Fundamentals of Computing: Lecture 21

Piyush P Kurur
Office no: 224
Dept. of Comp. Sci. and Engg.
IIT Kanpur

September 16, 2009

Merge sort

- In the last class we saw quick sort.
- This class we will see another algorithm called merge sort.

Merging sorted lists

Given two sorted lists $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_m\}$ how fast can we merge ?

Merging sorted lists

Given two sorted lists $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_m\}$ how fast can we merge ?

```
void mergearrays(int a[], int b[], int c[], int n, int m)
{
  int i=0,j=0,k=0;

  while( i < n && j < m)
  {
    if(a[i] < b[j]) { c[k] = a[i]; i++}
    else if(a[i] >= b[j]){ c[k] = b[j]; j++}
    k++;
  }

  while(i < n){ c[k] = a[i]; i++; k++;}
  while(j < m){ c[k] = b[j]; j++; k++;}
}
```

Main idea

- If the array is of size 1 then it is already sorted.
- Otherwise divide the array into two equal parts say A and B
- Sort A and B recursively.
- Merge A and B to get the sorted list.

Main idea

- If the array is of size 1 then it is already sorted.
- Otherwise divide the array into two equal parts say A and B
- Sort A and B recursively.
- Merge A and B to get the sorted list.

How many comparisons

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n = 2T\left(\frac{n}{2}\right) + n.$$

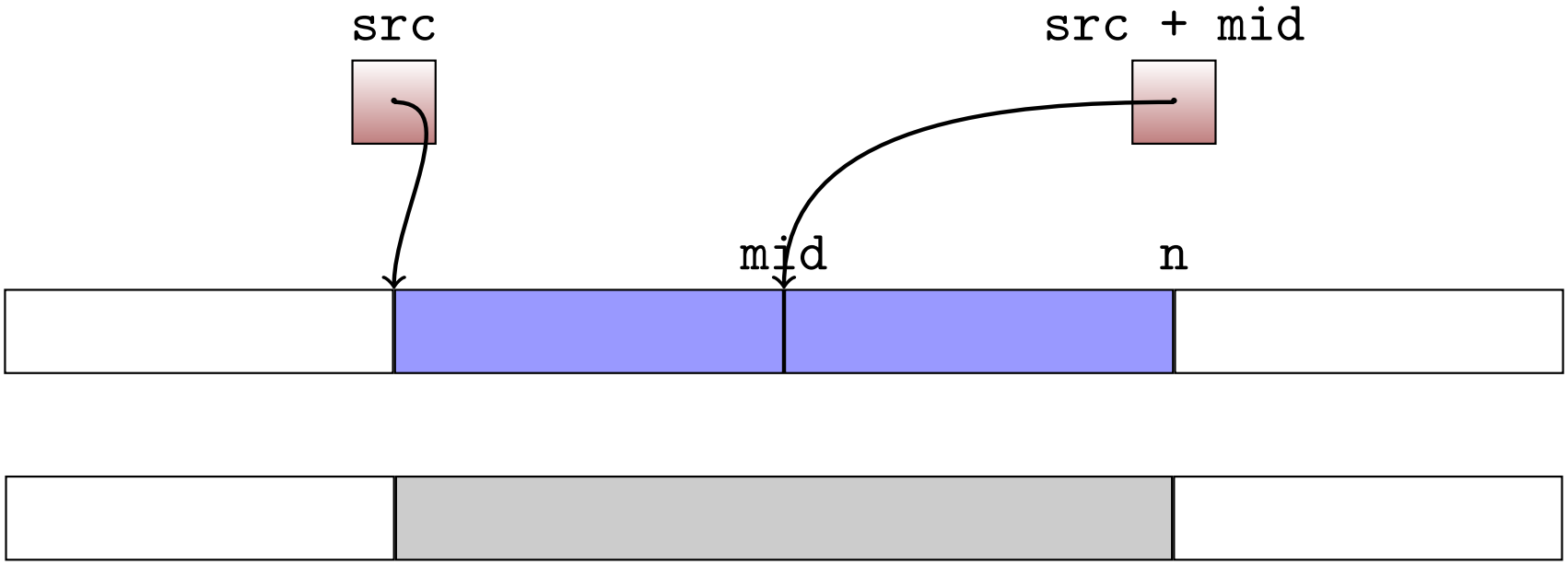
Unrolling the recursion

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n \\
&= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \\
&= 2^2 T\left(\frac{n}{2^2}\right) + (n+n) \\
\ldots \quad &\ldots \quad \ldots \\
&= 2^k T\left(\frac{n}{2^k}\right) + \overbrace{n + \ldots + n}^{k \text{ times}}
\end{aligned}
$$

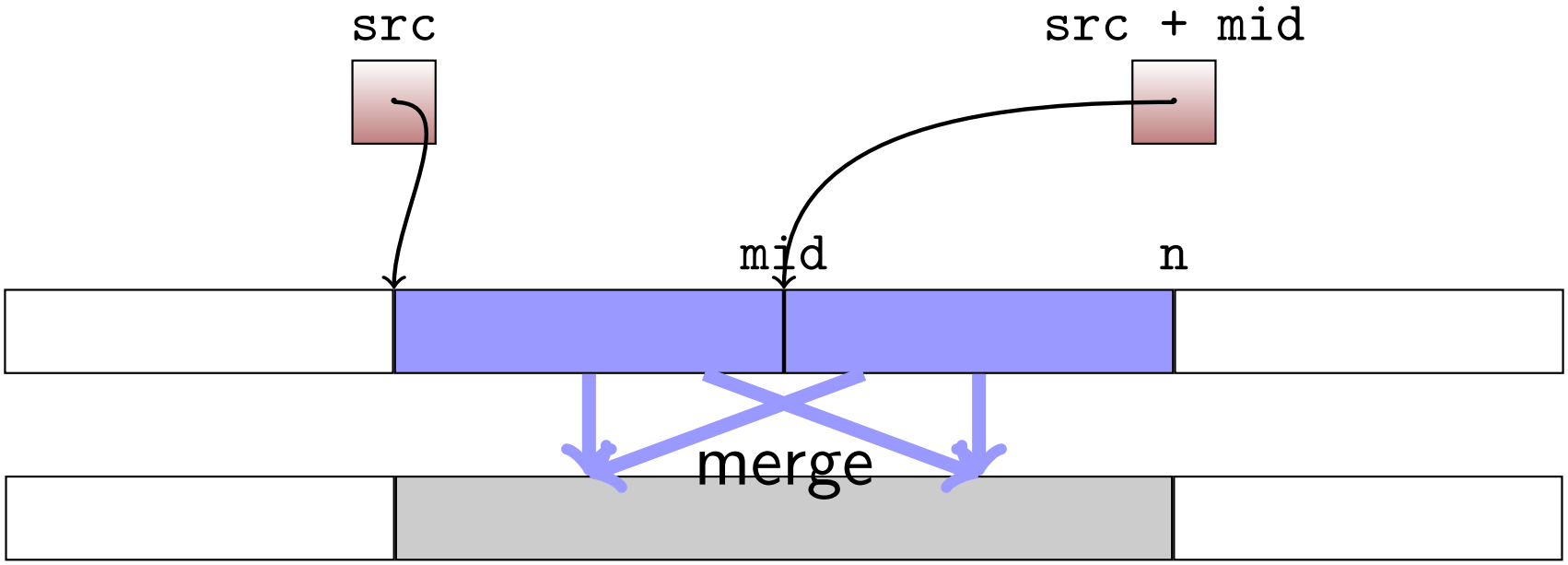
Choosing $k = \log n$ we have

$$T(n) = 2^{\log n} T(1) + n \log n = O(n \log n).$$

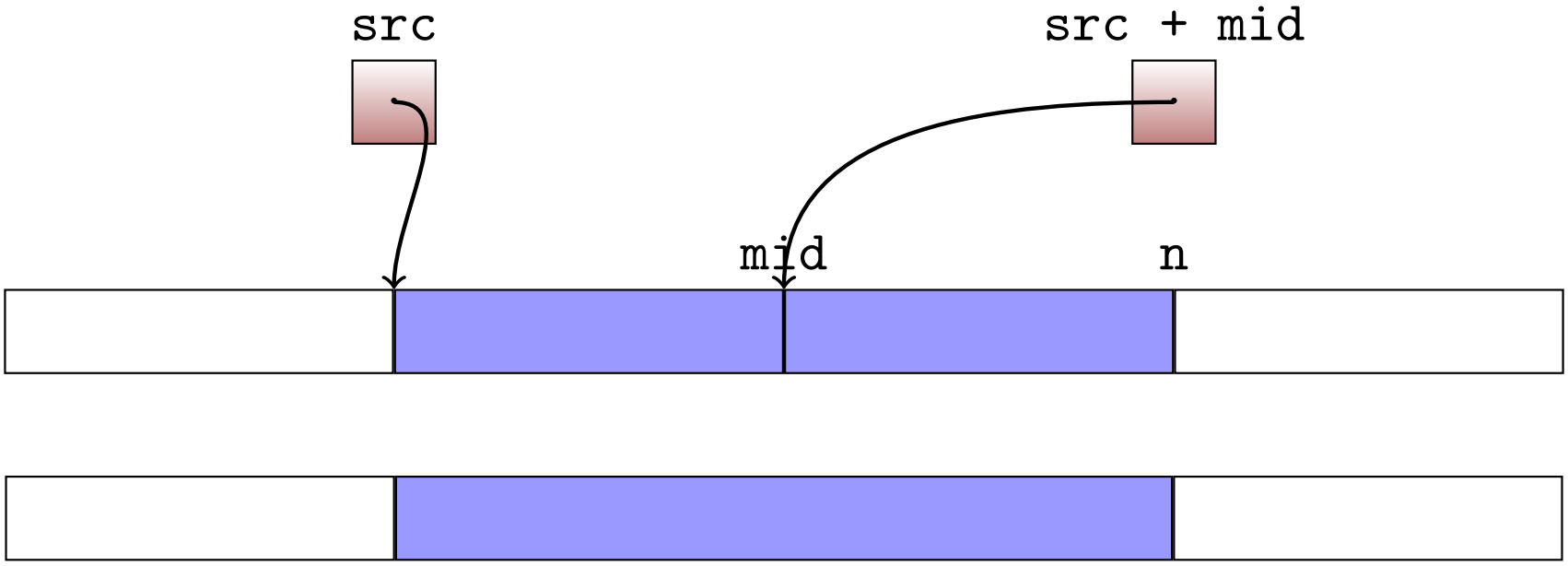
Merging array slices



Merging array slices



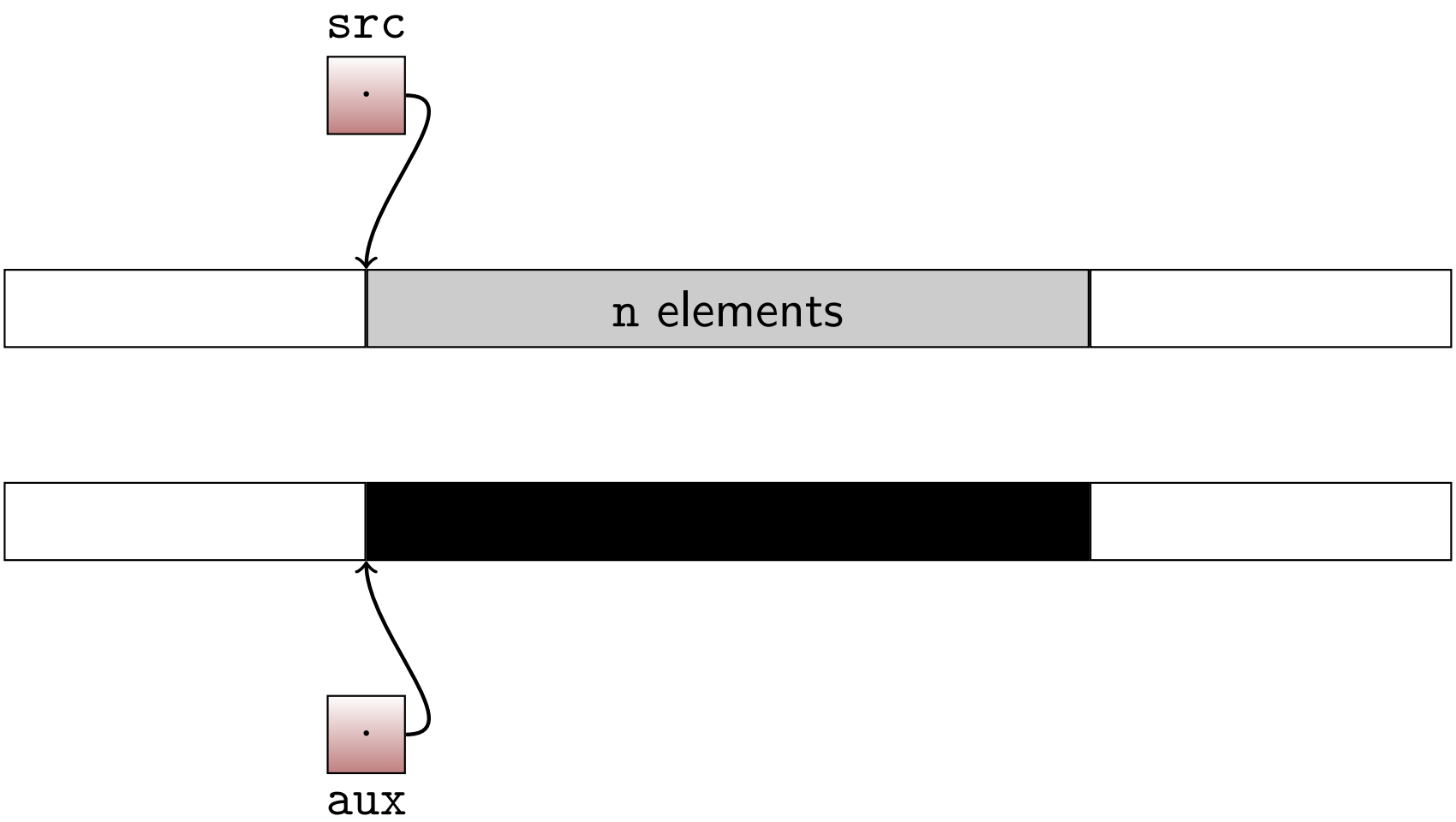
Merging array slices



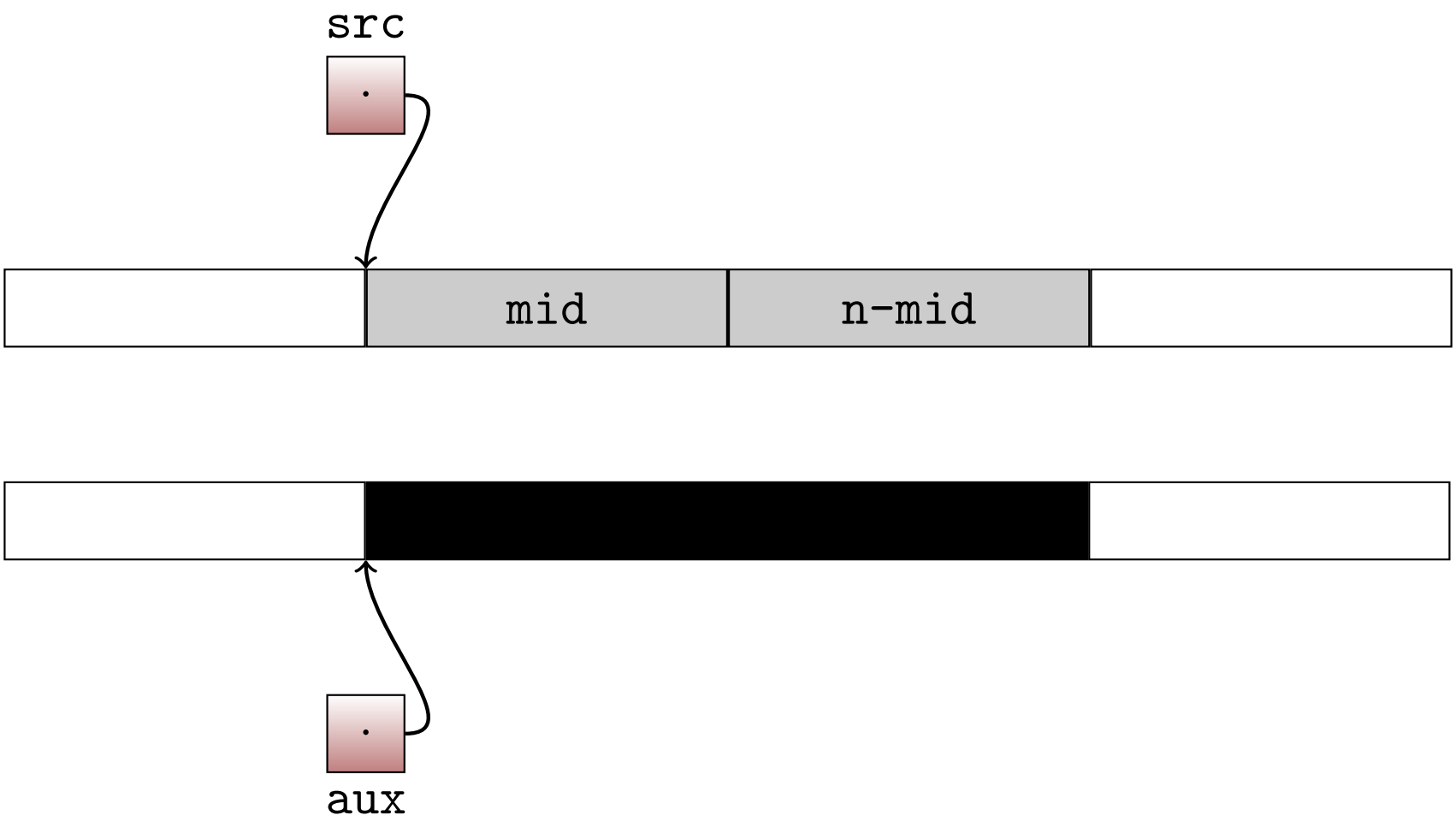
Merging array slices in C

```
int merge(int *src, int mid, int n, int *dest)
{
  int i=0, j = 0, k = 0;

  int *a = src;
  int *b = src + mid;
  while( i < mid && j < n - mid)
  {
    if(a[i] < b[j]) { dest[k] = a[i]; i++}
    else if(a[i] >= b[j]){ dest[k] = b[j]; j++}
    k++;
  }
  while(i < mid)       { dest[k] = a[i]; i++; k++;}
  while(j < n - mid)   { dest[k] = b[j]; j++; k++;}

}
```

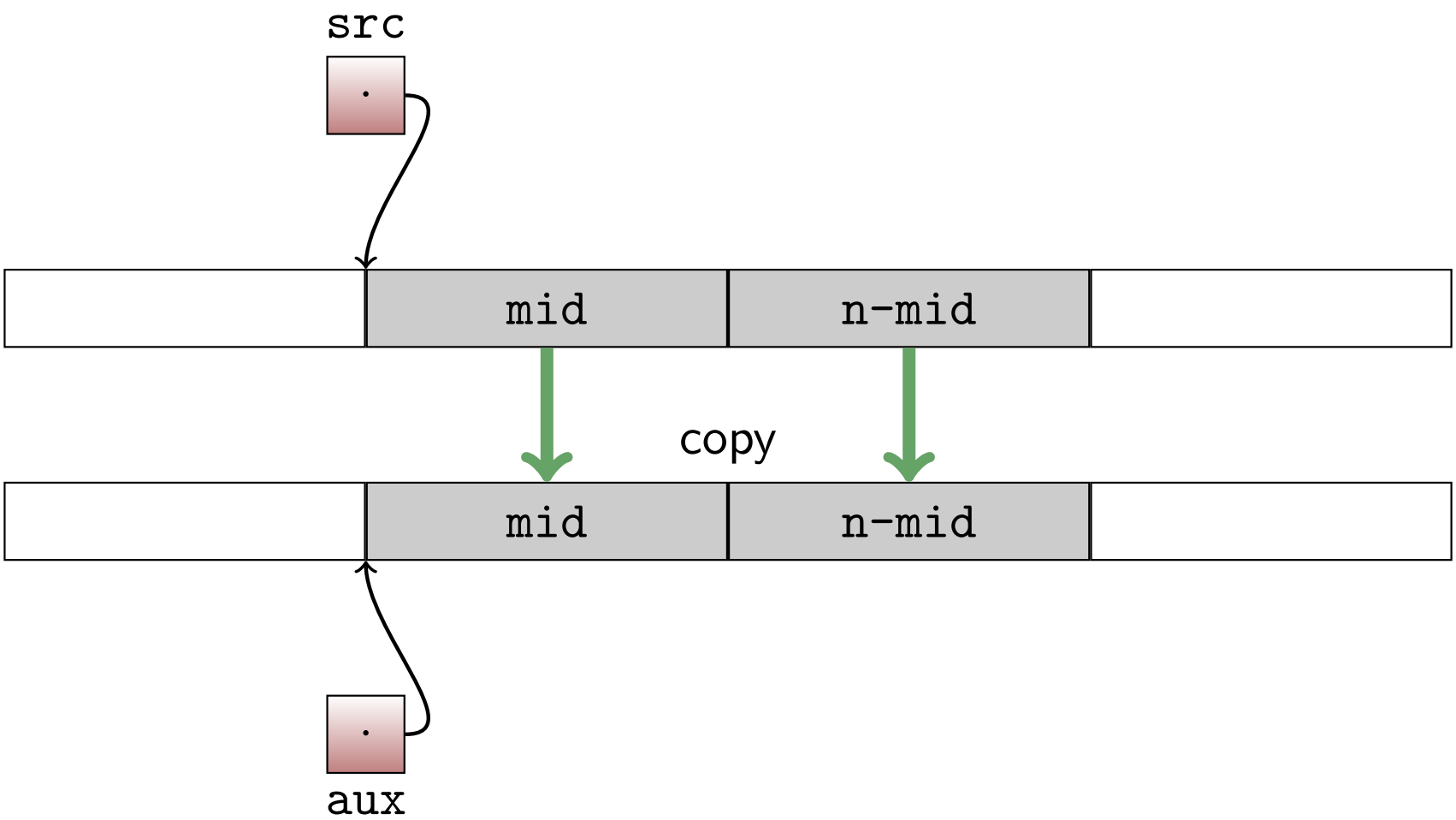
Merge-sorting array using an auxilary array



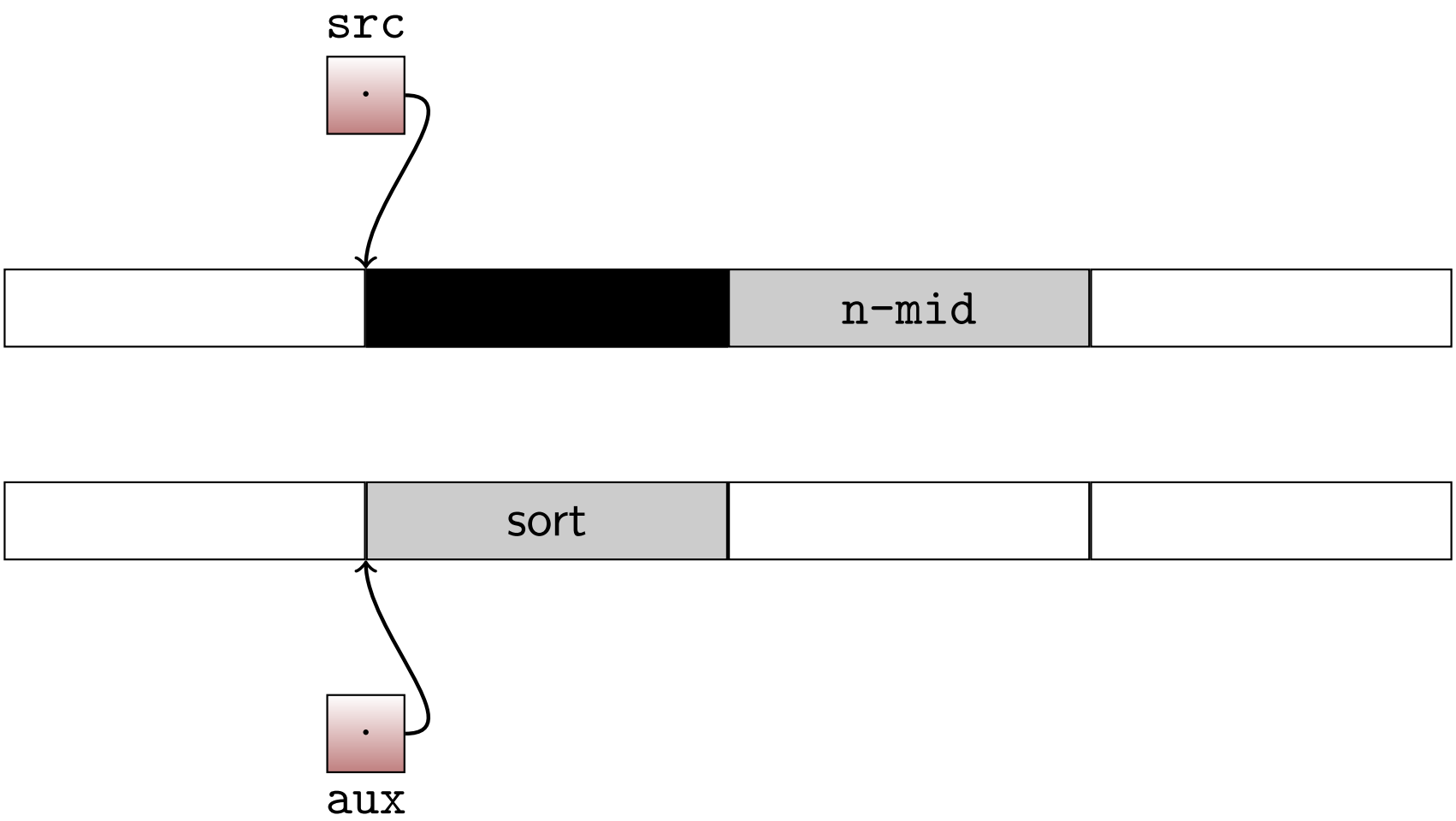
Merge-sorting array using an auxilary array



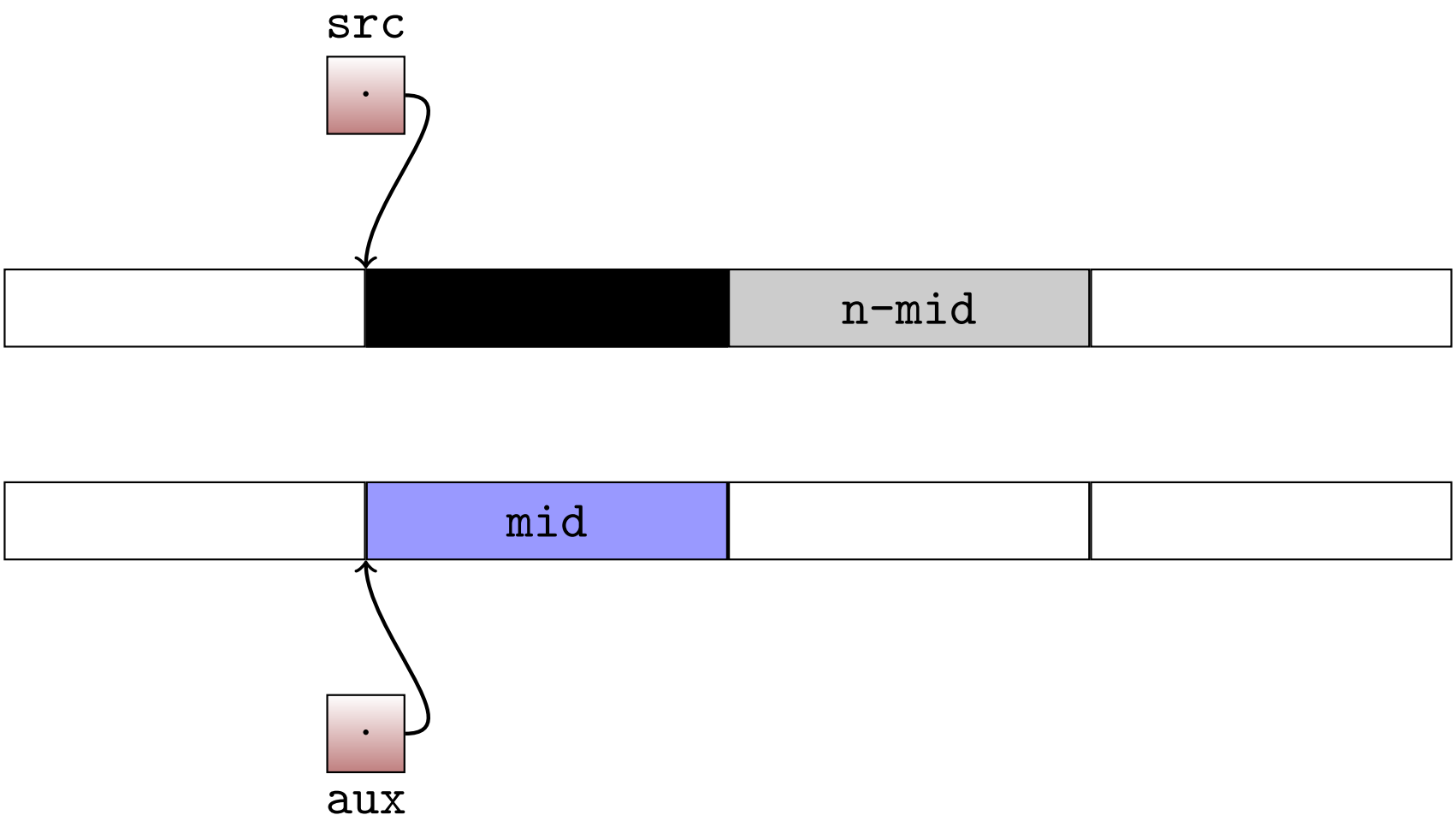
Merge-sorting array using an auxilary array



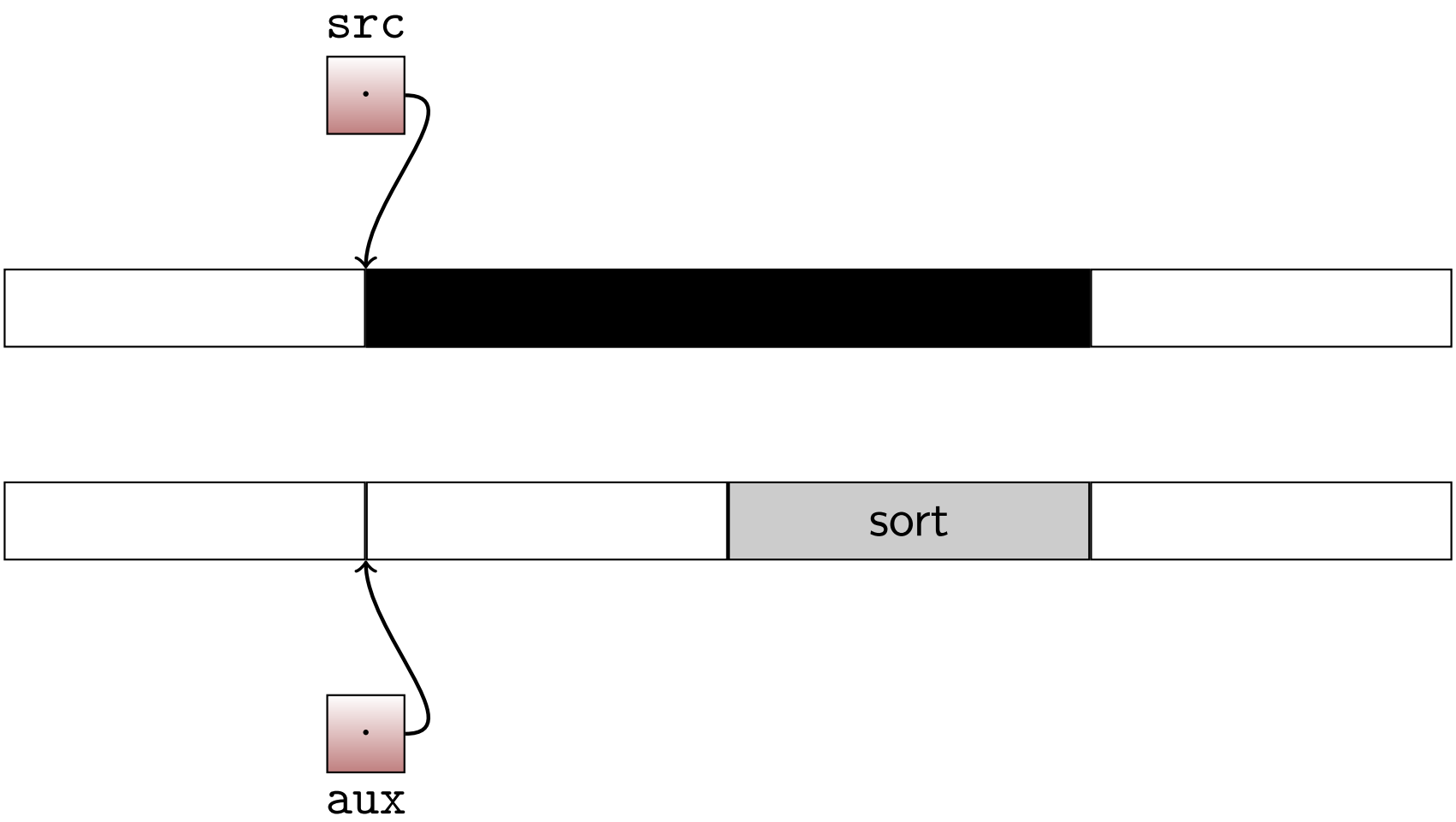
Merge-sorting array using an auxilary array



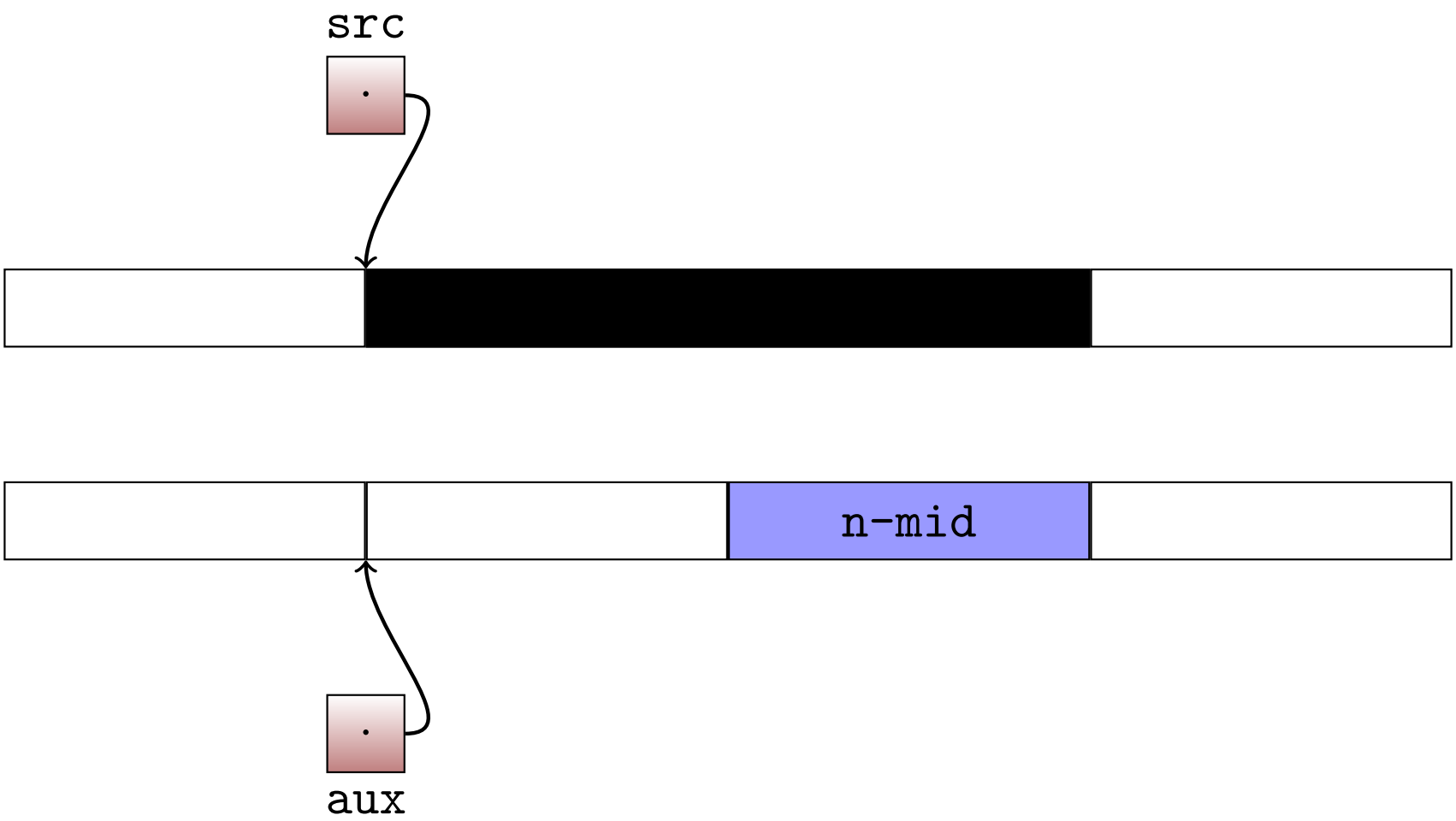
Merge-sorting array using an auxilary array



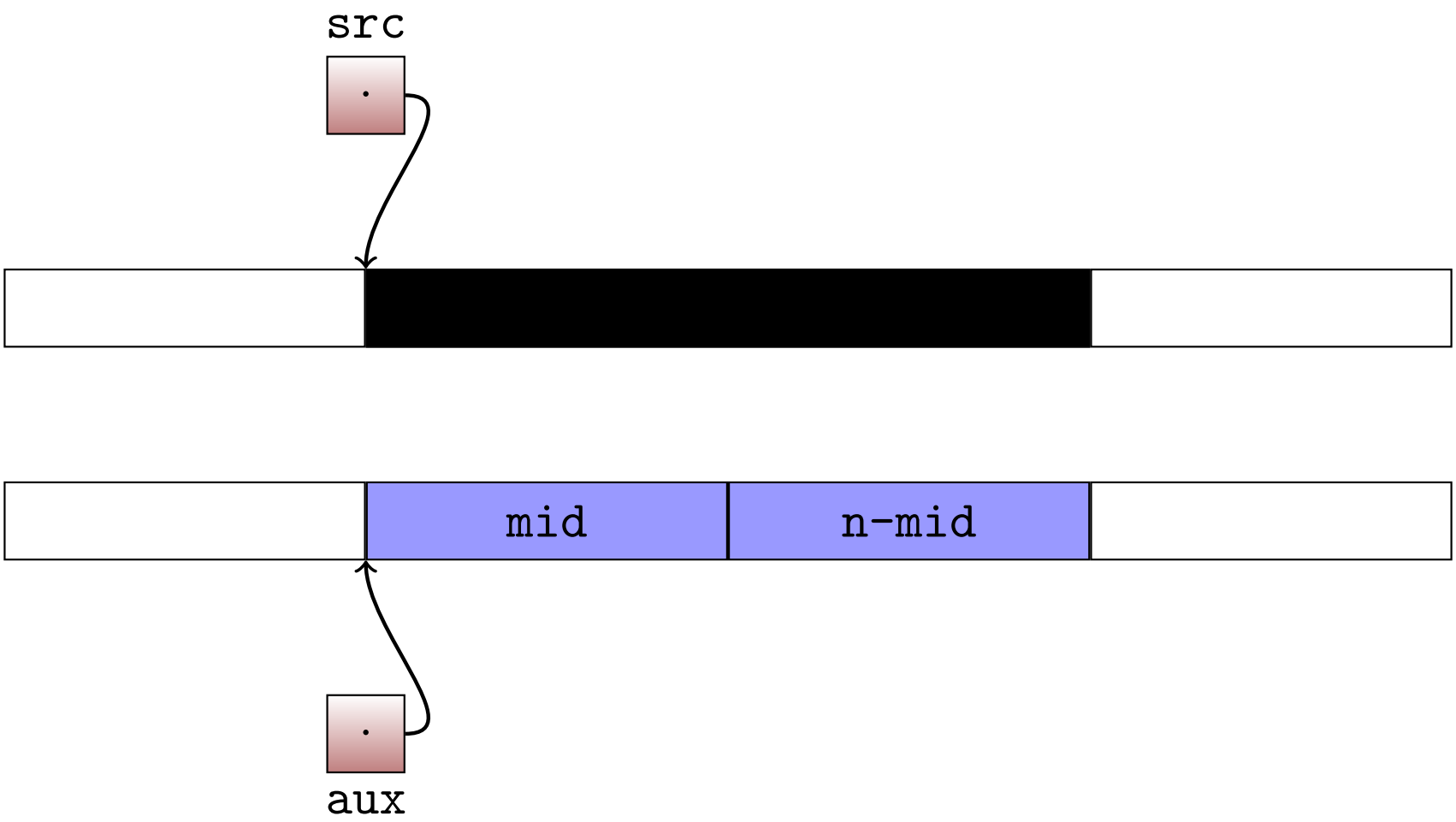
Merge-sorting array using an auxilary array



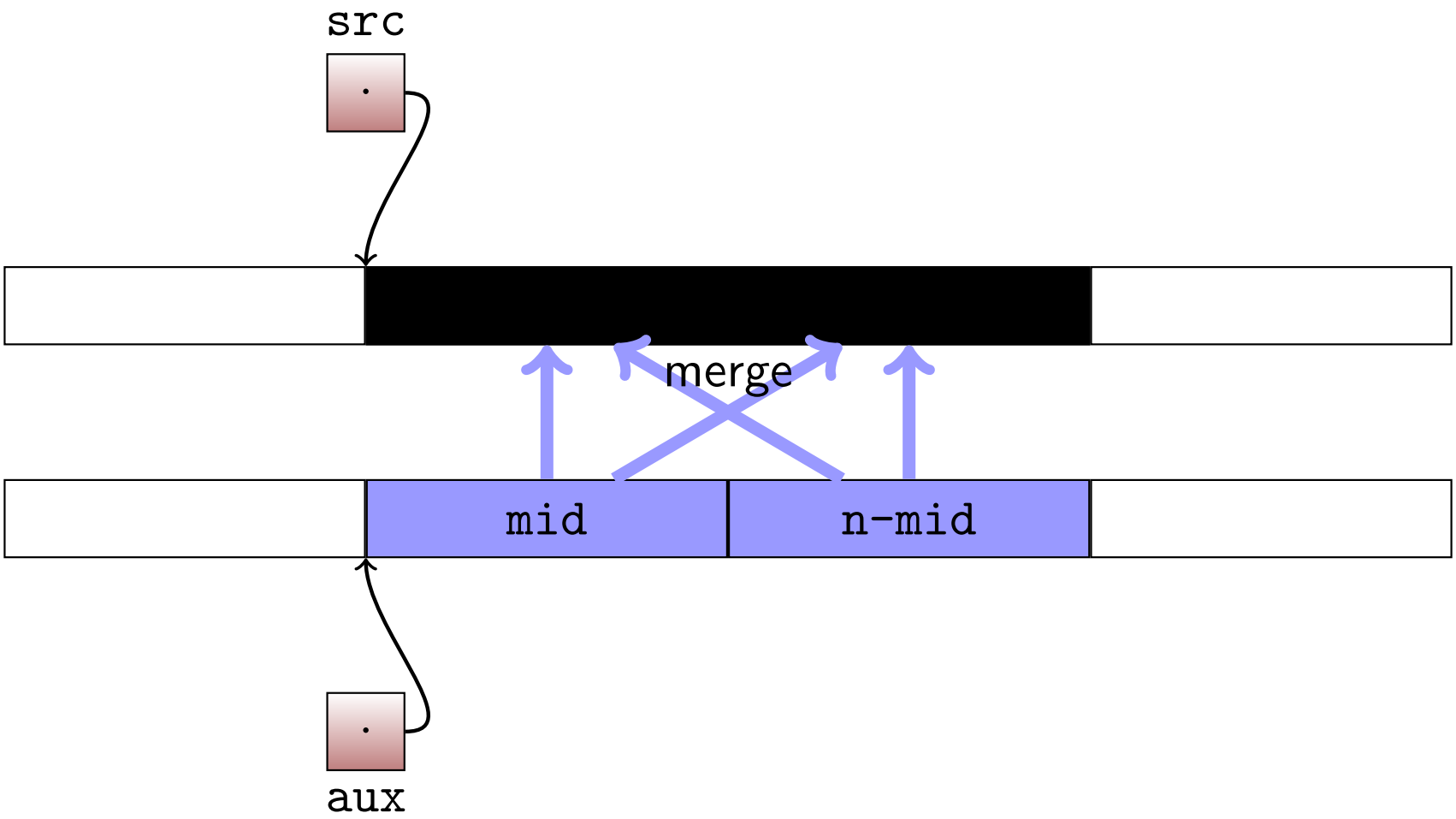
Merge-sorting array using an auxilary array



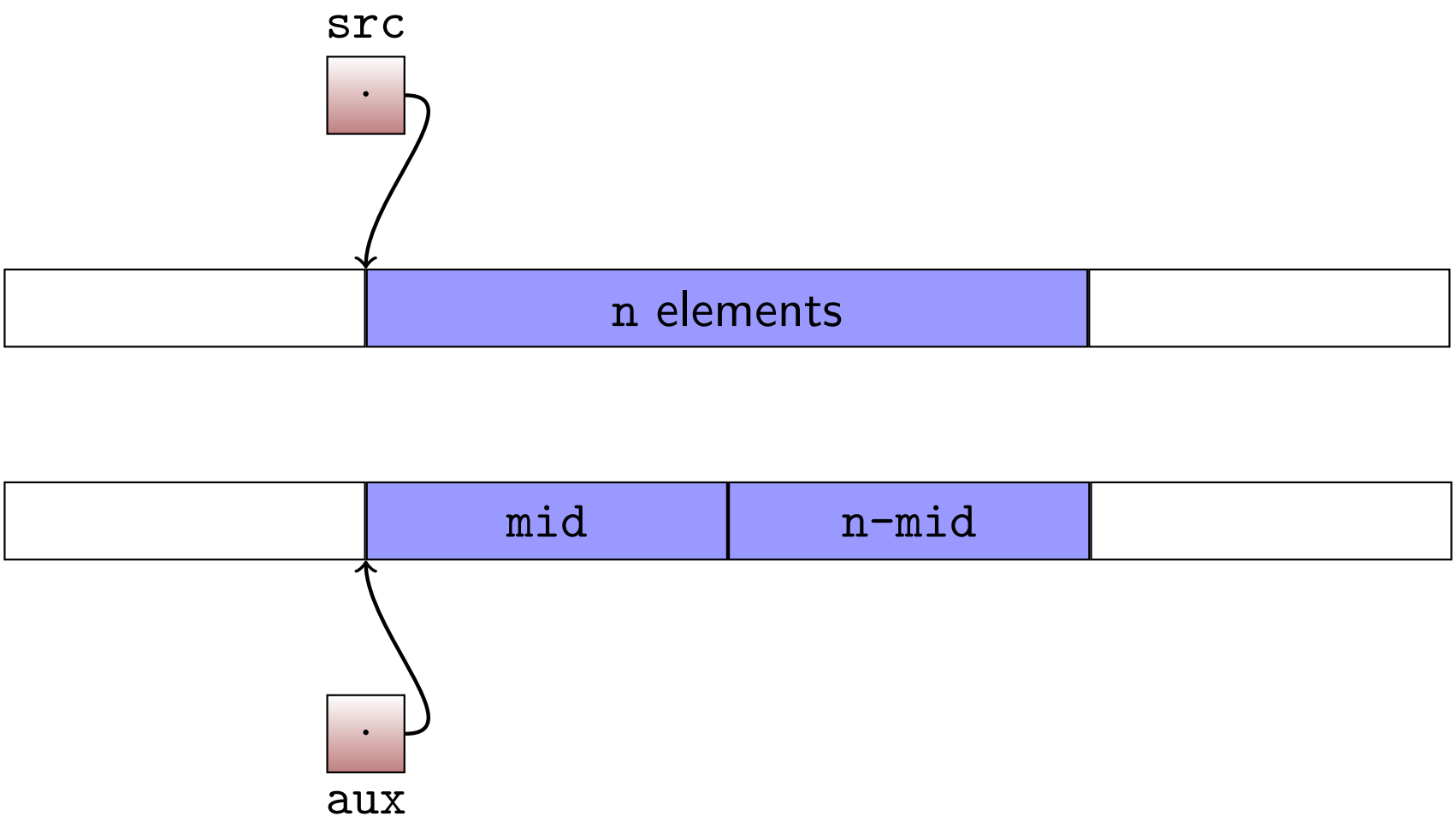
Merge-sorting array using an auxilary array



Merge-sorting array using an auxilary array



Merge-sorting array using an auxilary array



Merge sorting in C

```
void mergeSort( int *ptr, int n, int *aux)
{
  if(n <= 1) return;
  mid = n /2;
  copy(ptr, mid, aux);
  copy(ptr + mid, n-mid, aux + mid);
  mergeSort(aux,mid, ptr);
  mergeSort(aux + mid, n - mid, ptr + mid);
  merge(aux, mid, n, a);
}
```