

Tri par insertion

```
tri_insertion( tableau[ ] ){  
    taille = tableau.length  
    pour i de 1 à taille {  
        index = tableau[i]  
        j = i-1  
        tantque j >= 0 et tableau[j] > index {  
            tableau[j+1] = tableau[j]  
            j--  
        }  
        tableau[j+1] = index  
    }  
}
```

Tri par insertion

Tableau = [13;2;5;15]

```
tri_insertion( tableau[ ] ){  
    taille = tableau.length  
    pour i de 1 à taille {  
        index = tableau[i]  
        j = i-1  
        tantque j >=0 et tableau[j] > index {  
            tableau[j+1] = tableau[j]  
            j--  
        }  
        tableau[j+1] = index  
    }  
}
```

Tri par insertion

Tableau = [13;2;5;15]

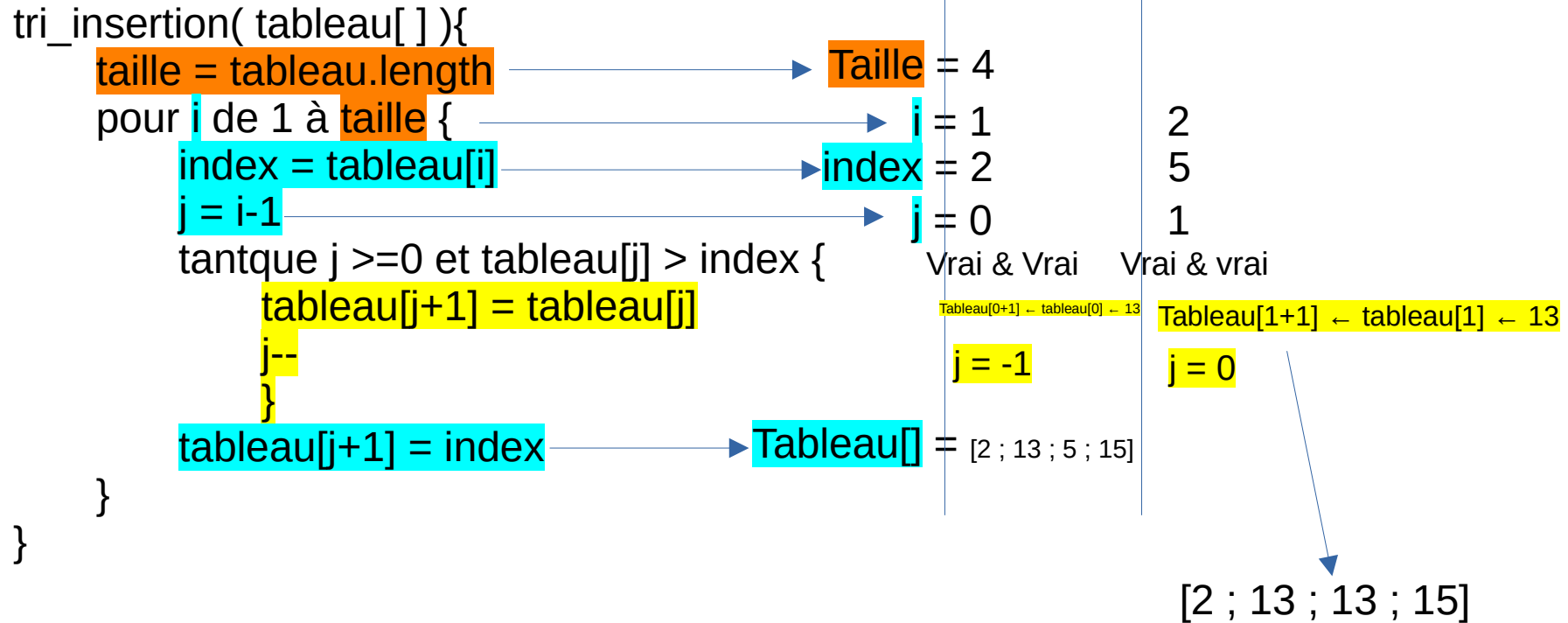
```
tri_insertion( tableau[ ] ){  
    taille = tableau.length  
    pour i de 1 à taille {  
        index = tableau[i]  
        j = i-1  
        tantque j >= 0 et tableau[j] > index {  
            tableau[j+1] = tableau[j]  
            j--  
        }  
        tableau[j+1] = index  
    }  
}
```

The diagram illustrates the state of the algorithm during the first iteration of the `tri_insertion` function. A vertical line separates the code from the variable values and array state.

- `taille = tableau.length` is mapped to `Taille = 4`.
- `pour i de 1 à taille {` is mapped to `i = 1`.
- `index = tableau[i]` is mapped to `index = 2`.
- `j = i-1` is mapped to `j = 0`.
- The loop condition `tantque j >= 0 et tableau[j] > index {` is evaluated as `Vrai & Vrai` (True & True).
- Inside the loop, `tableau[j+1] = tableau[j]` is shown as `Tableau[0+1] ← tableau[0] ← 13`.
- Then, `j--` is shown as `j = -1`.
- After the loop, `tableau[j+1] = index` is mapped to `Tableau[] = [2 ; 13 ; 5 ; 15]`.

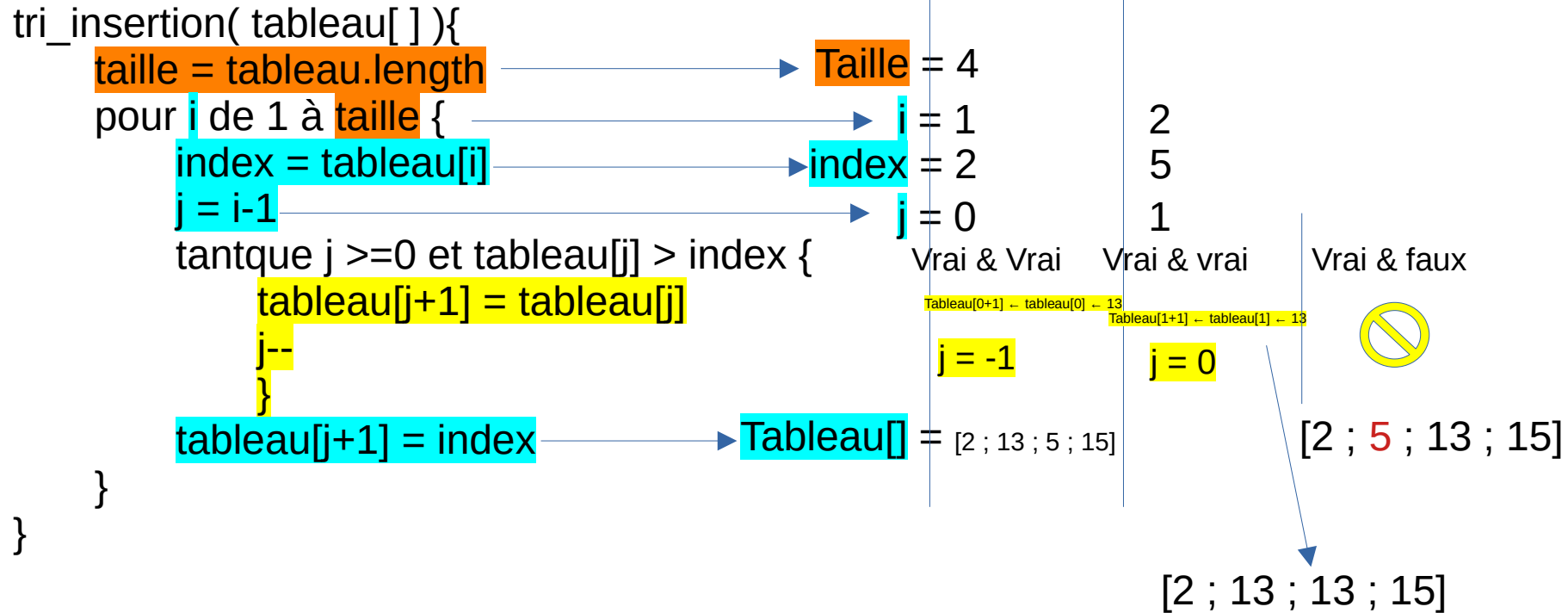
Tri par insertion

Tableau = [13;2;5;15]



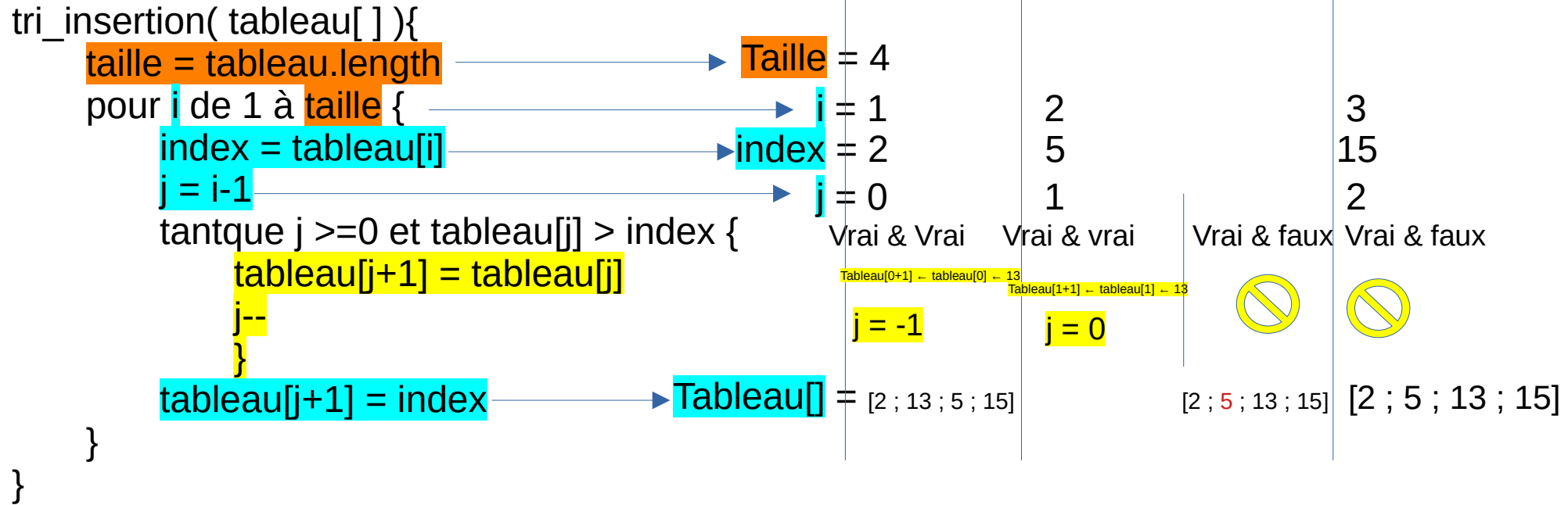
Tri par insertion

Tableau = [13;2;5;15]



Tri par insertion

Tableau = [13;2;5;15]



Tri par insertion

Tableau = [13;2;5;15]

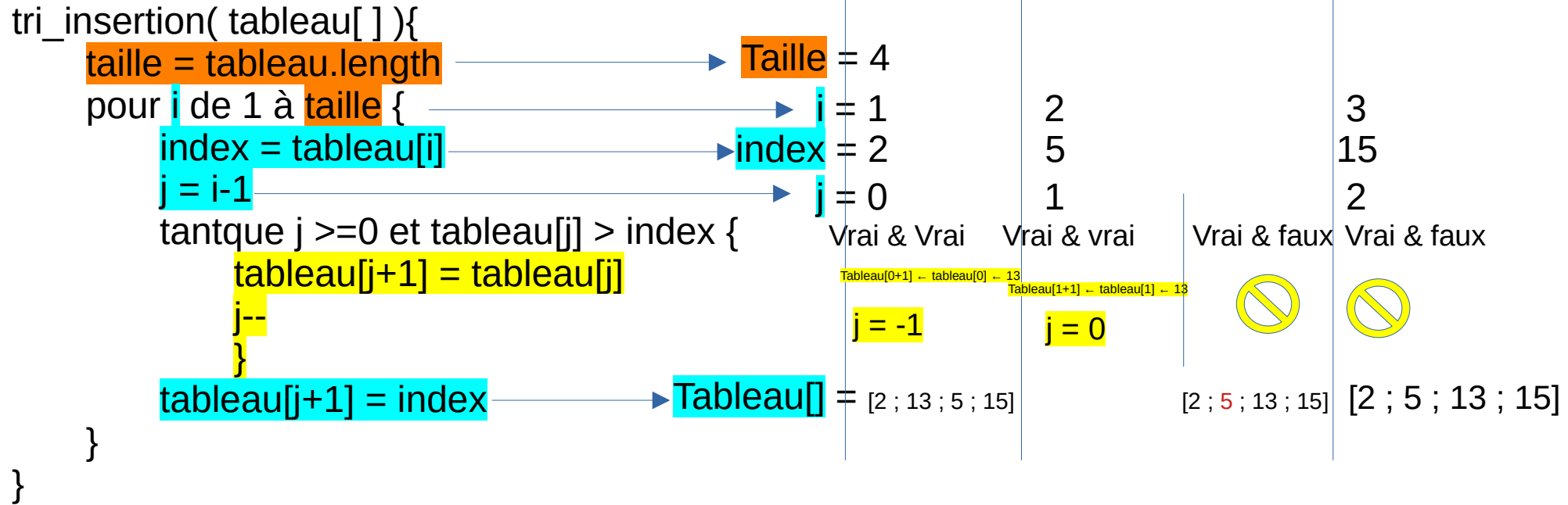


Tableau initial = [13;2;5;15] → Tableau final = [2 ; 5 ; 13 ; 15]