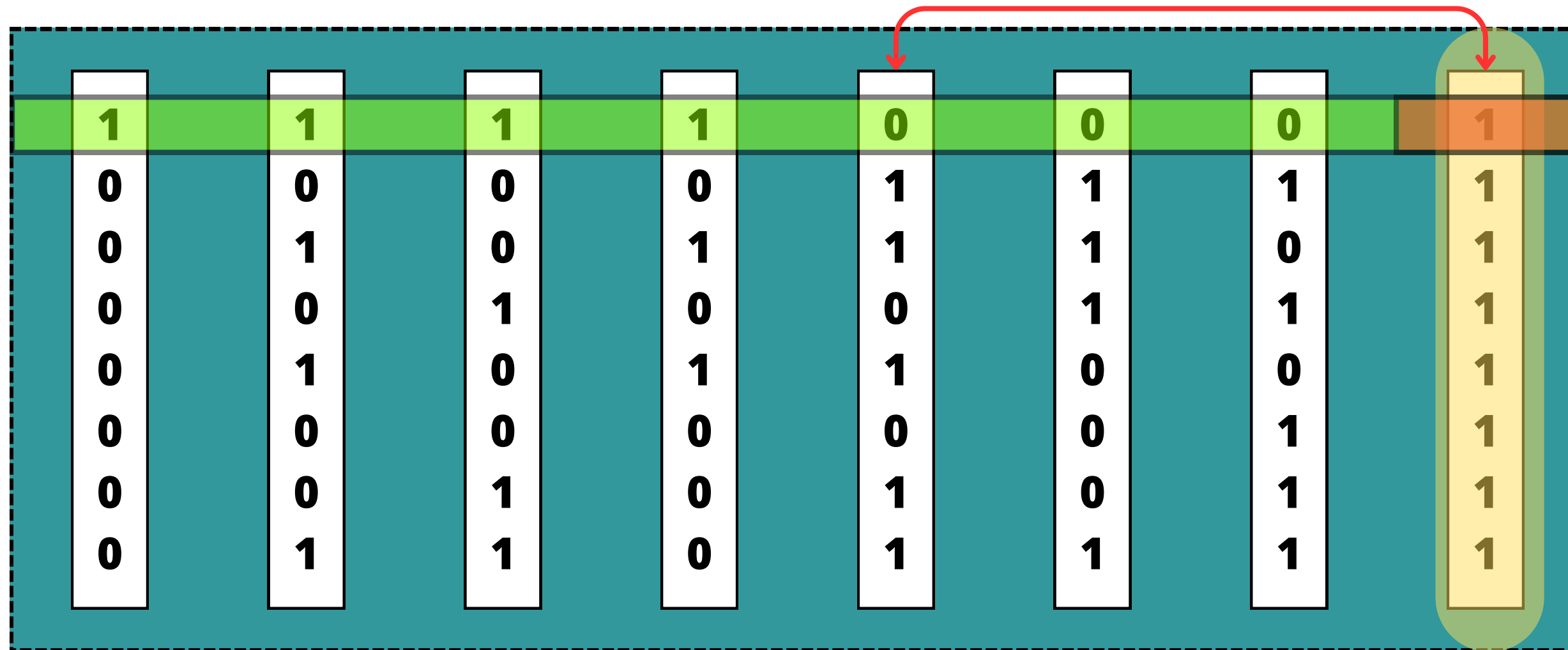
**V<sub>a</sub>+1**

**V<sub>n-1</sub>**

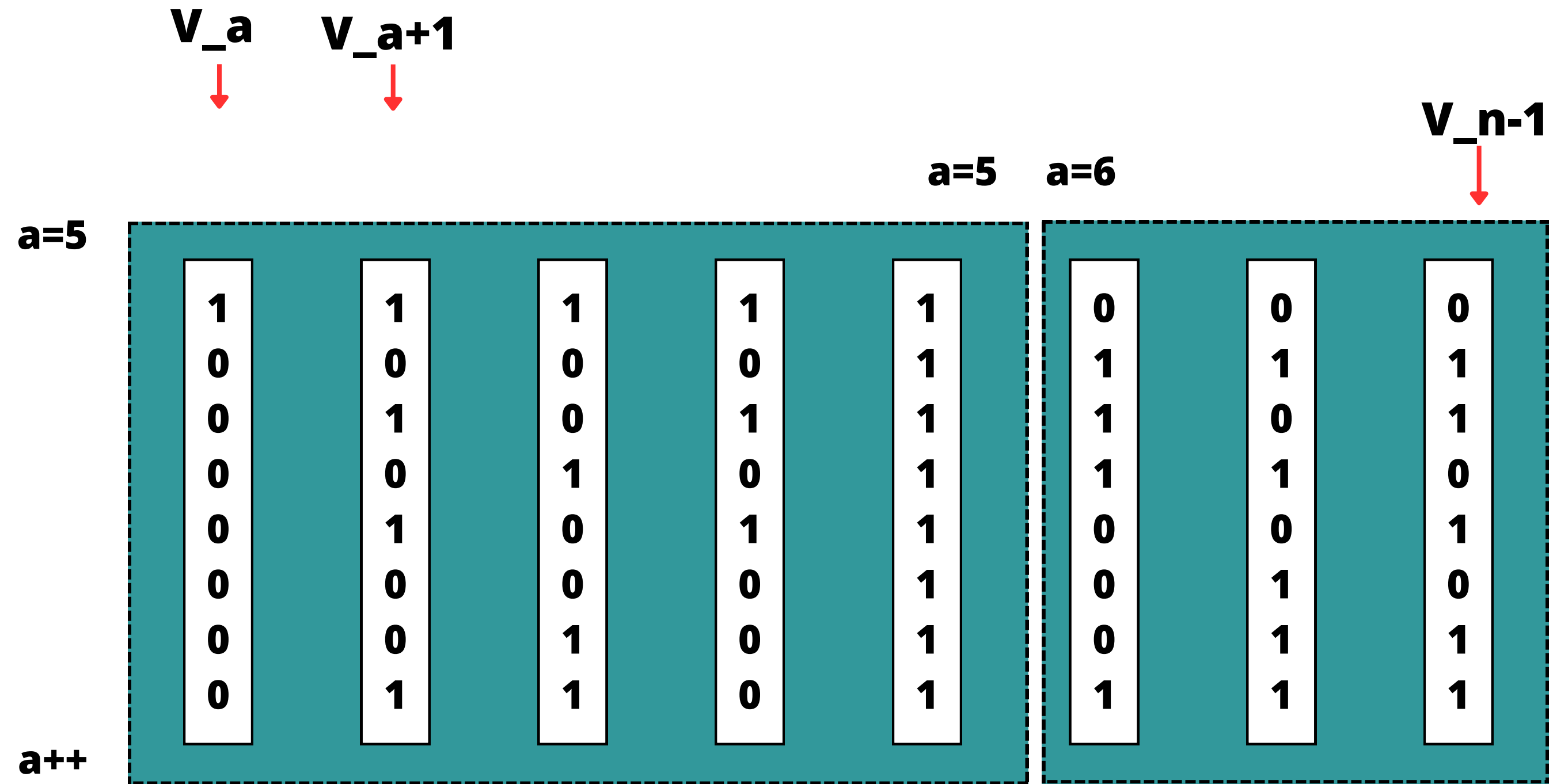
# Cambio

**a=4**

# a++



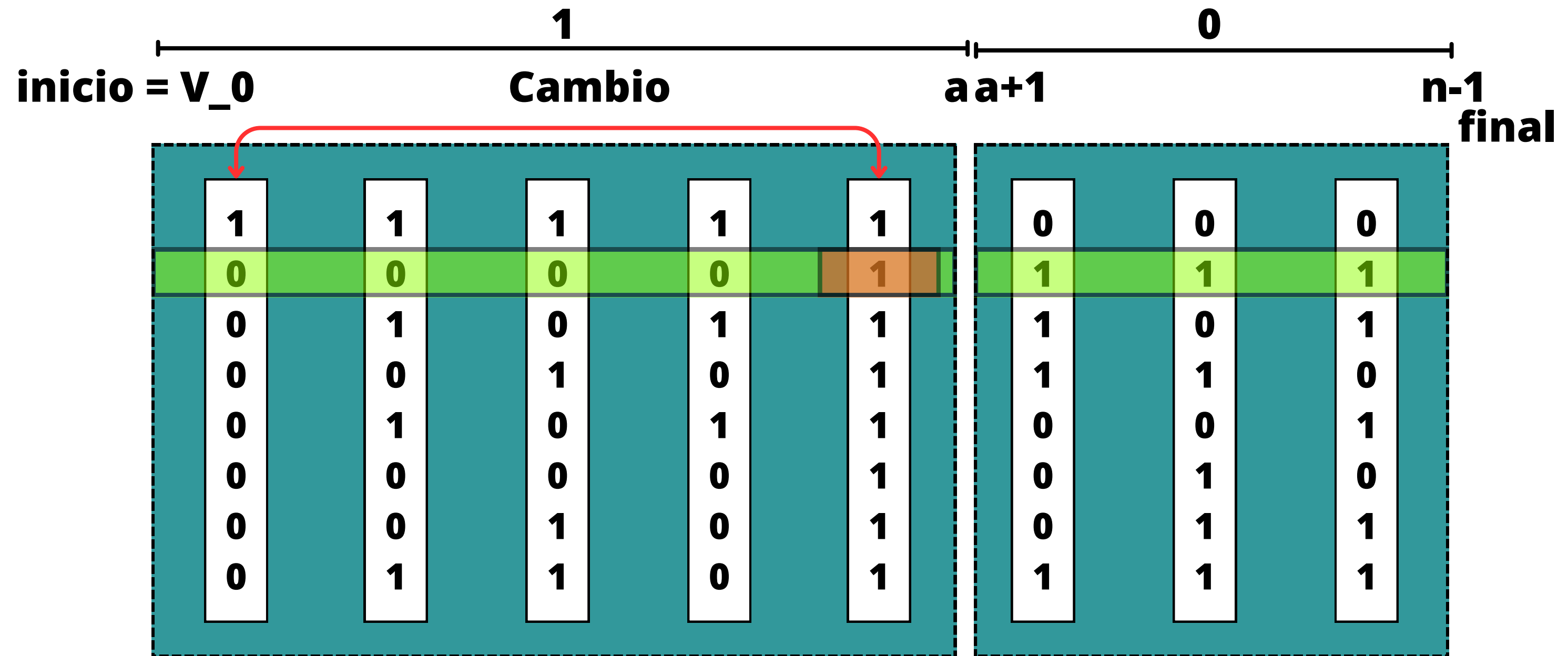
**Si se encuentra un 1: a++, a = 0**



The diagram illustrates a 2D array structure. A horizontal line at the top indicates a partition at index  $aa+1$ . The left part of the array is labeled **1** and the right part is labeled **0**. The array is bounded by **inicio = V\_0** and **n-1 final**. A row is highlighted in green, showing a transition from 1 to 0 at the partition. A column is highlighted in yellow, showing a transition from 0 to 1 at the partition. The array is represented as a grid of cells, with the highlighted row and column intersecting at the partition point.

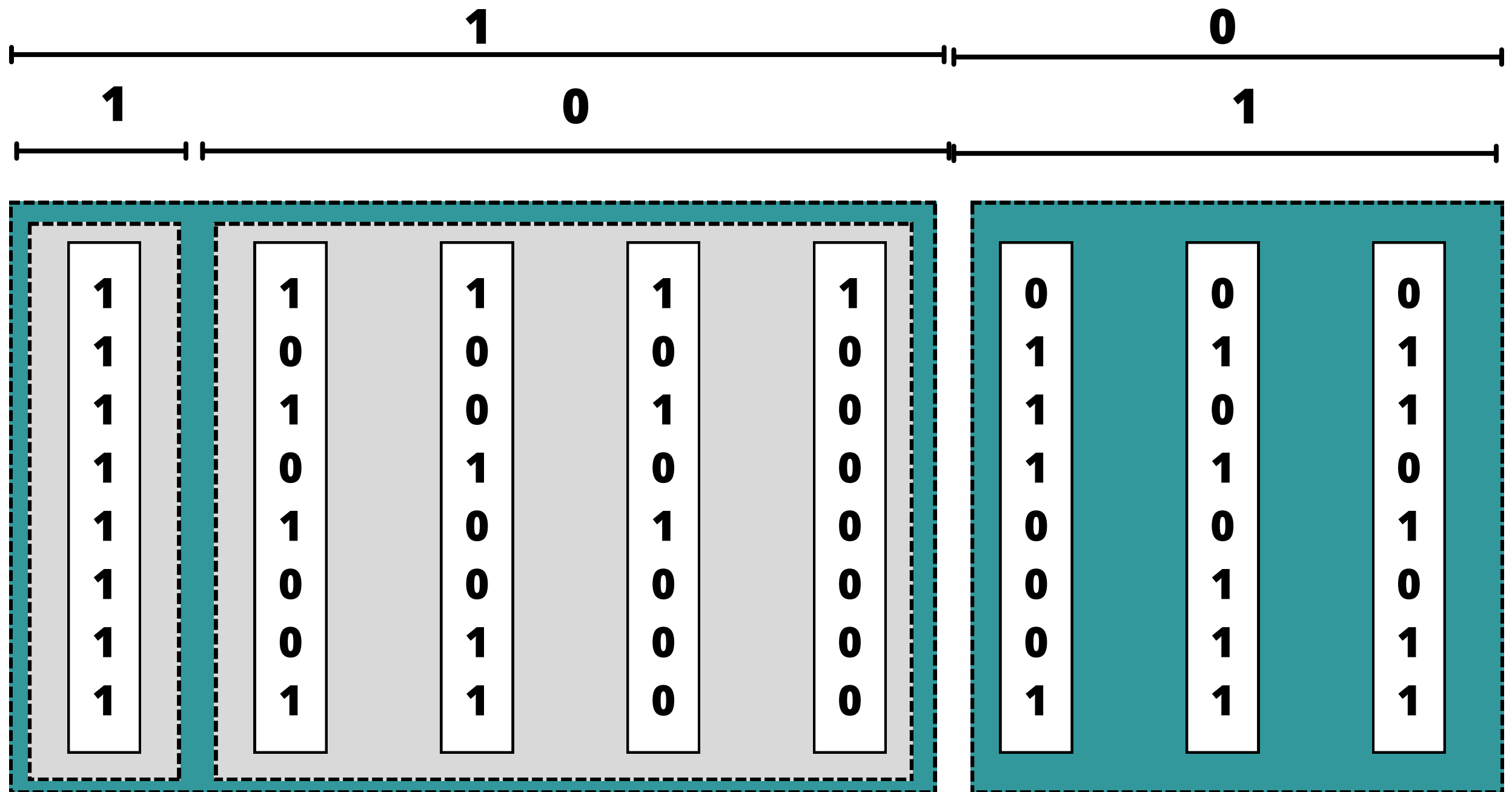
1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1
0	1	0	1	1	1	0	1
0	0	1	0	1	0	1	0
0	0	0	0	1	0	1	1
0	0	1	0	1	0	1	1
0	1	1	0	1	1	1	1

## 2 grupos

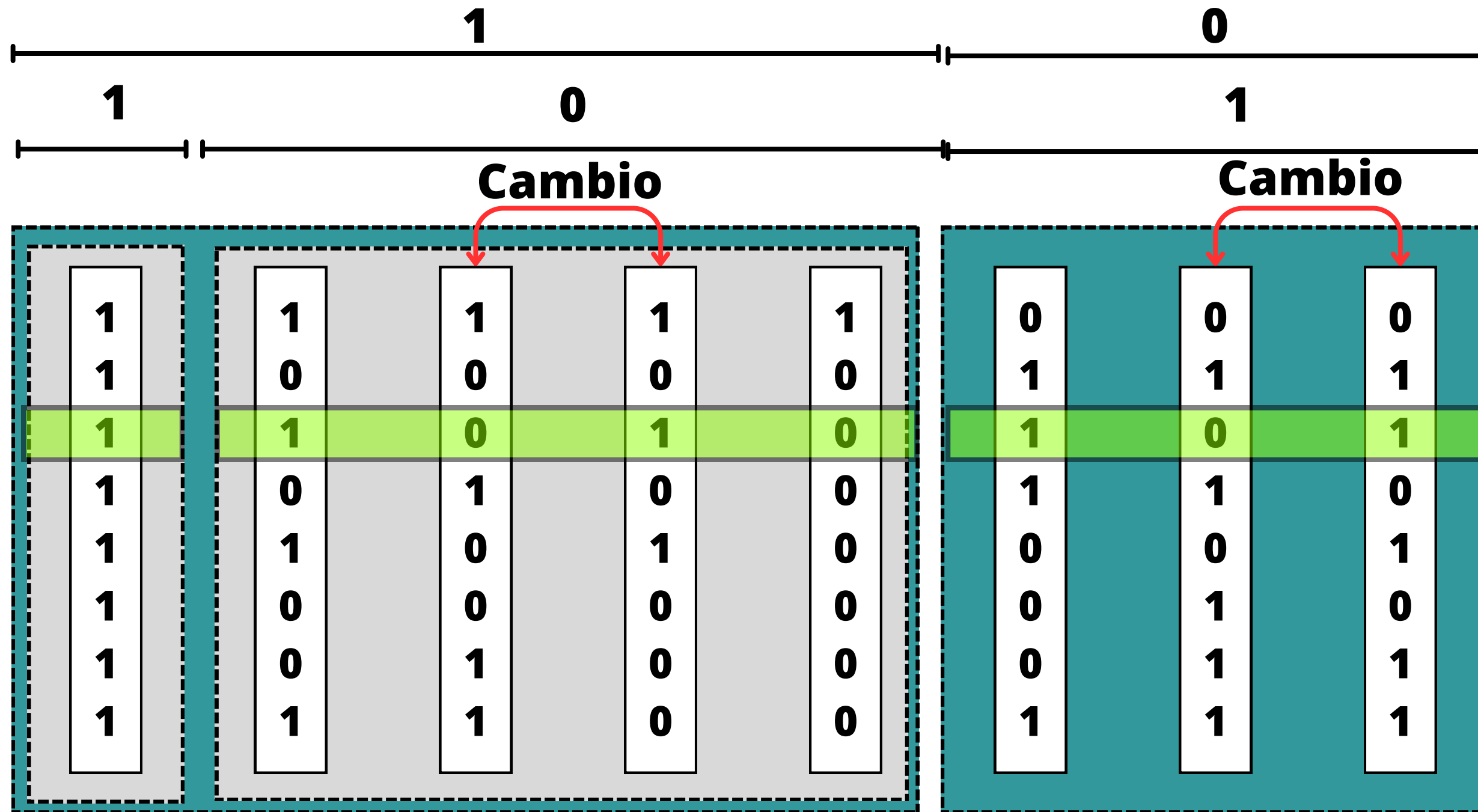


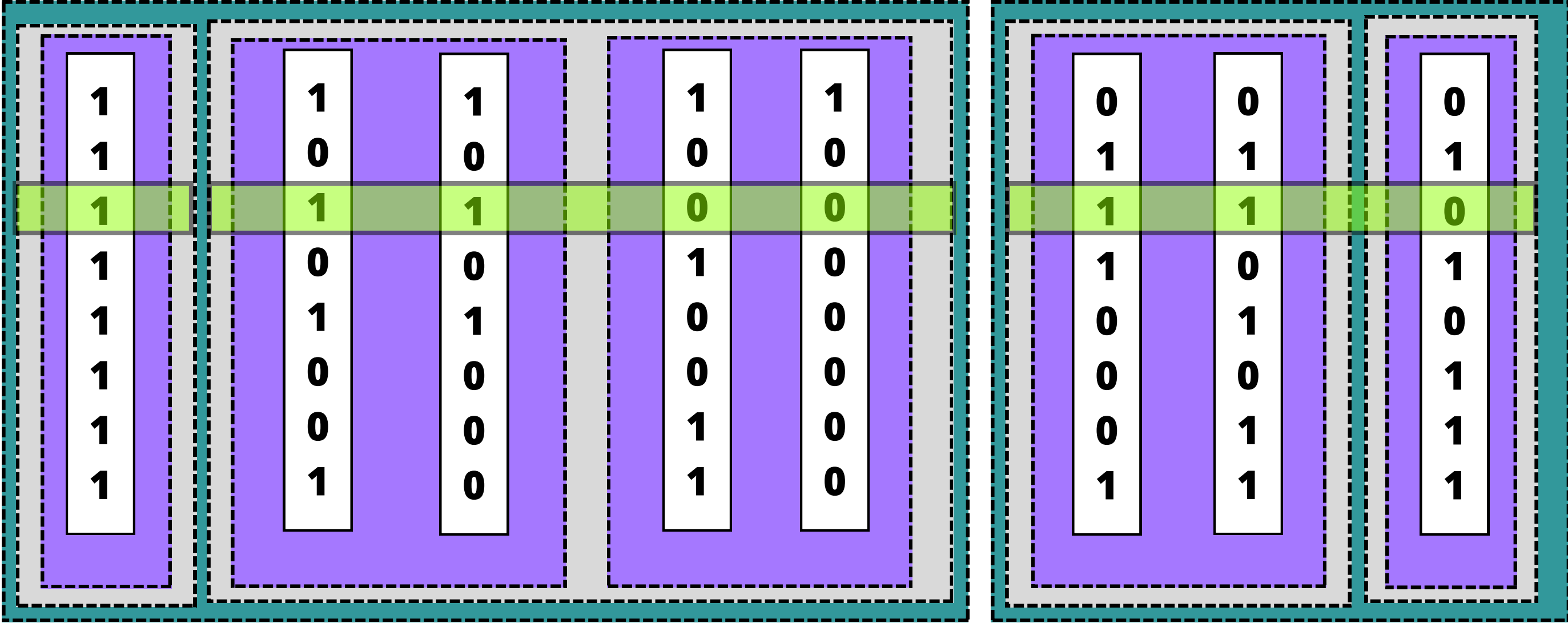


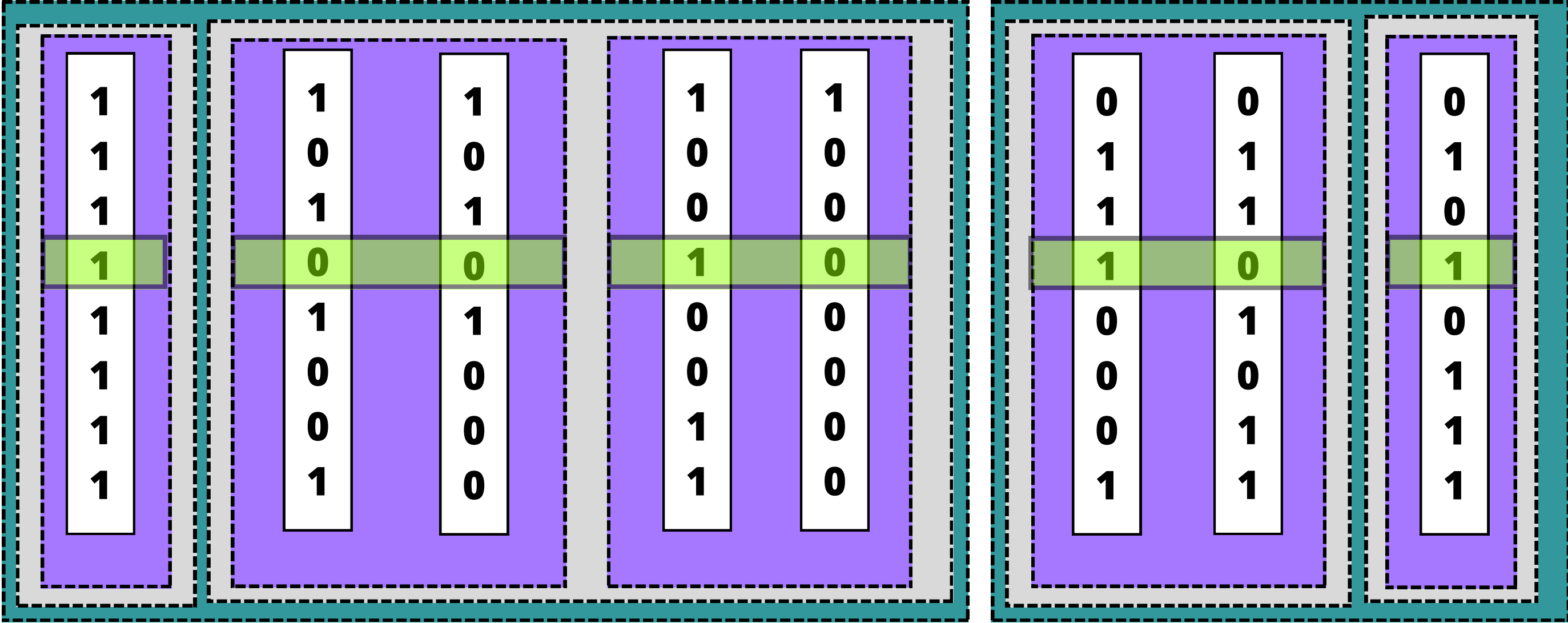
**3 grupos**



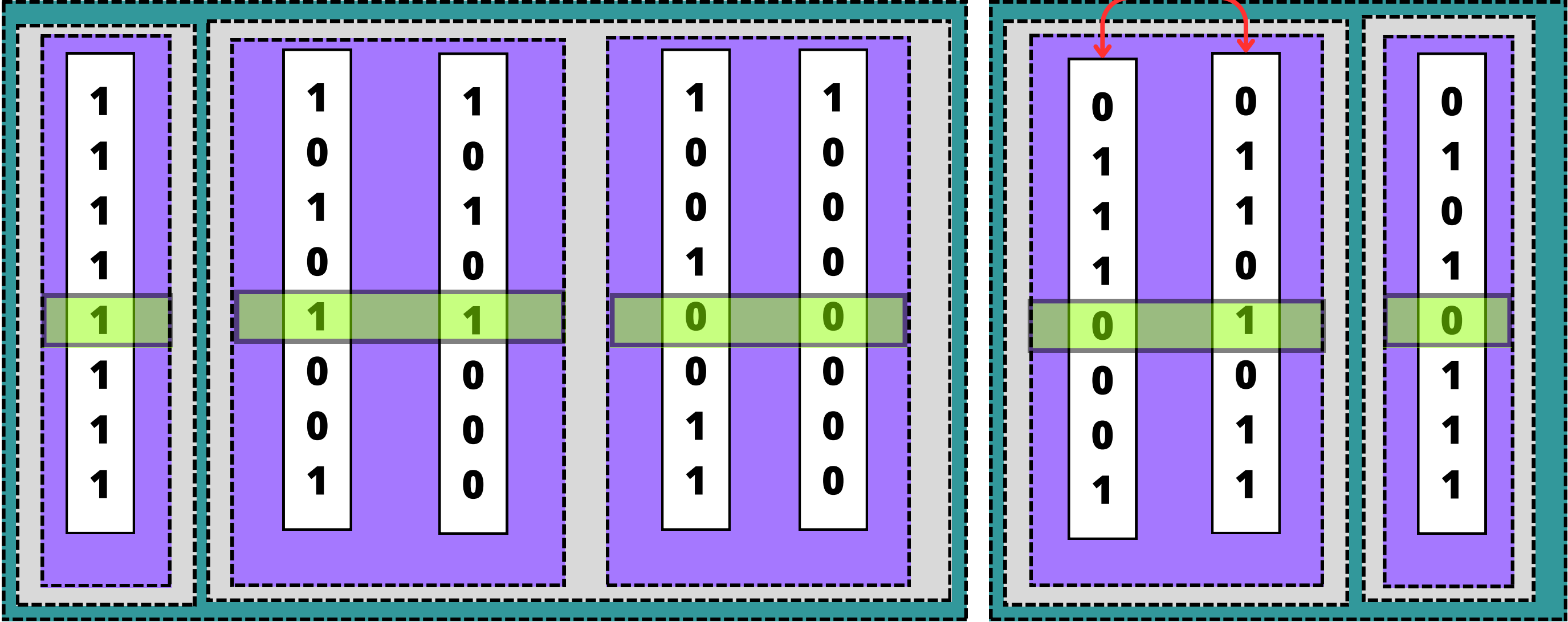
## 3 grupos

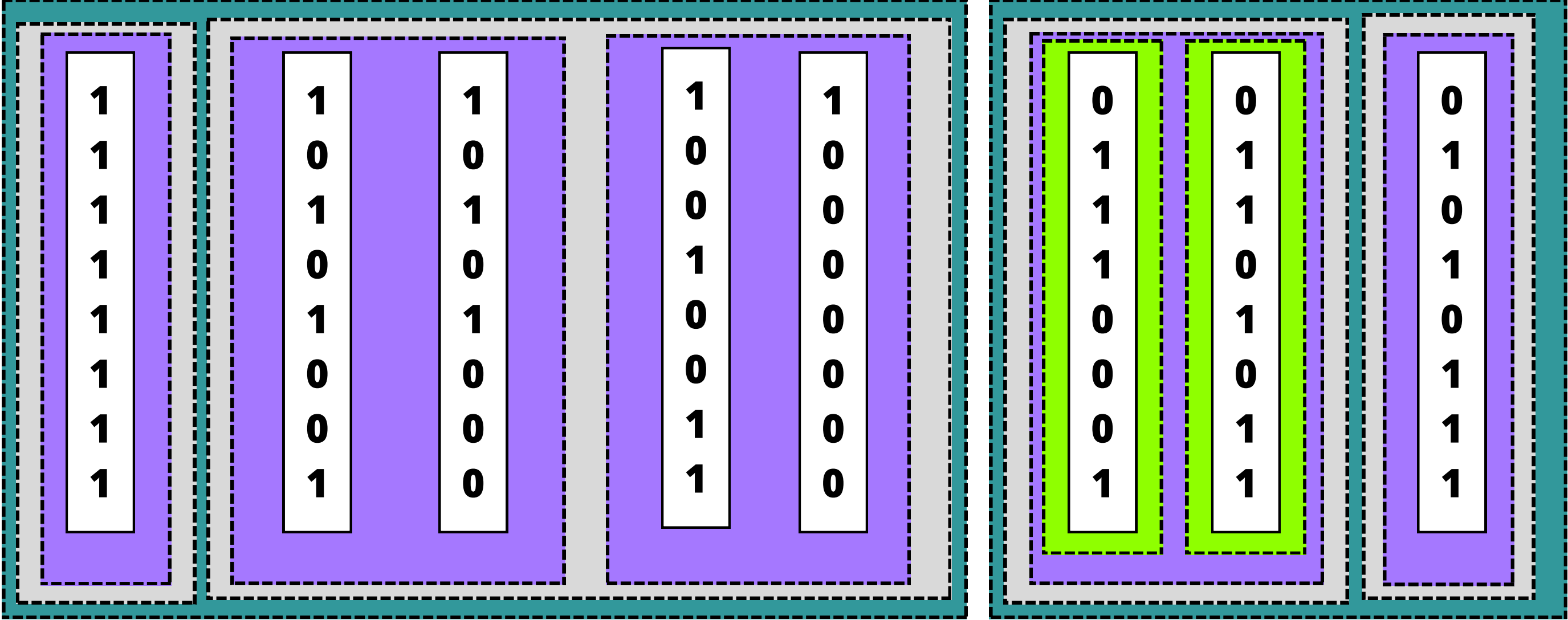


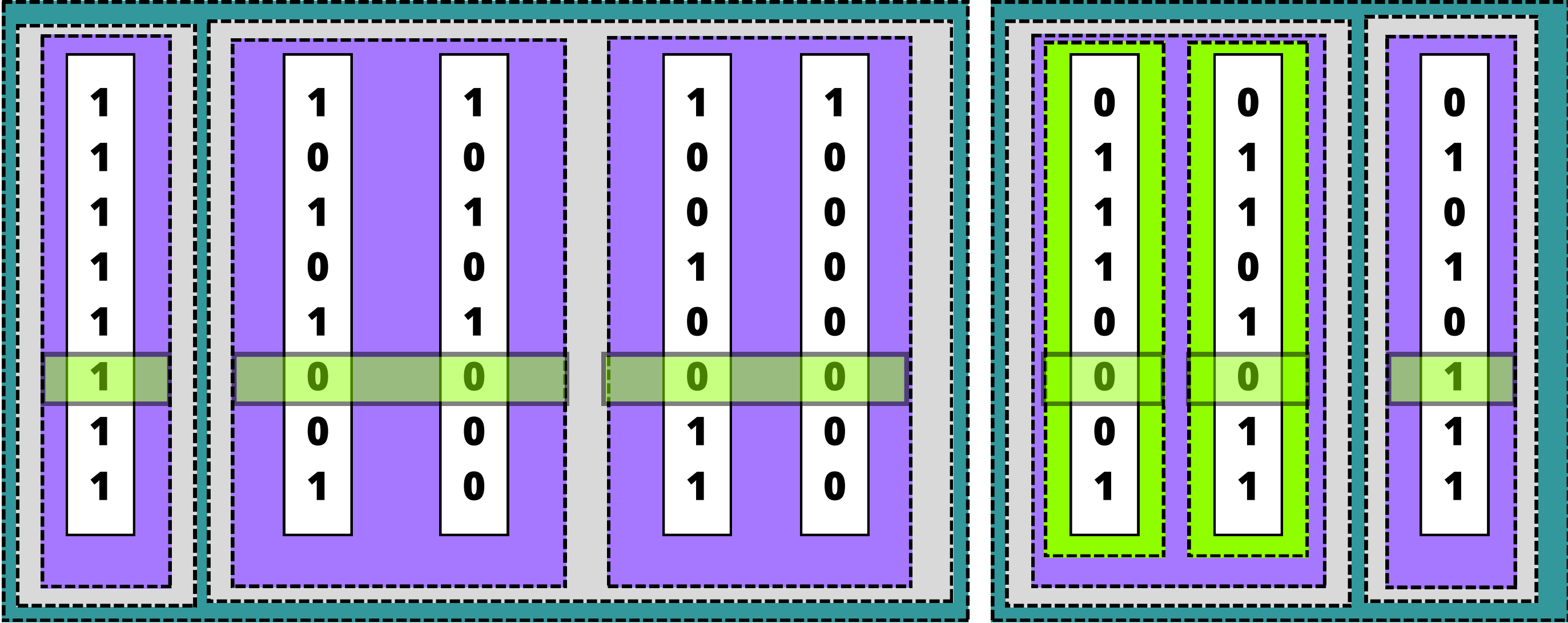


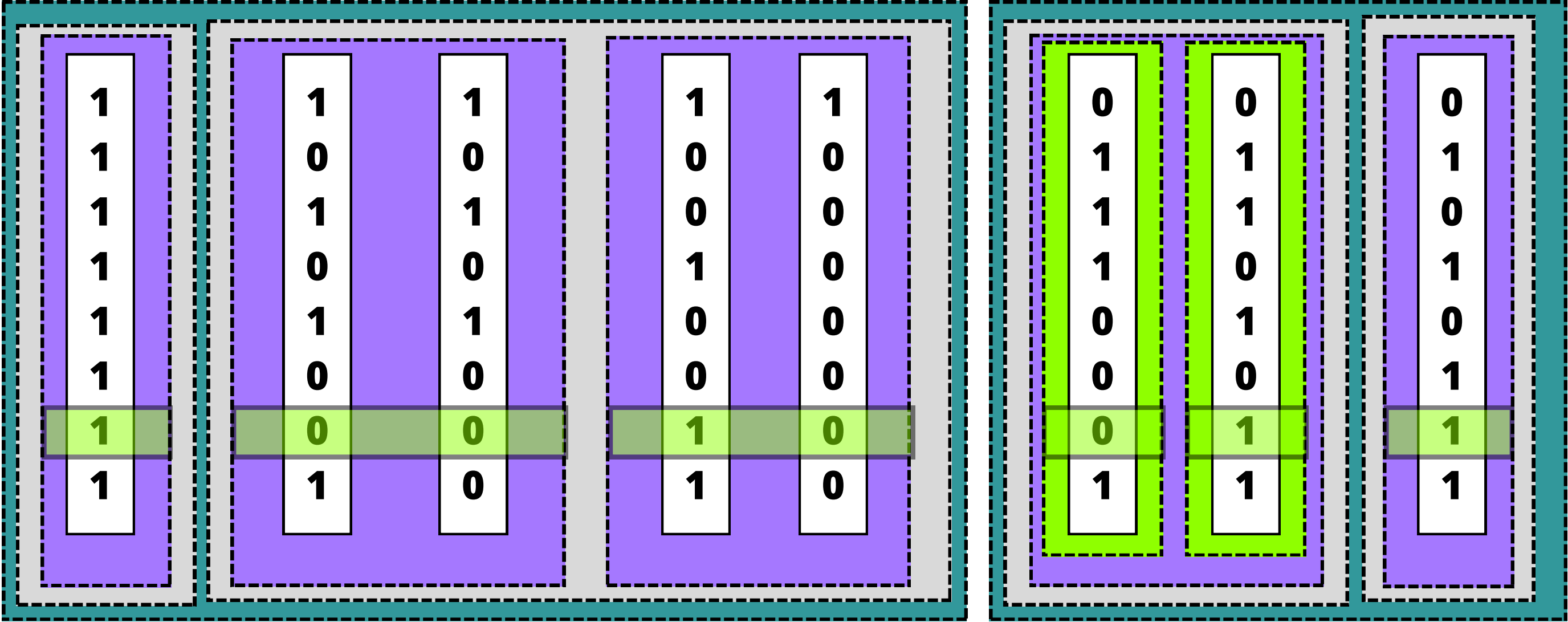


Cambio





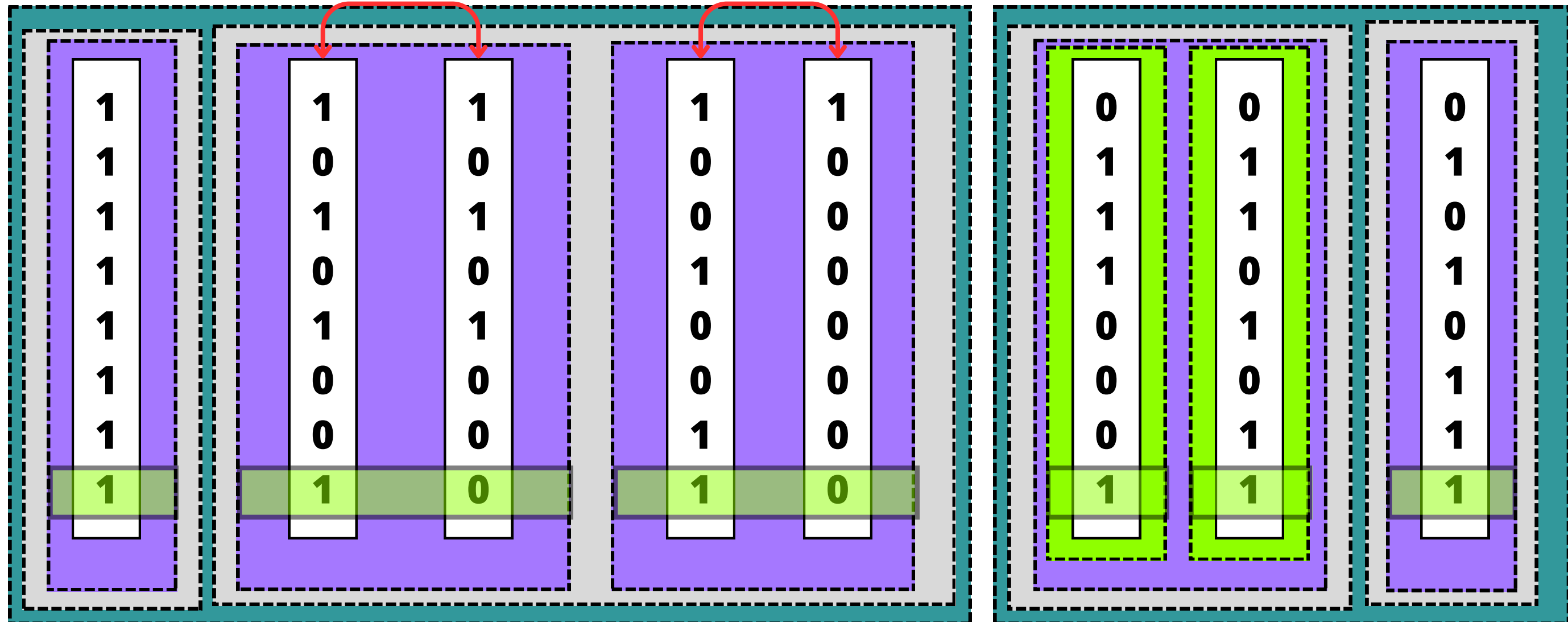


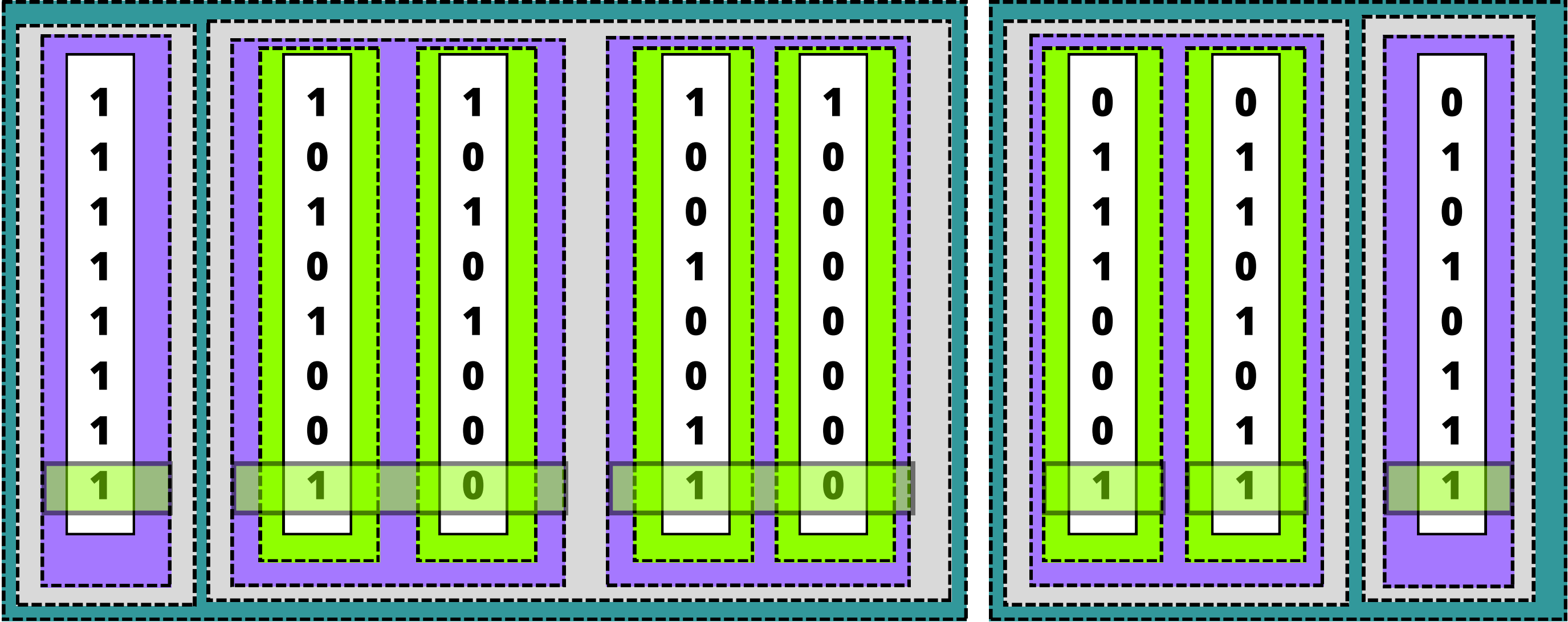




**Cambio**

**Cambio**





n

32

