Esercitazione:
algoritmo per il **problema del partizionamento (*partition*)**

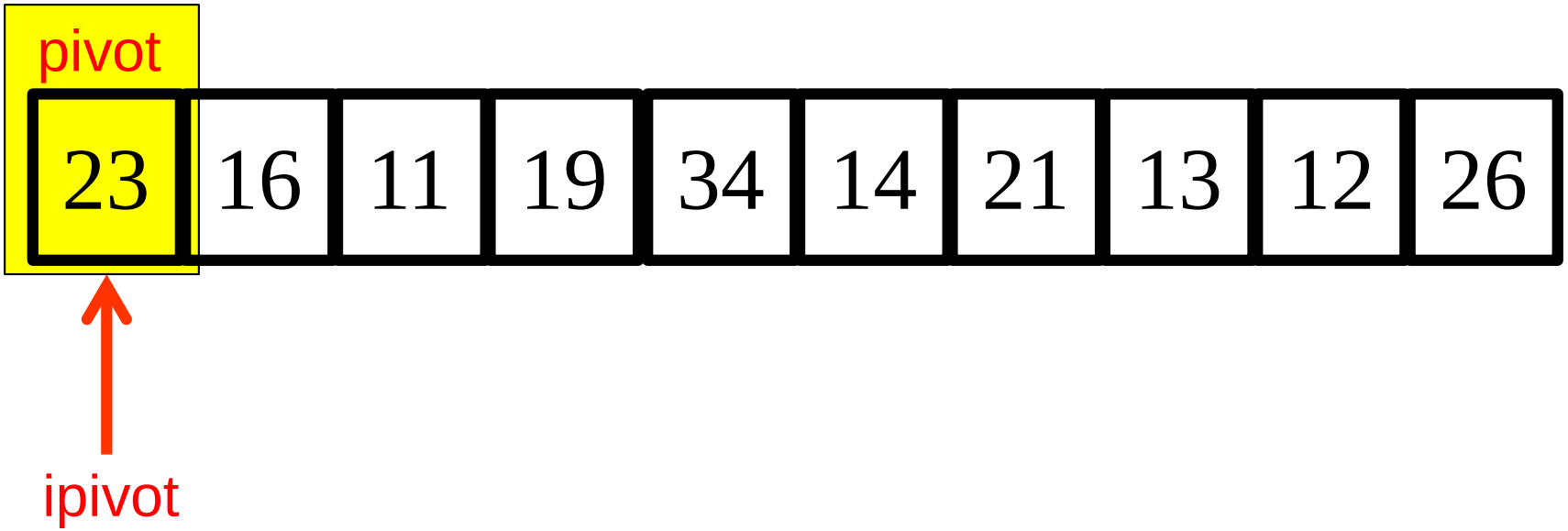
input

- ✓ un array (1D) ***A***
- ✓ il suo size ***n***

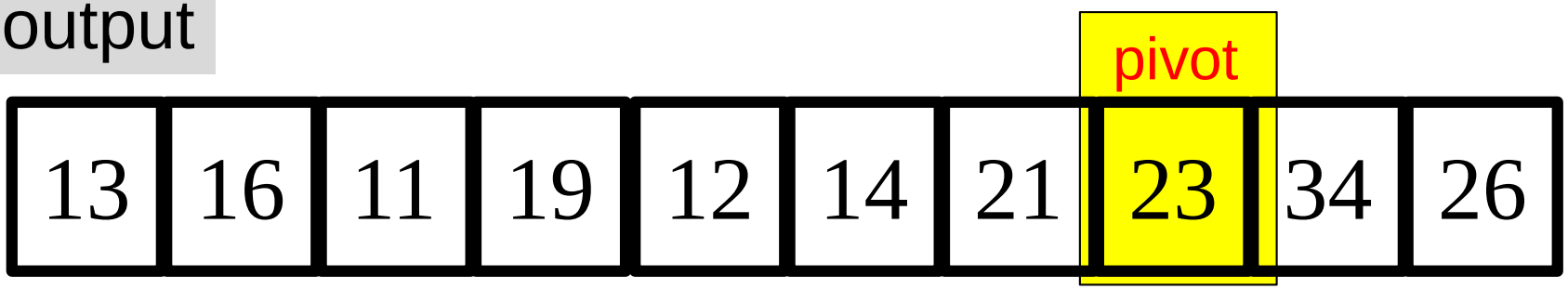
output

- ✓ l'array ***A*** **partizionato,** cioè con l'elemento ***A[0]*** , detto **pivot**, inserito in modo tale che **alla sua sinistra** ci sono tutti gli **elementi minori** (ma non necessariamente ordinati!) e **alla sua destra** tutti gli **elementi maggiori** (ma non necessariamente ordinati !)

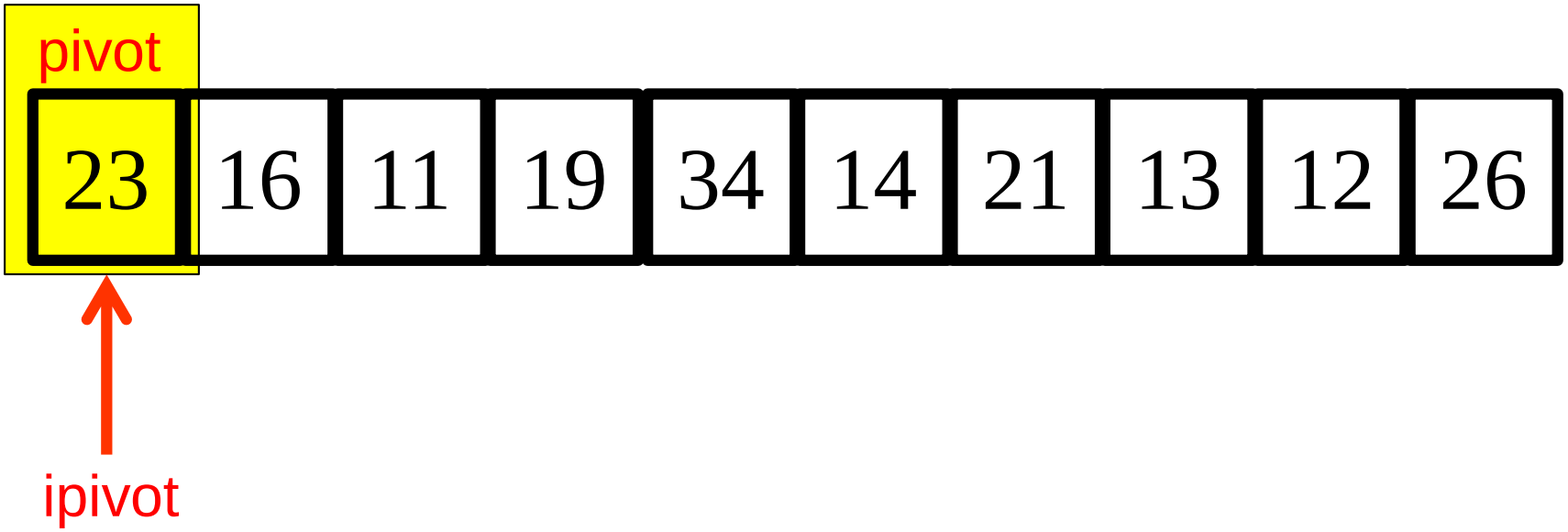
esempio di input



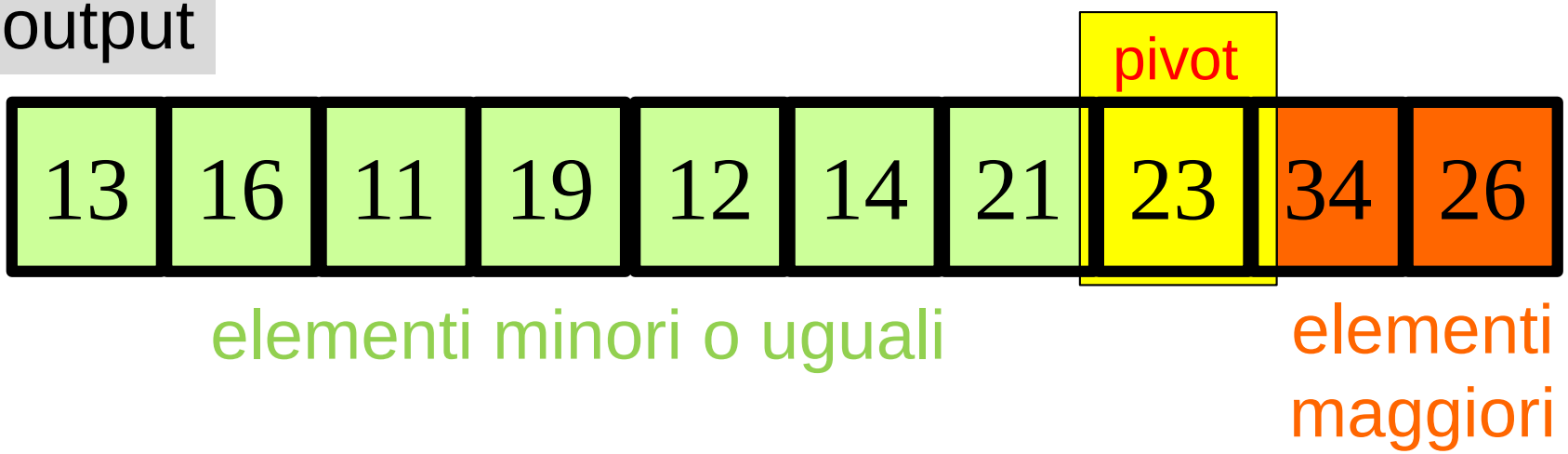
output

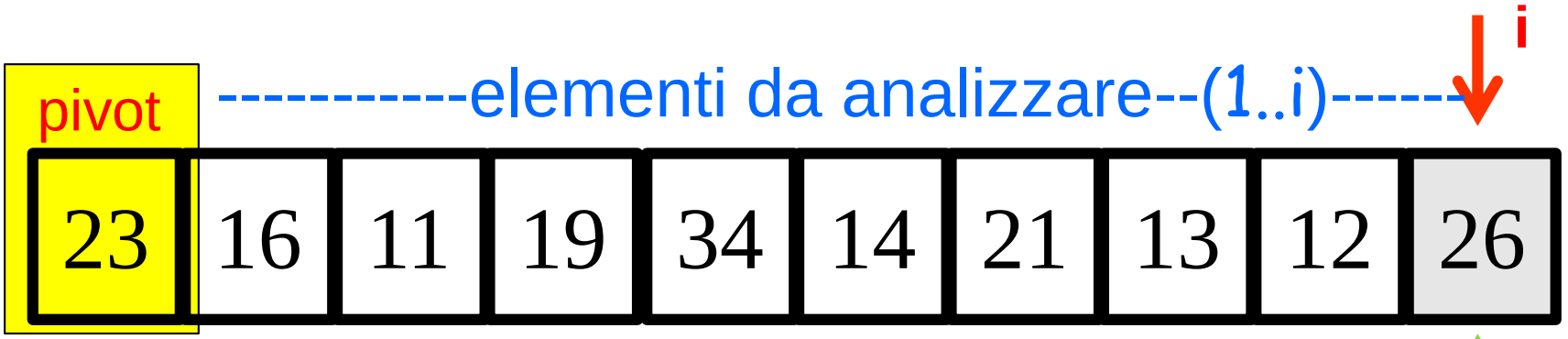


esempio di input



output



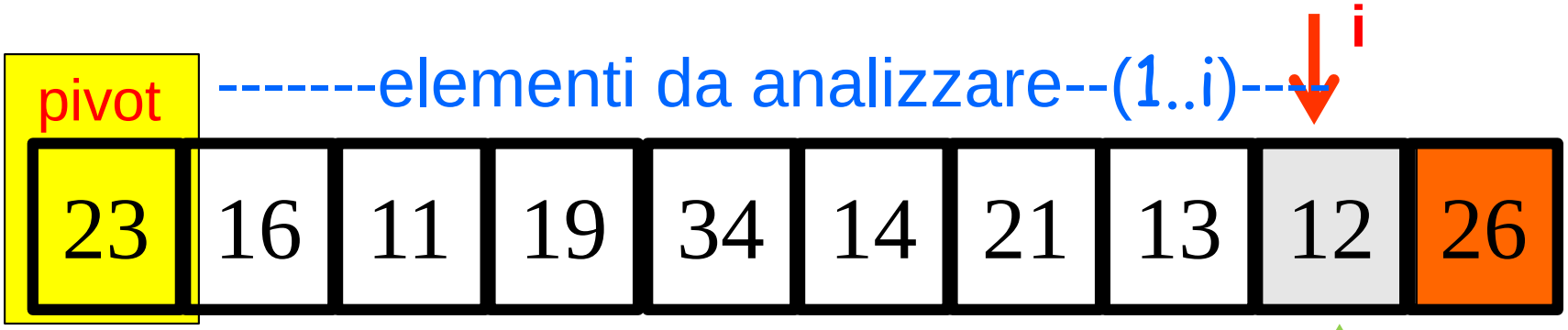



```
for (i=n-1; i>0; i--) {
    confrontare a[i]  con  pivot
     ....
}
```

23

pivot

l'indice memo individua la porzione (memo+1..n-1)
degli elementi maggiori di pivot



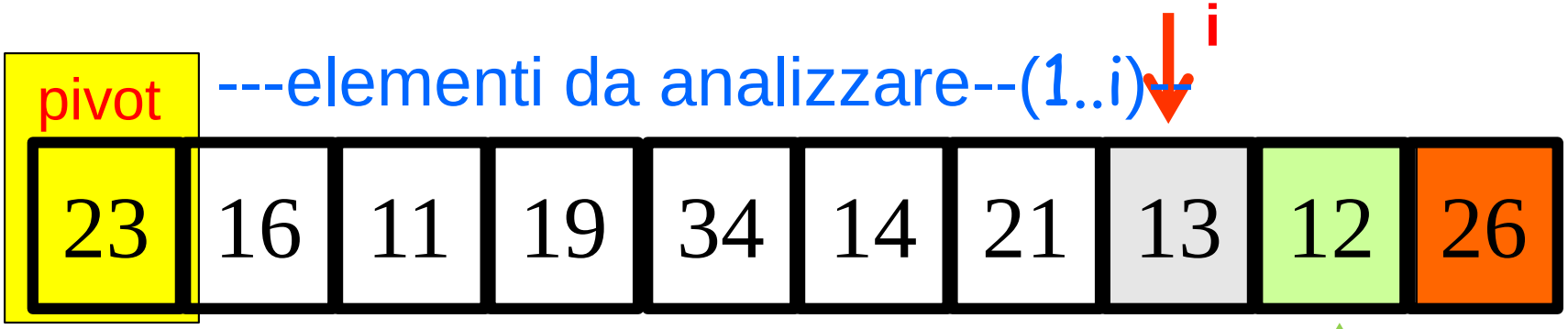
memo

```
for (i=n-1; i>0; i--) {
    confrontare a[i]  con  pivot
     ....
}
```

23

pivot

l'indice memo individua la porzione (memo+1..n-1)
degli elementi maggiori di pivot



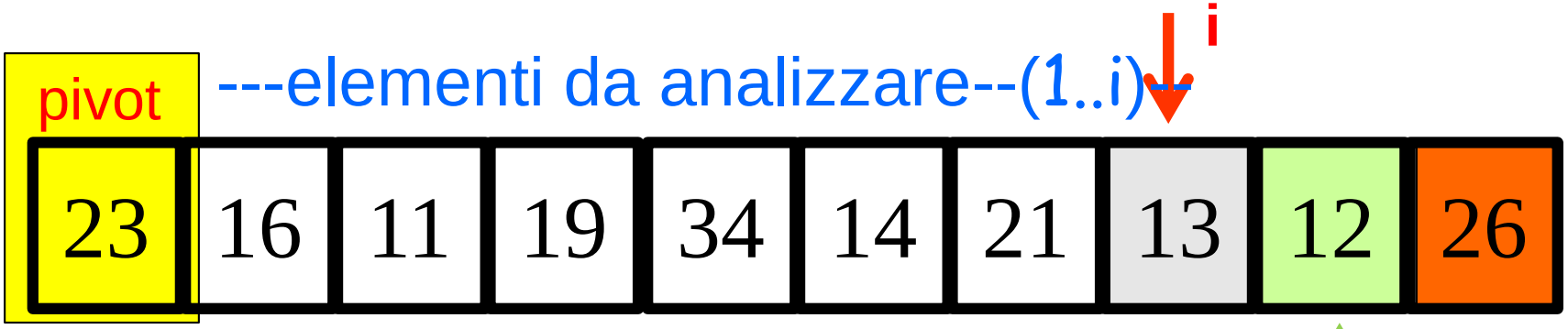

```
for (i=n-1; i>0; i--) {
    confrontare a[i]  con  pivot
     ....
}
```

memo

23

pivot

l'indice memo individua la porzione (memo+1..n-1)
degli elementi maggiori di pivot



memo

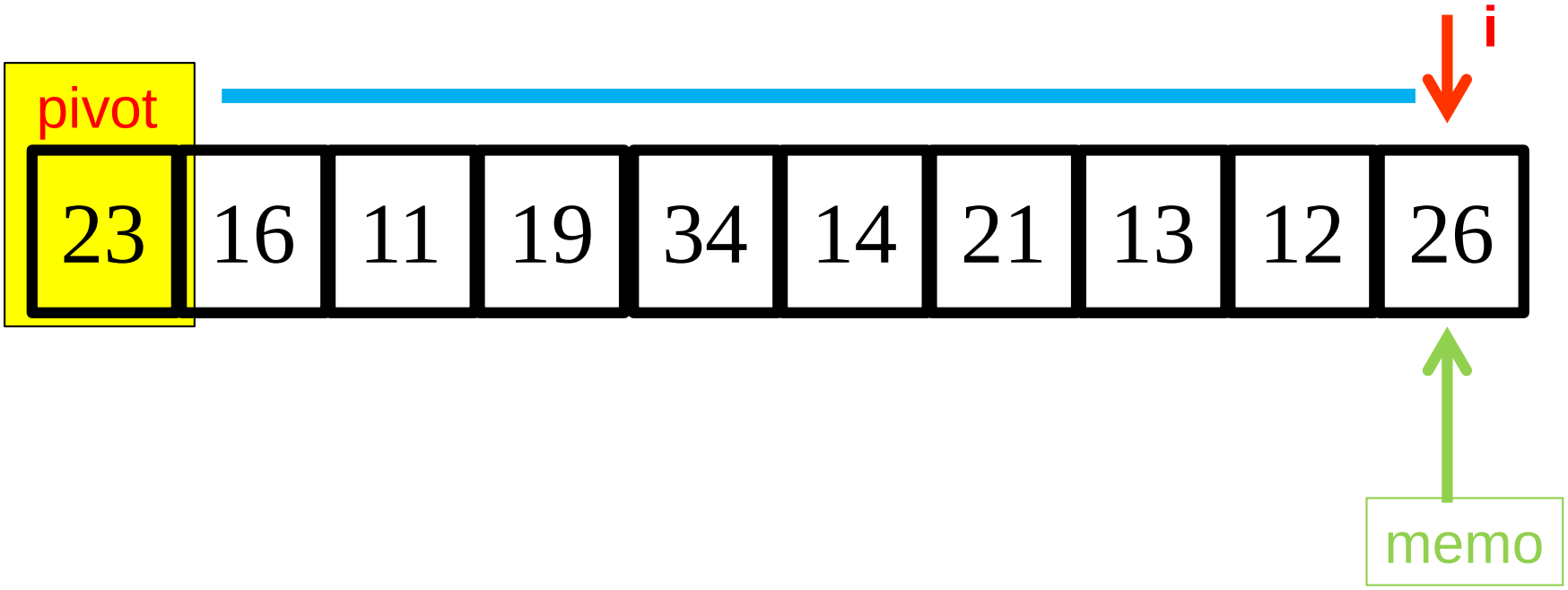
```
for (i=n-1; i>0; i--) {
    confrontare a[i]  con  pivot
     ....
}
```

23

pivot

l'indice memo individua

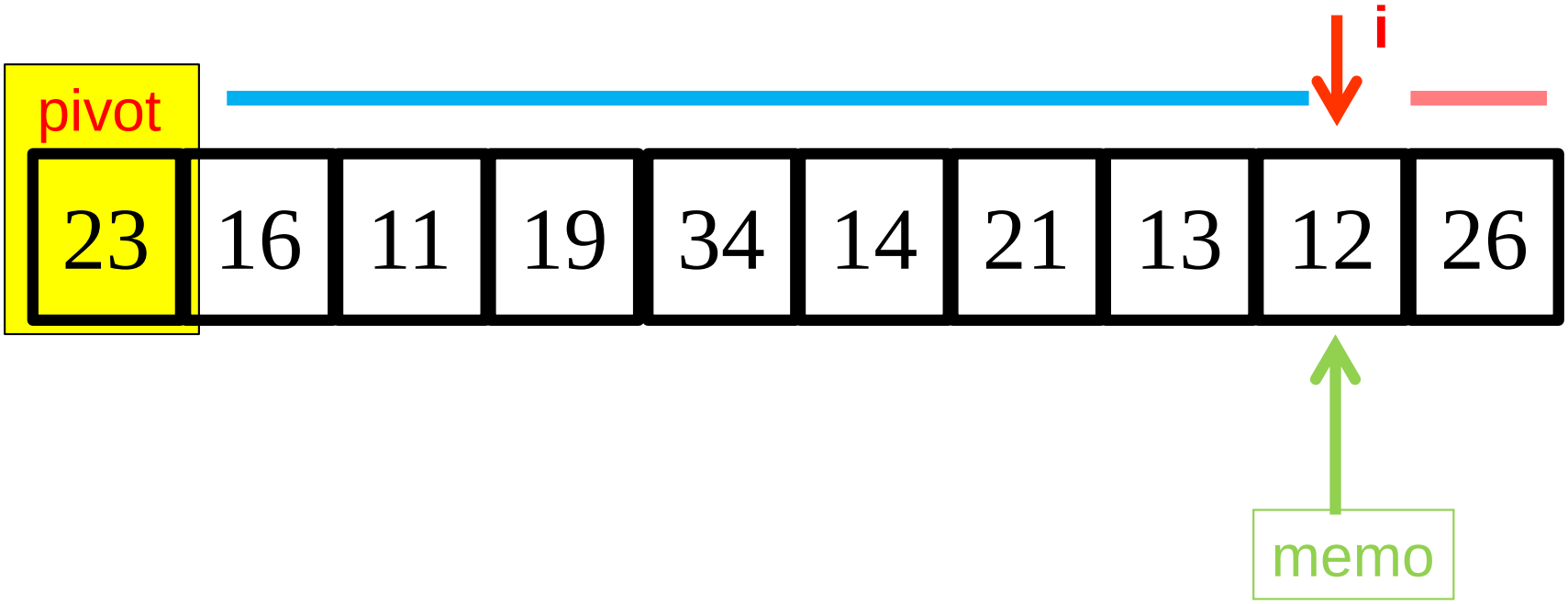
- ✓ la porzione (memo+1..n-1) degli elementi > di pivot
- ✓ la porzione (i+1..memo) degli elementi <= di pivot



23

pivot

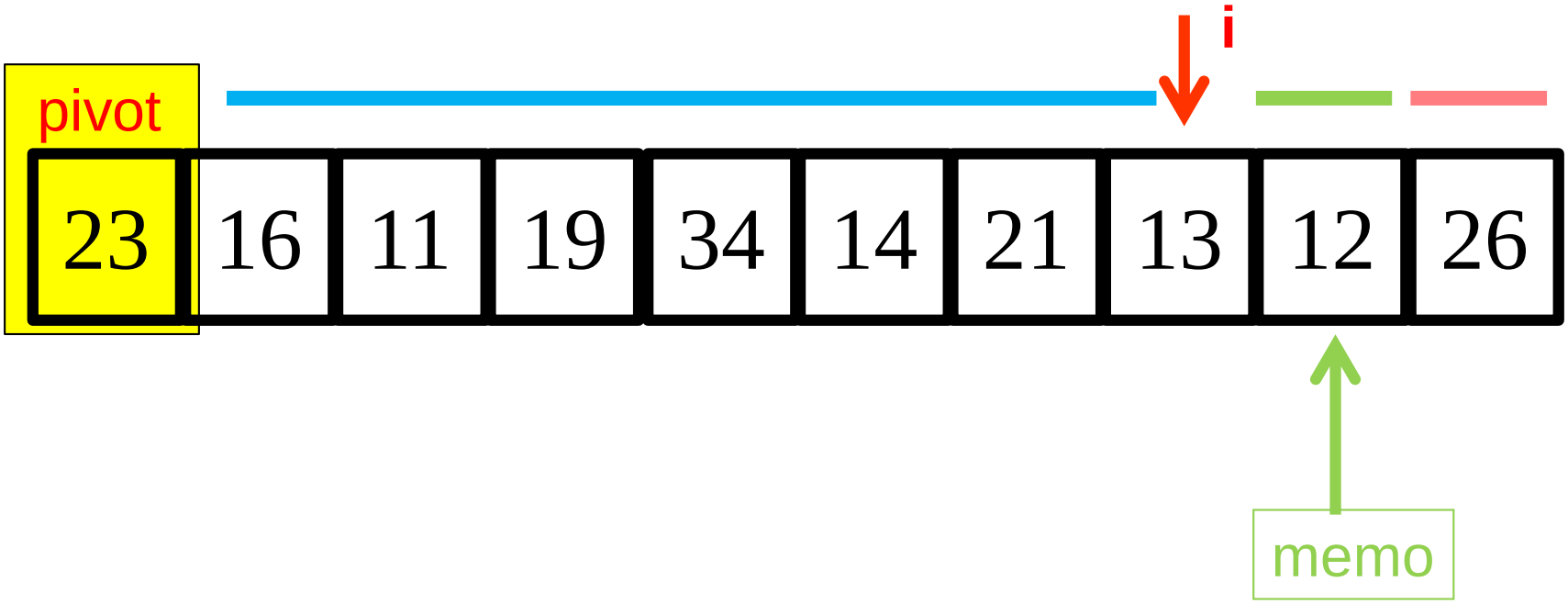
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

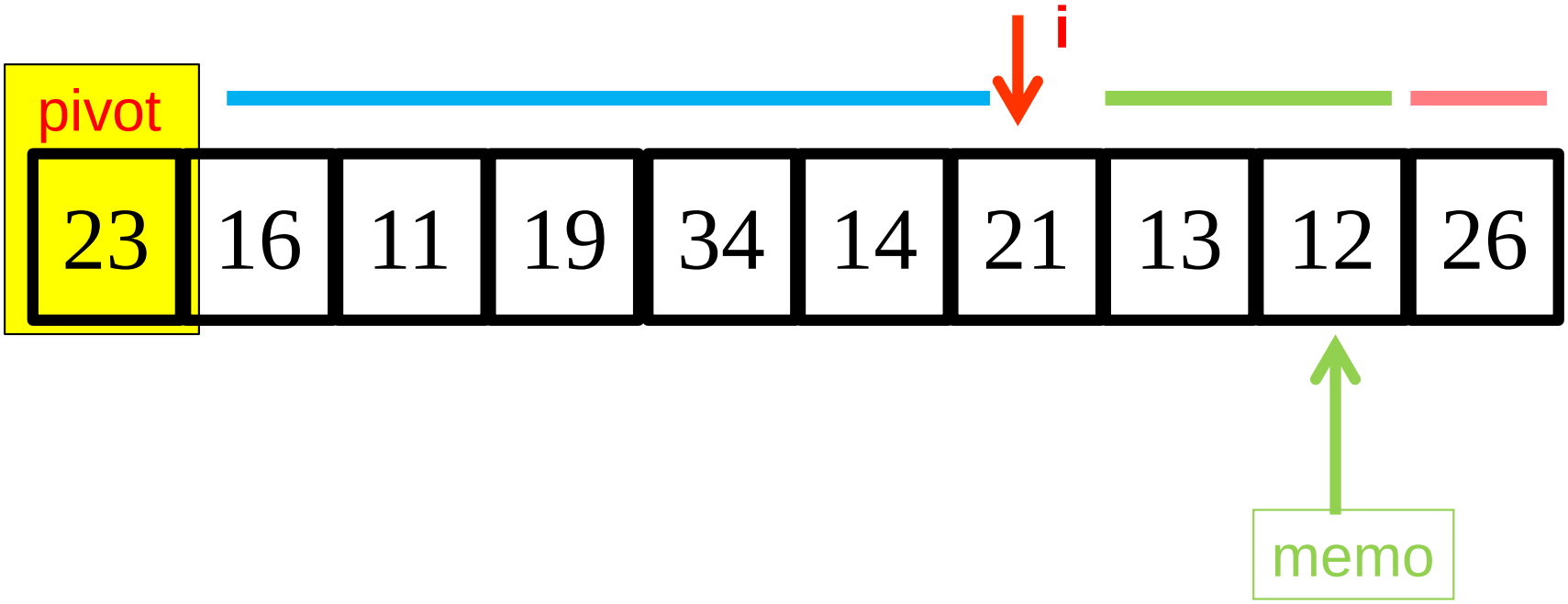
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

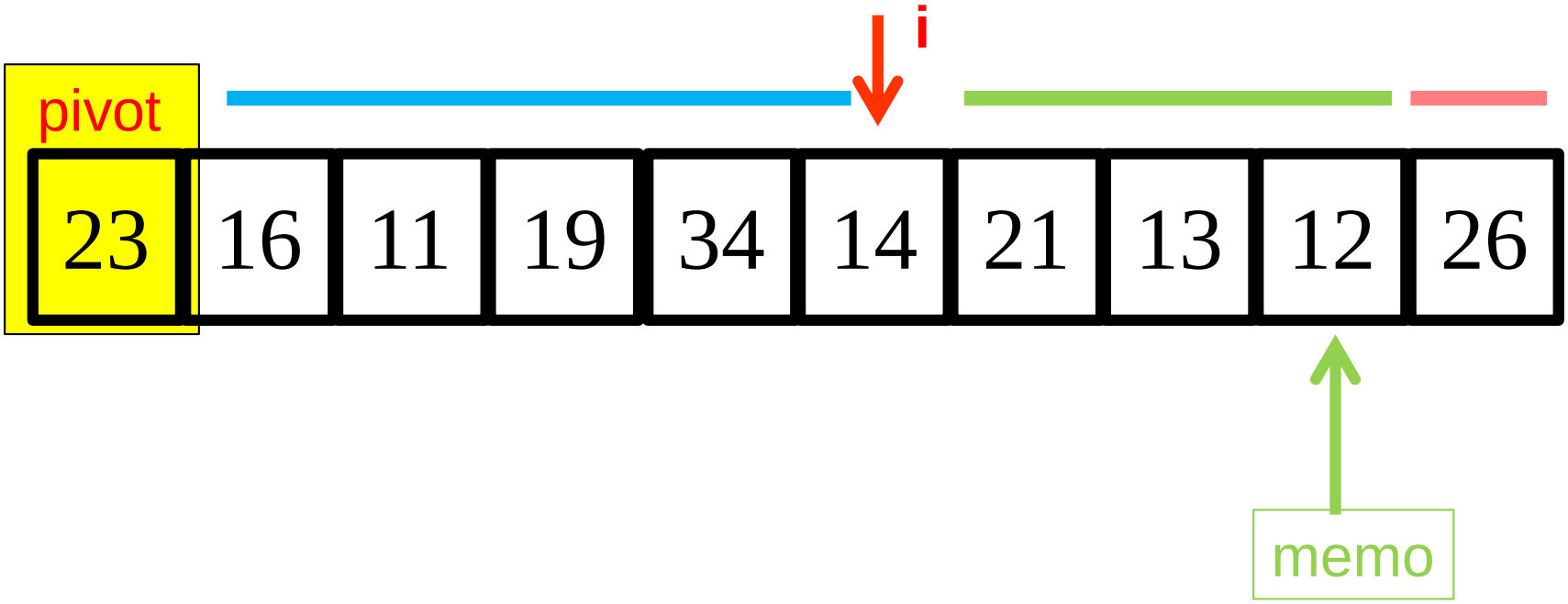
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

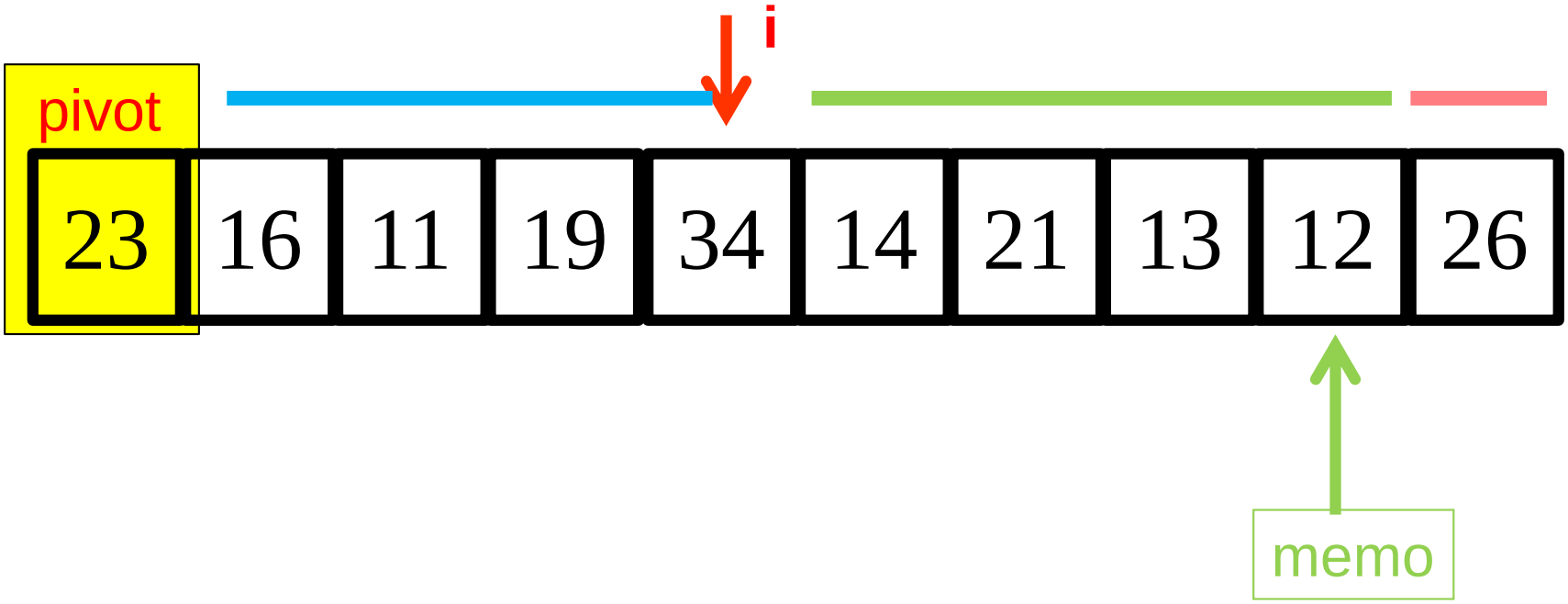
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

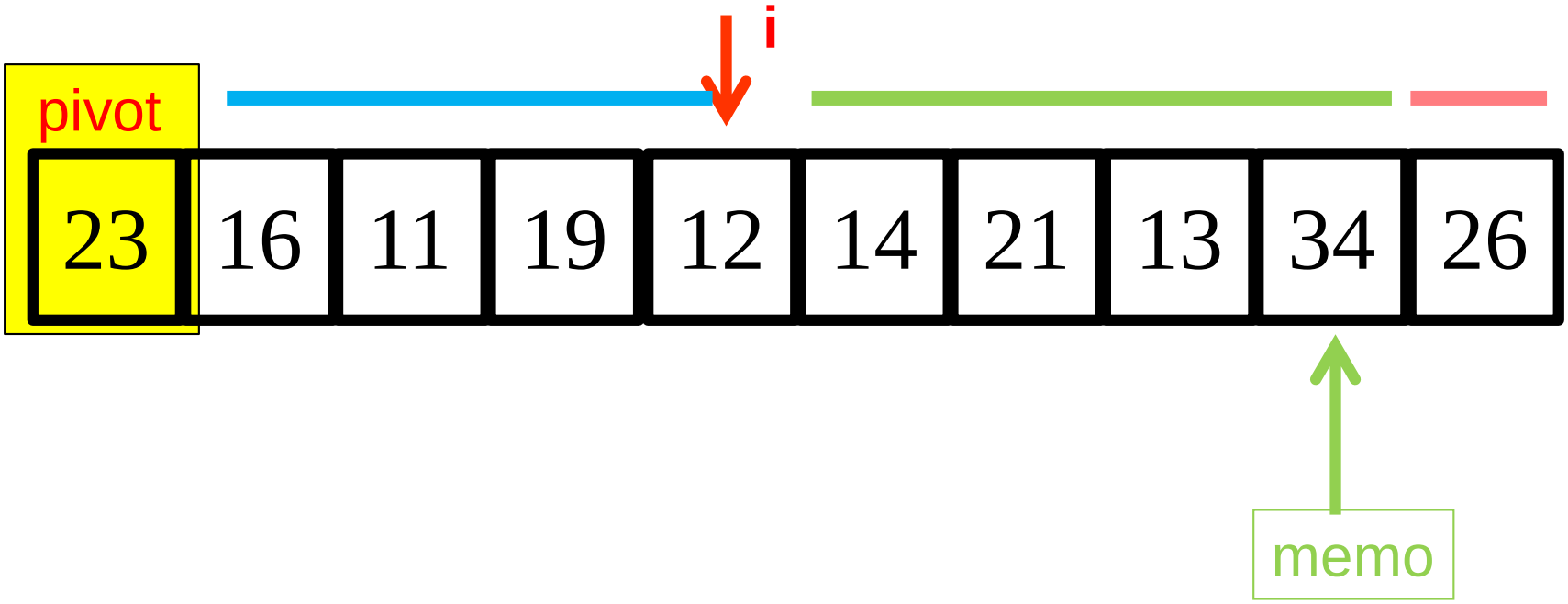
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

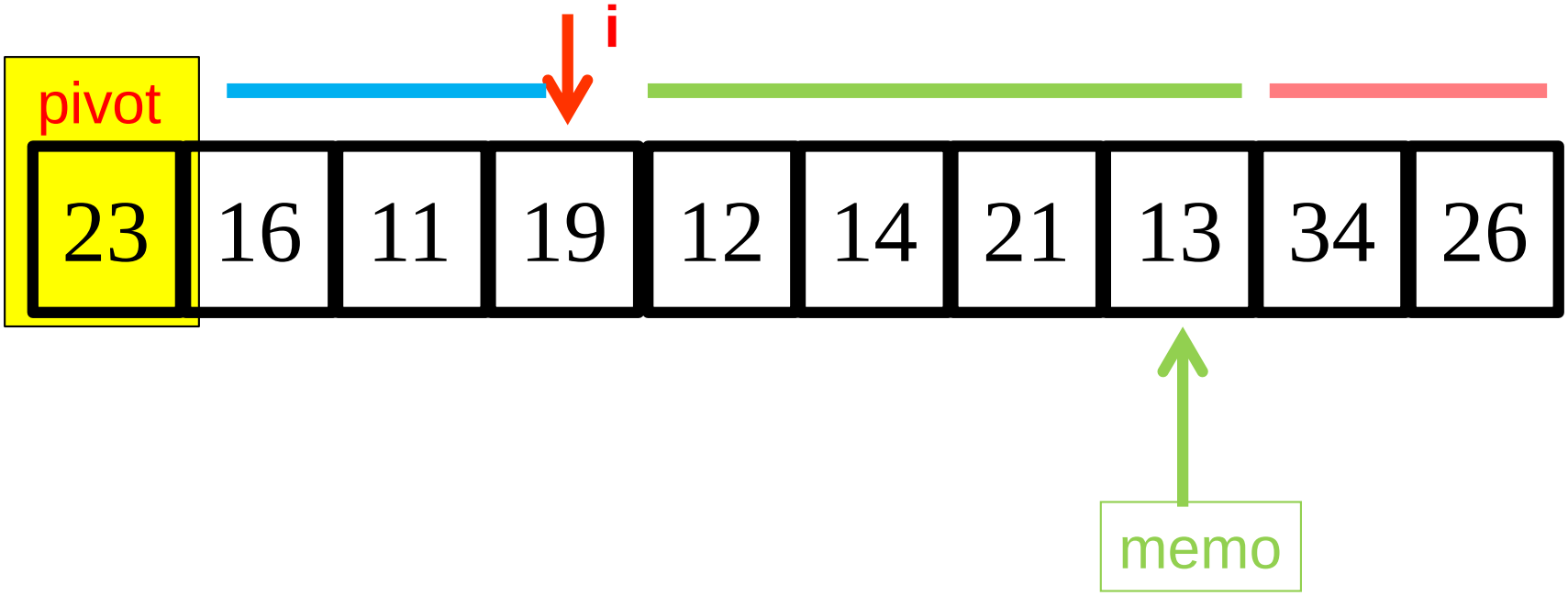
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

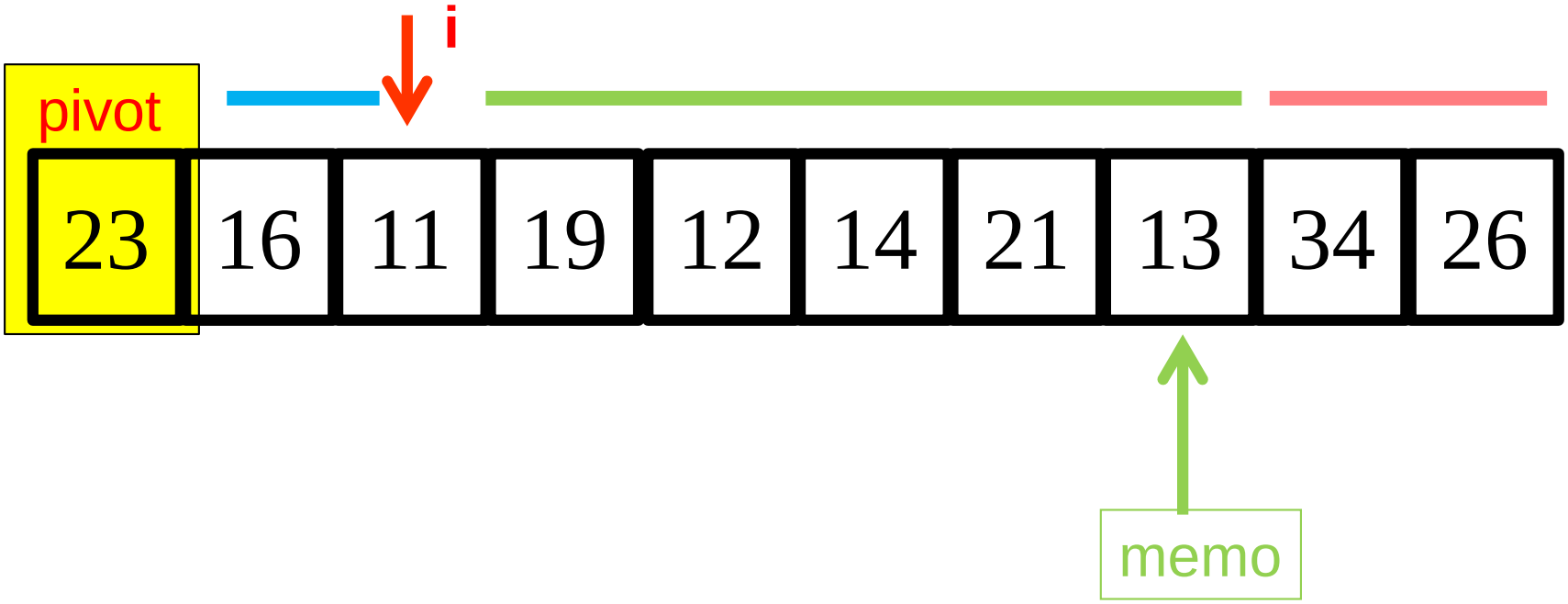
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

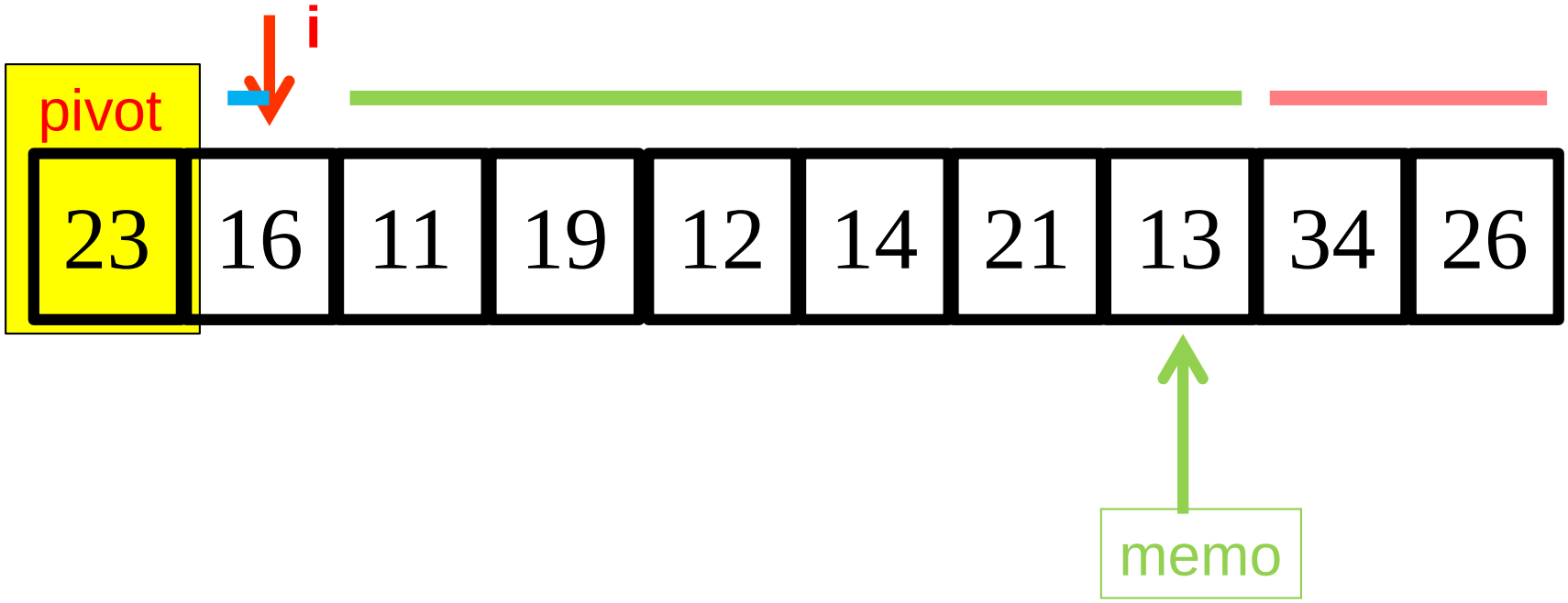
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

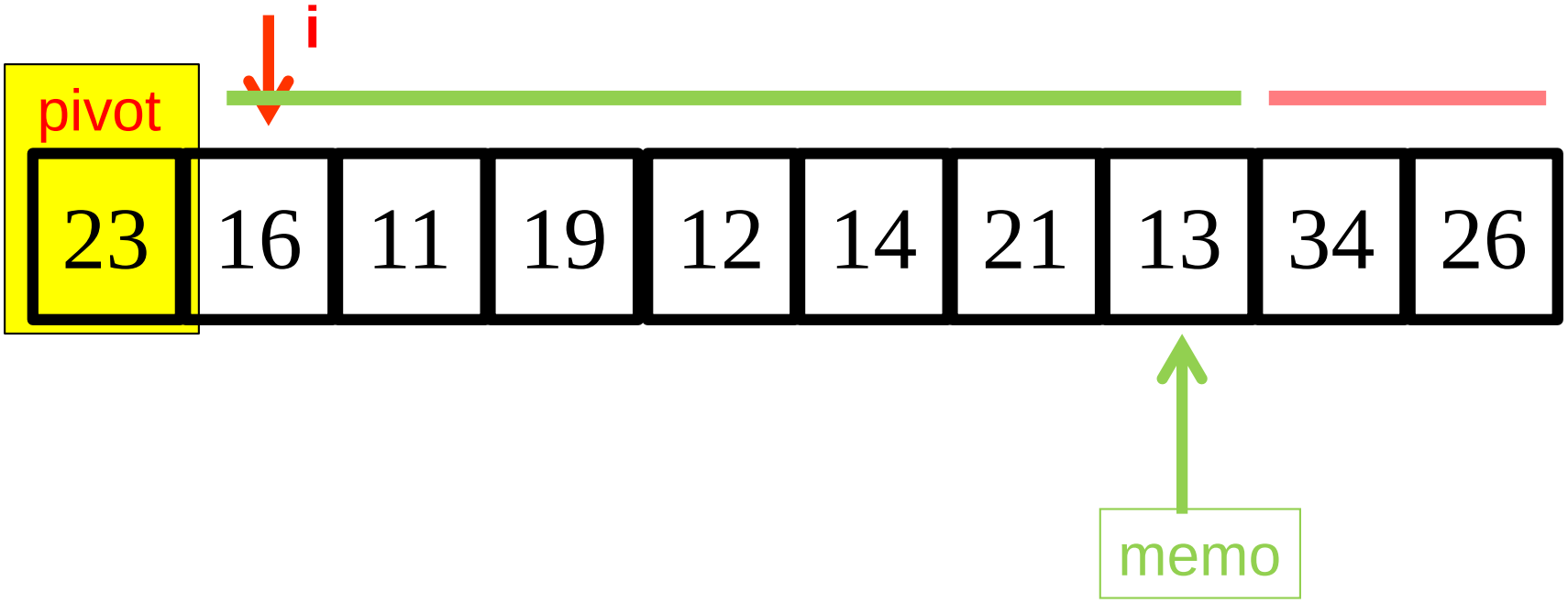
```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```



23

pivot

```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```

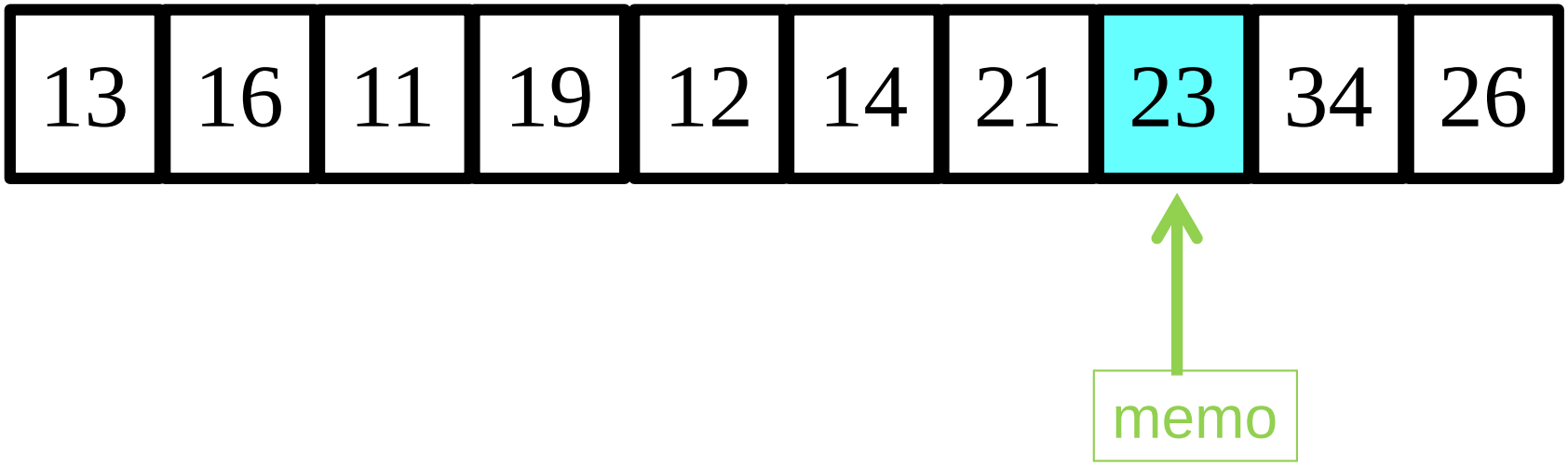


23

pivot

```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```

```
scambiare(&a[0], &a[memo])
```

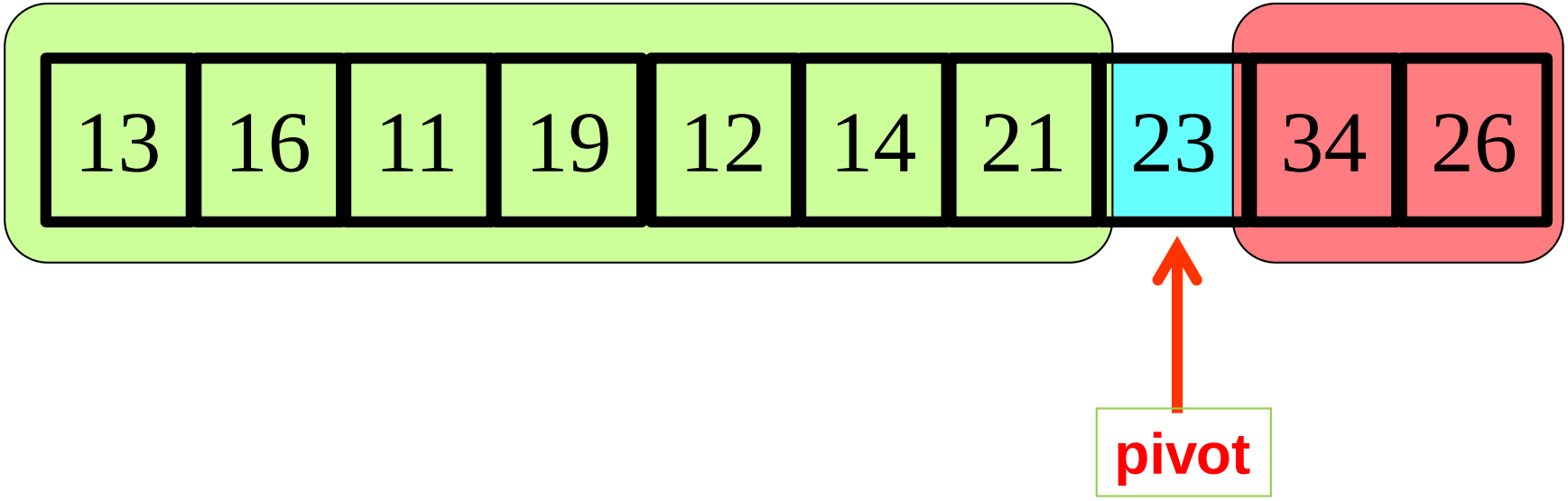


23

pivot

```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```

```
scambiare(&a[0], &a[memo])
```



23

pivot

```
if (a[i] > pivot)
   {
      scambiare(&a[i], &a[memo]) ;
      memo = memo - 1 ;
   } ;
```

```
scambiare(&a[0], &a[memo])
```

```
void partition (int a[], int n) {
int i,pivot, memo;
 pivot = a[0] ;
 memo = n-1 ;
 for (i=n-1, i >= 1, i--) {
    if (a[i] > pivot)
     {
       scambiare(&a[i], &a[memo]) ;
        memo = memo - 1 ;
     }
 }
 scambiare(&a[0], &a[memo]) ;
}
```

n-1 confronti

n scambi (al più)

Esercizio:

usare
`for (i=0; i<n-1; i++)`

input
- ✓ un array (1D) *A*
- ✓ il suo size *n*

output
- ✓ l'array *A* **partizionato**, cioè con l'elemento *A*[n-1], detto **pivot**, inserito in modo tale che **alla sua sinistra** ci sono tutti gli **elementi minori** (ma non necessariamente ordinati!) e **alla sua destra** tutti gli **elementi maggiori** (ma non necessariamente ordinati !)