Functional Programming III

Spring 2014

Carola Wenk

Merge Sort

Suppose that we know how to merge two sorted lists. Then, we can sort recursively:

Merge Sort:

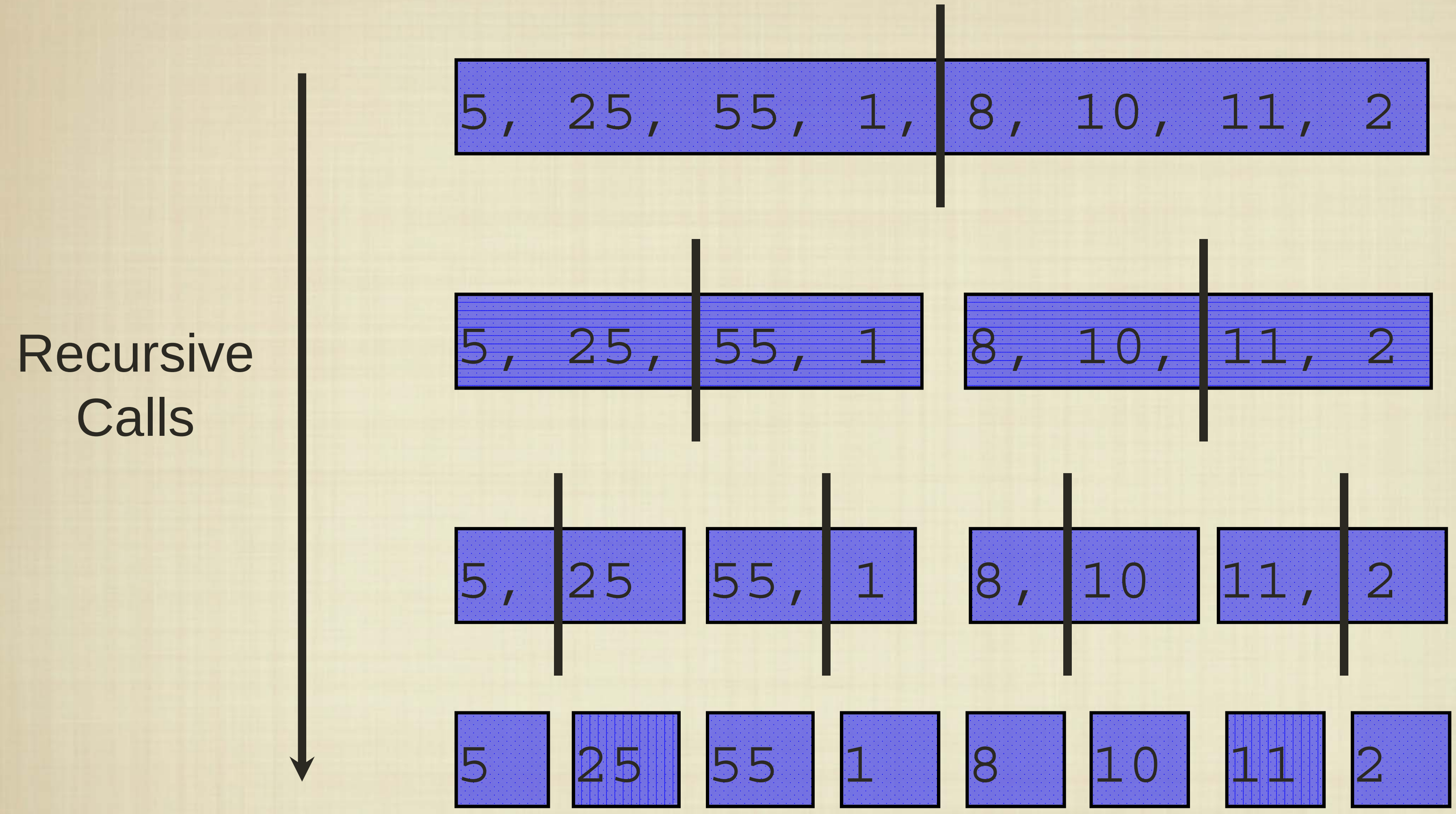
- 1. Split the given list into two equal parts.
- 2. Recursively sort each half.
- 3. Merge the sorted halves and return the result.

Merge Sort (Python)

<ins>Merge Sort:</ins>
1. Split the given list into two equal parts.
2. Recursively sort each half.
3. Merge the sorted halves and return the result.

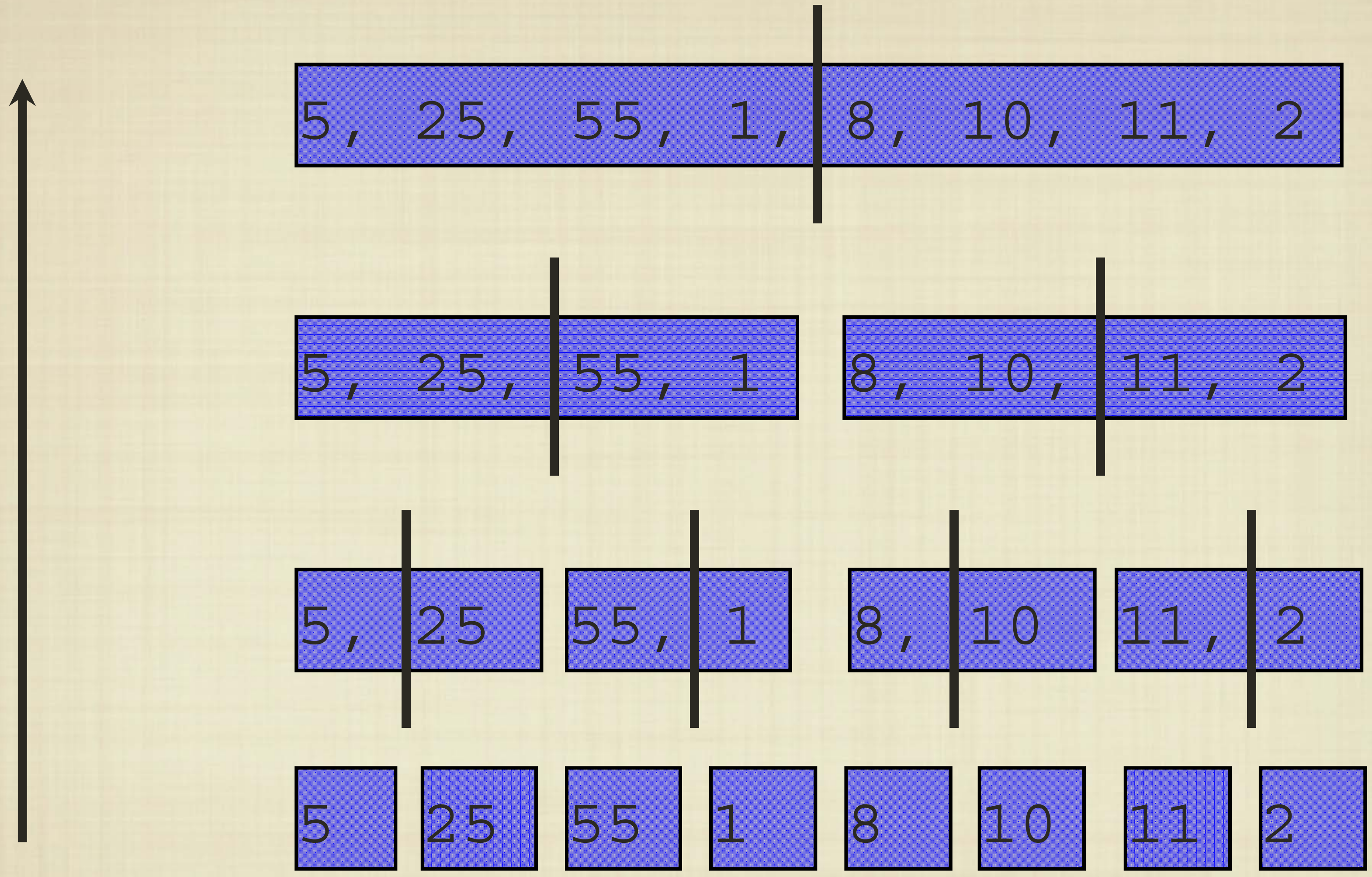
```
def merge_sort (L):
    n = len(L)
    #base case:
    if n<=1:
        return L
    #recursive case: Recursively sort each half
    A = merge_sort(L[:n/2])   # left half, L[0..n/2-1]
    B = merge_sort(L[n/2:])   # right half, L[n/2..n-1]
    # merge sorted halves:
    return merge(A,B)
```

Merge Sort



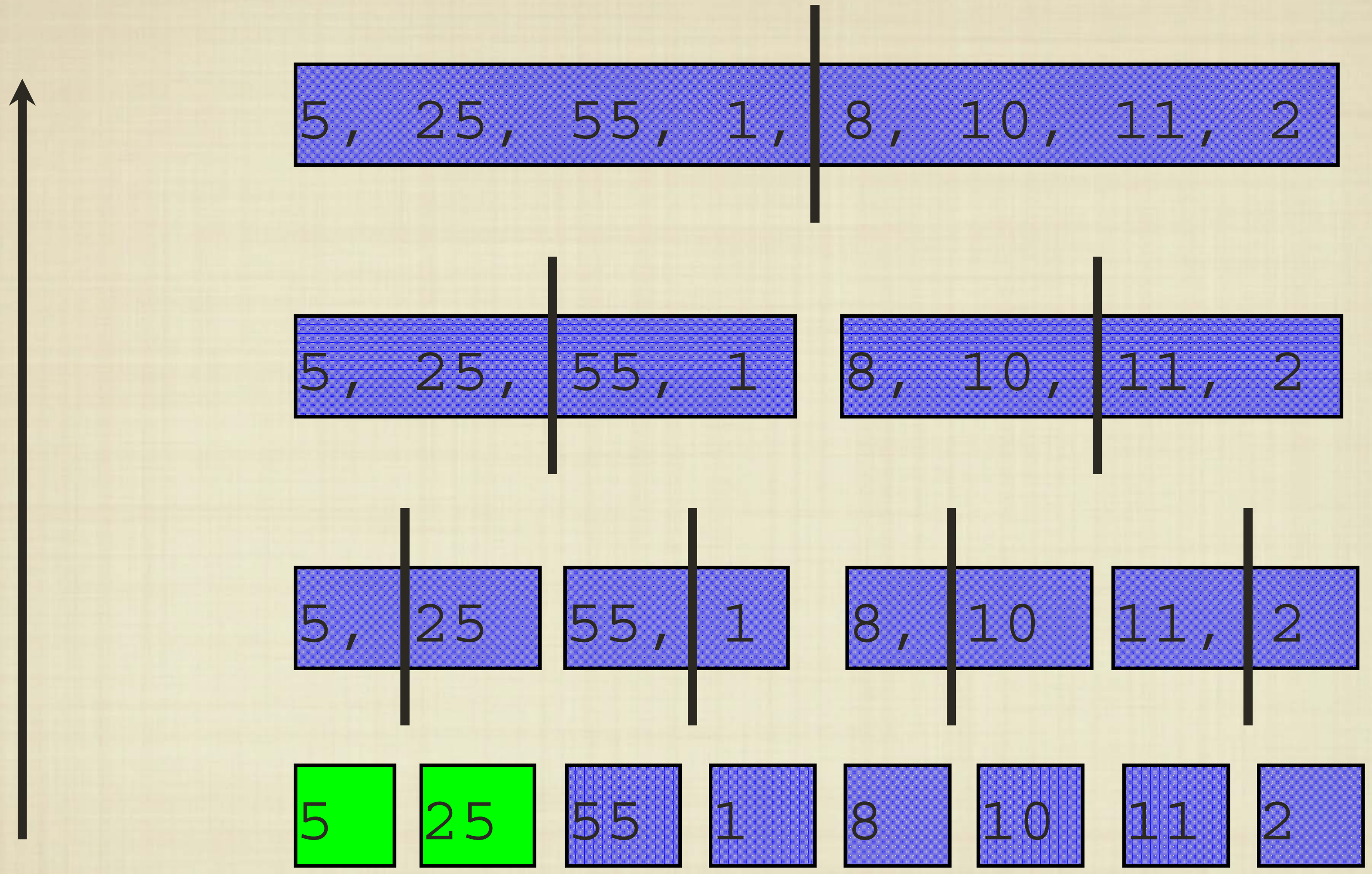
Actually, not a lot is happening in the recursive calls. So where is the sorting happening?

Merge Sort



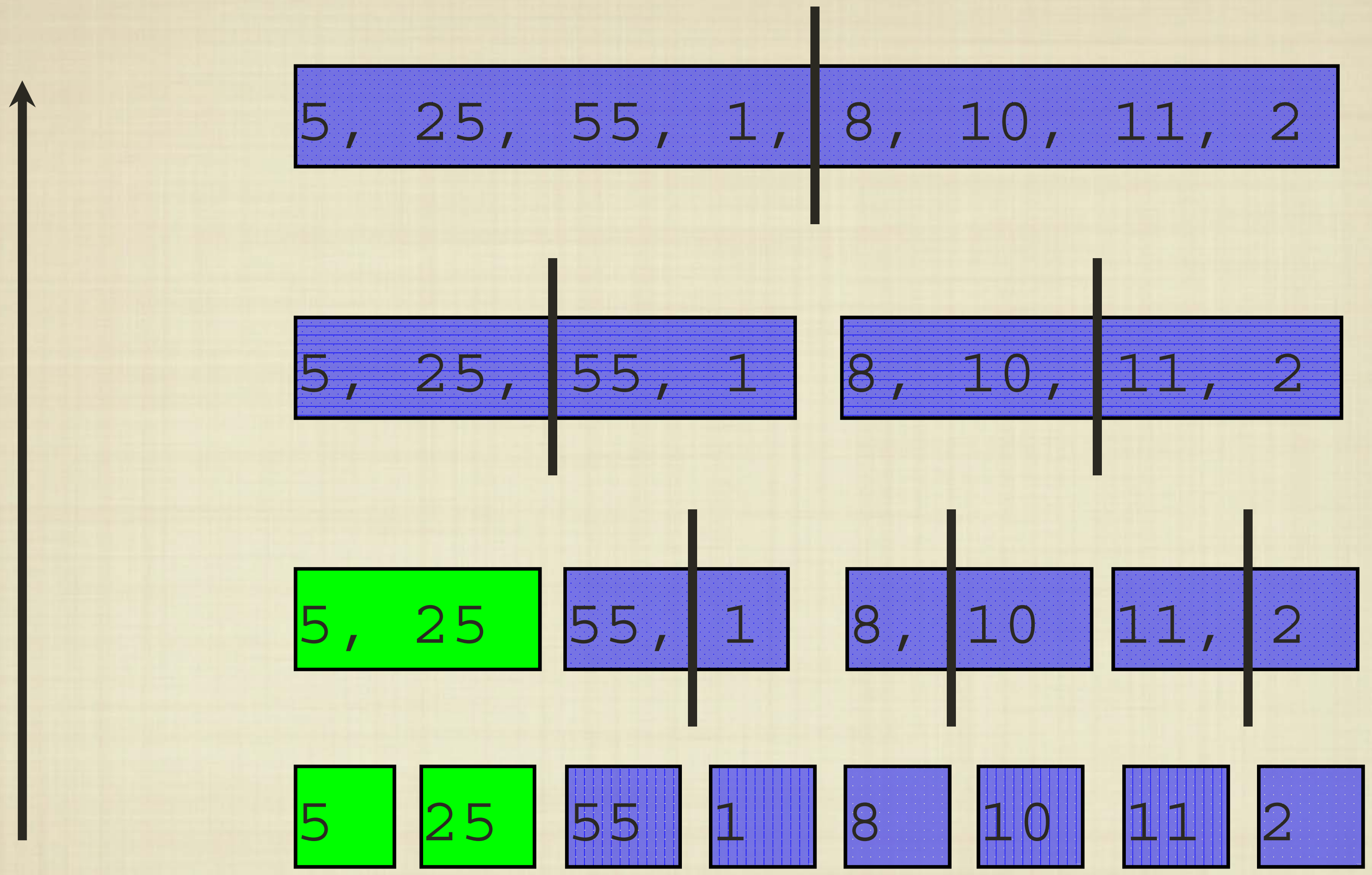
The merge step is actually doing all of the work!

Merge Sort



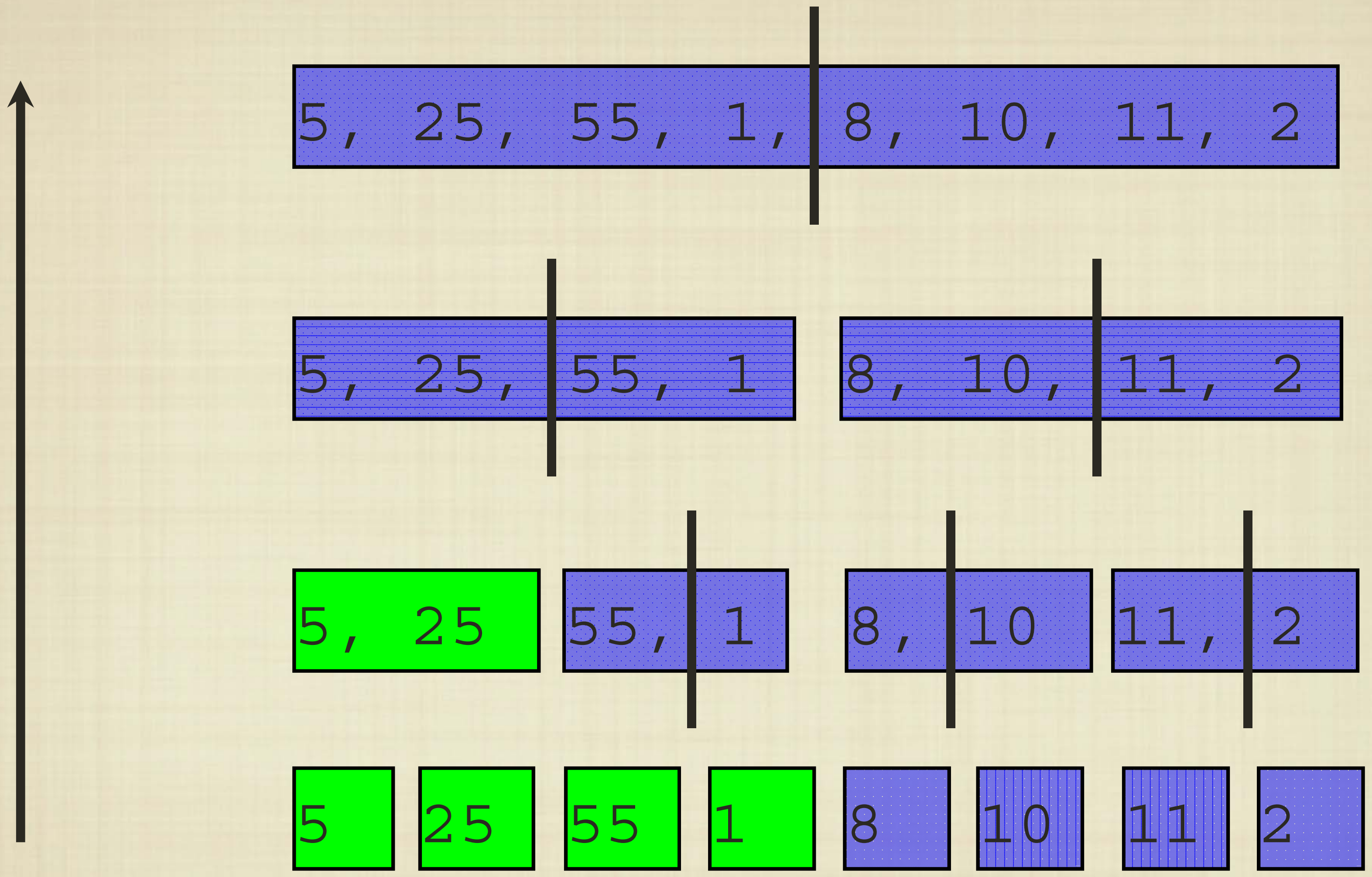
The merge step is actually doing all of the work!

Merge Sort



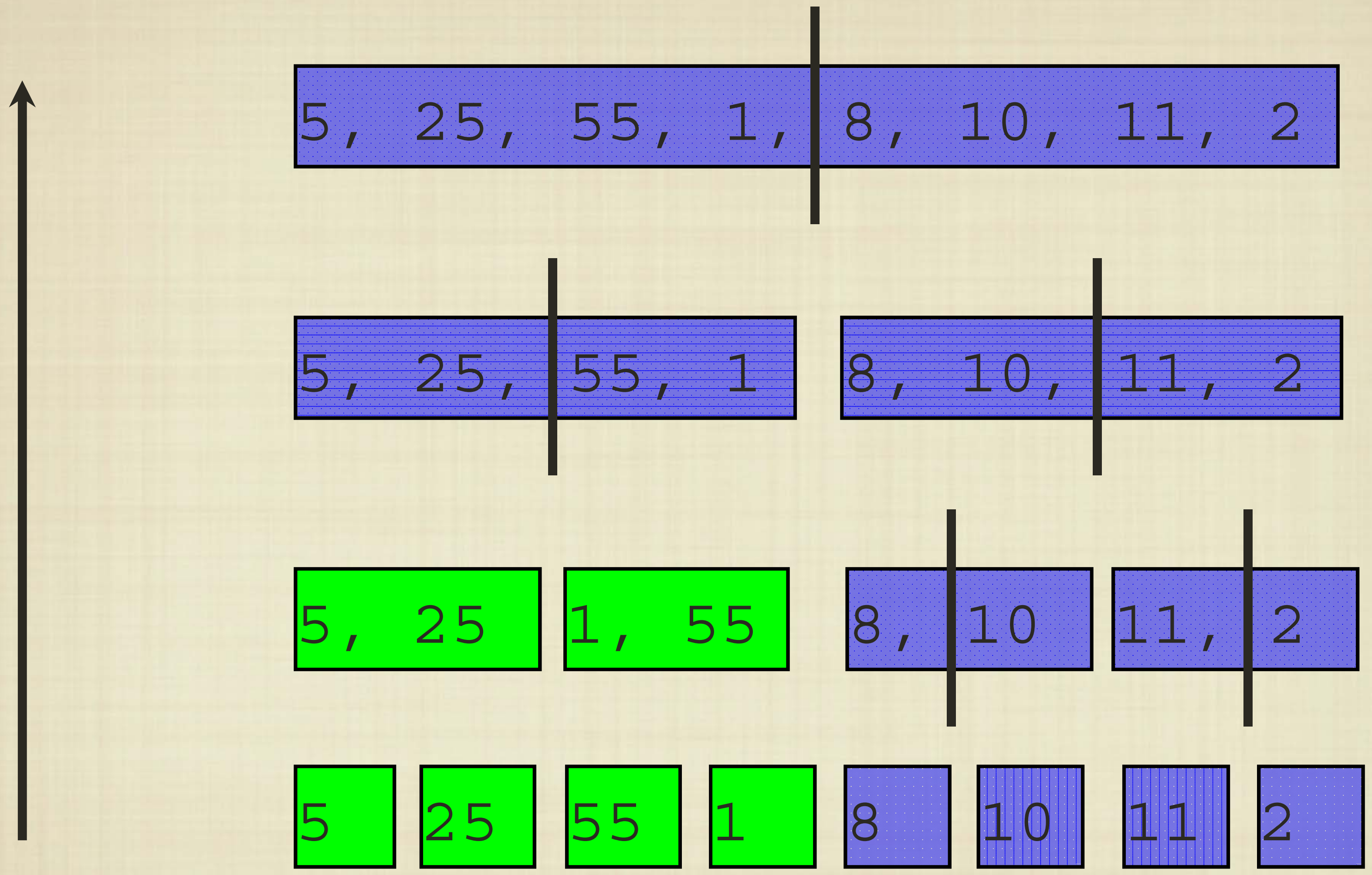
The merge step is actually doing all of the work!

Merge Sort



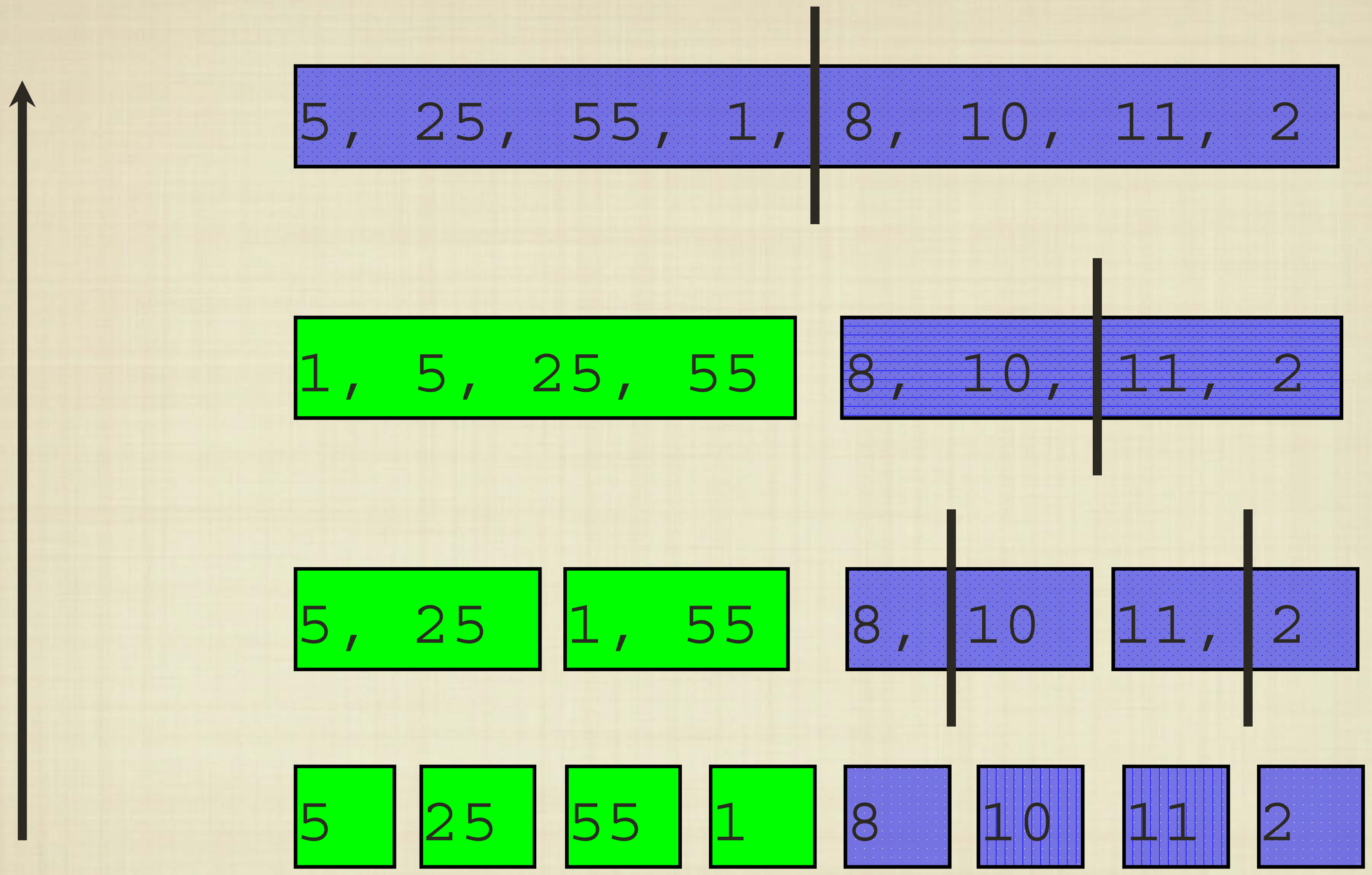
The merge step is actually doing all of the work!

Merge Sort



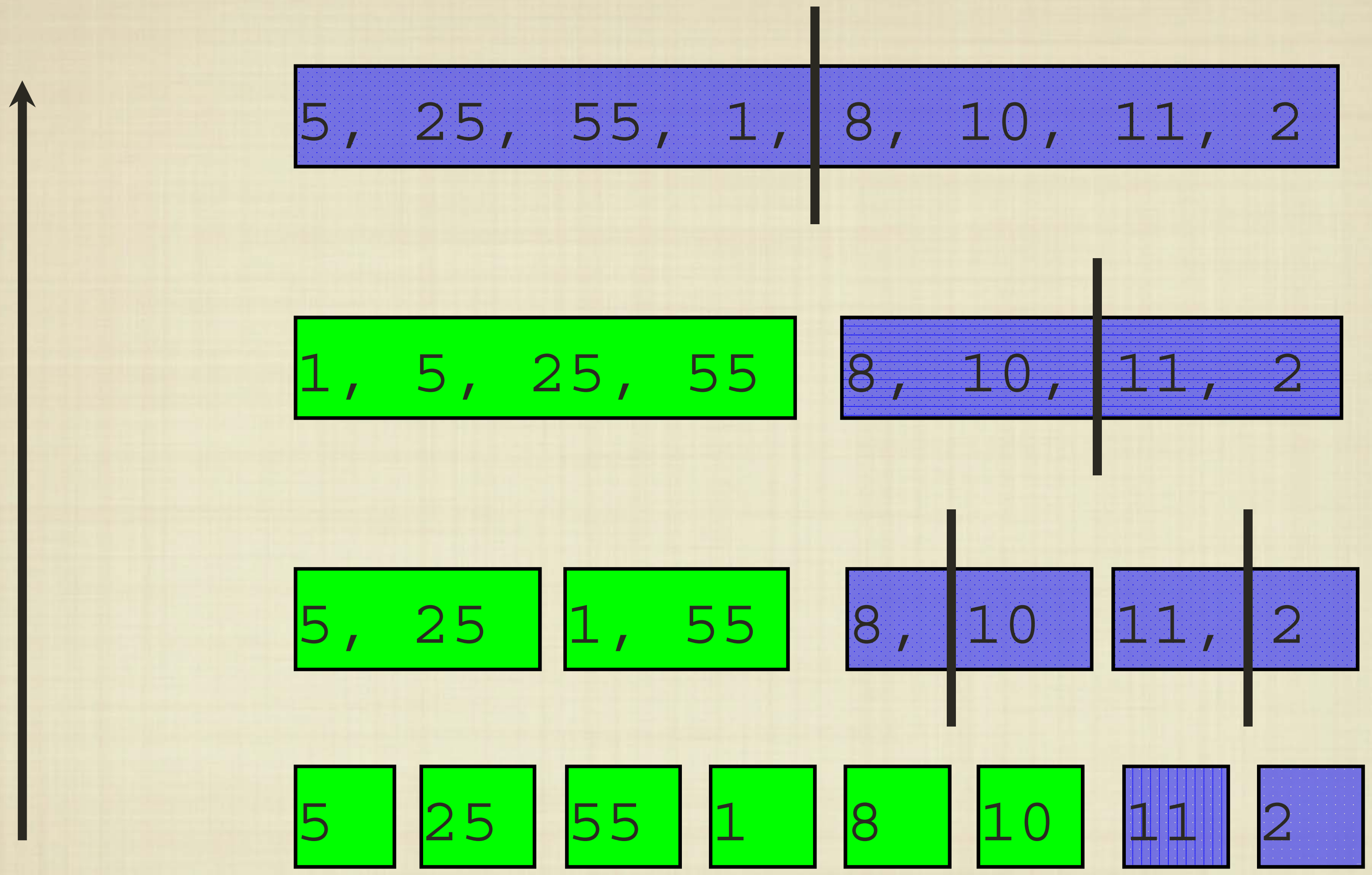
The merge step is actually doing all of the work!

Merge Sort



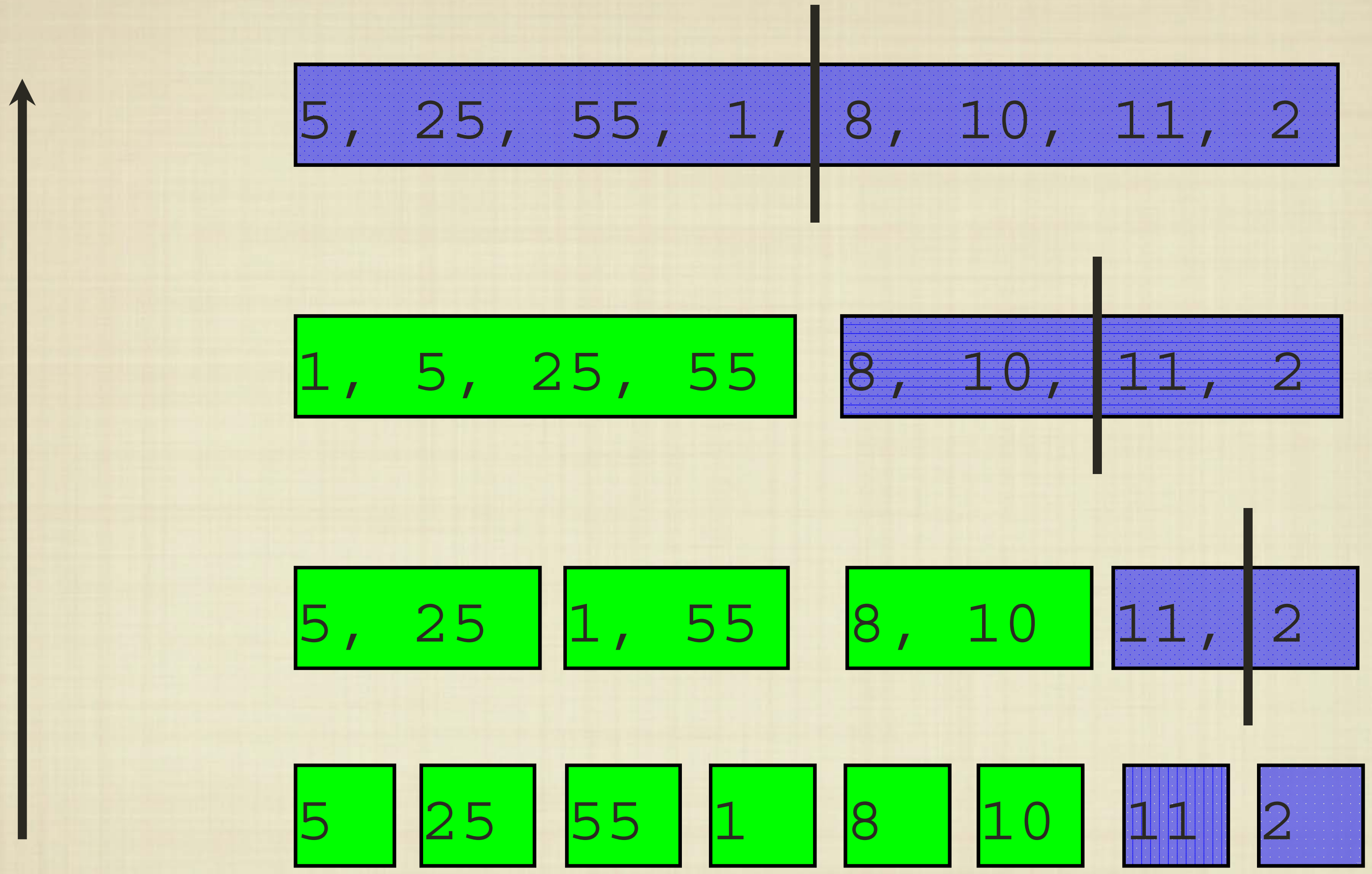
The merge step is actually doing all of the work!

Merge Sort



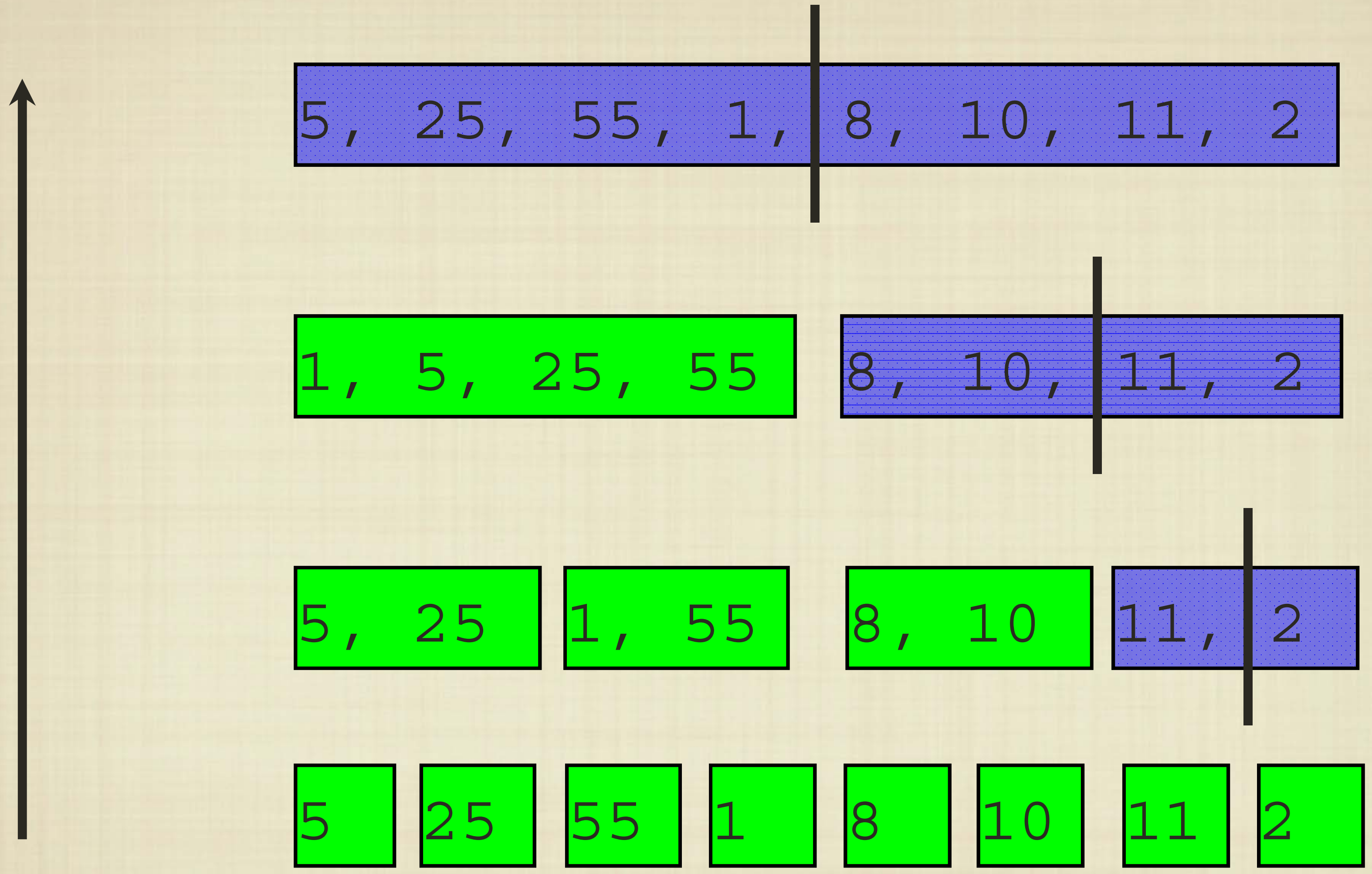
The merge step is actually doing all of the work!

Merge Sort



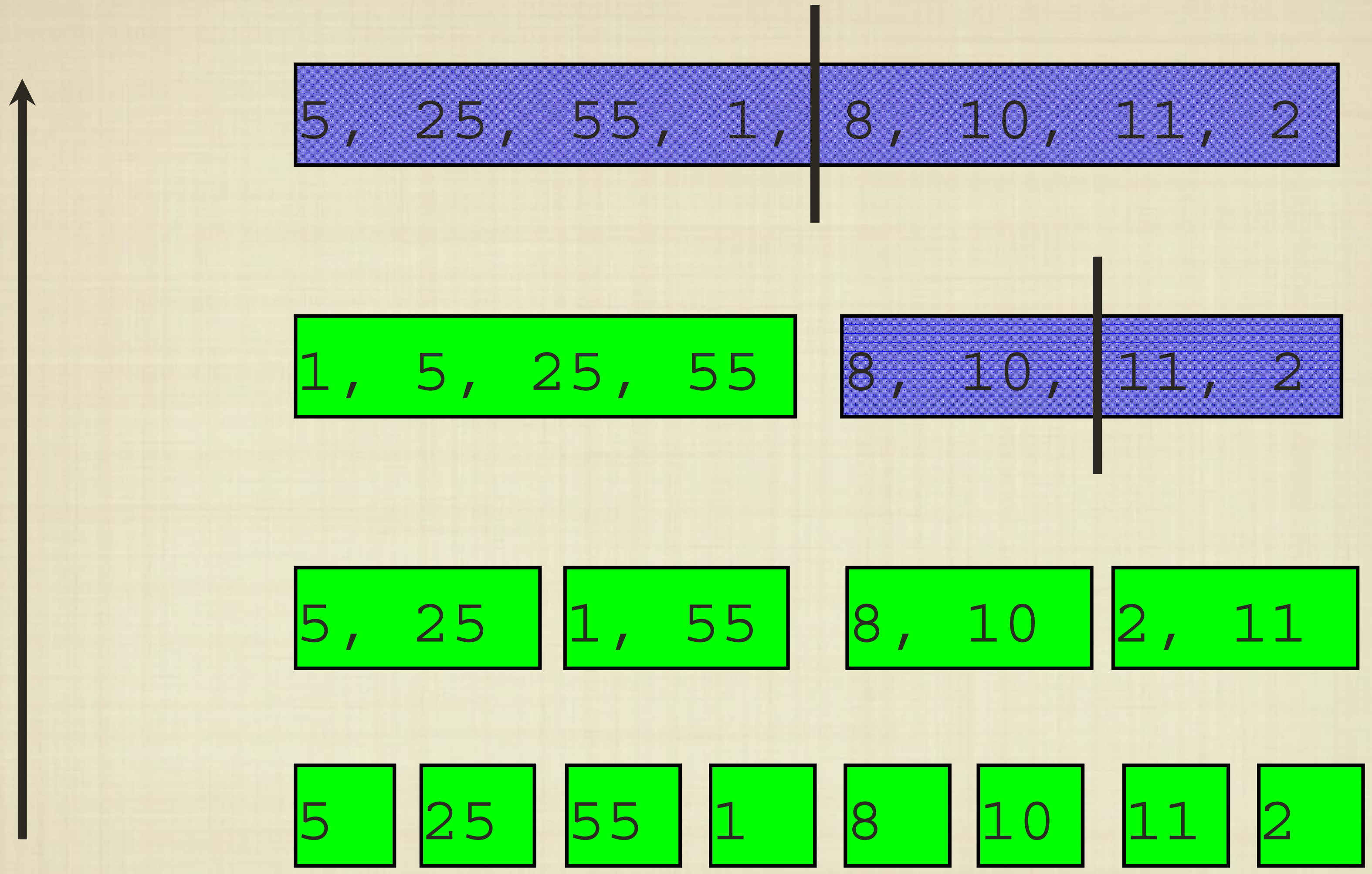
The merge step is actually doing all of the work!

Merge Sort



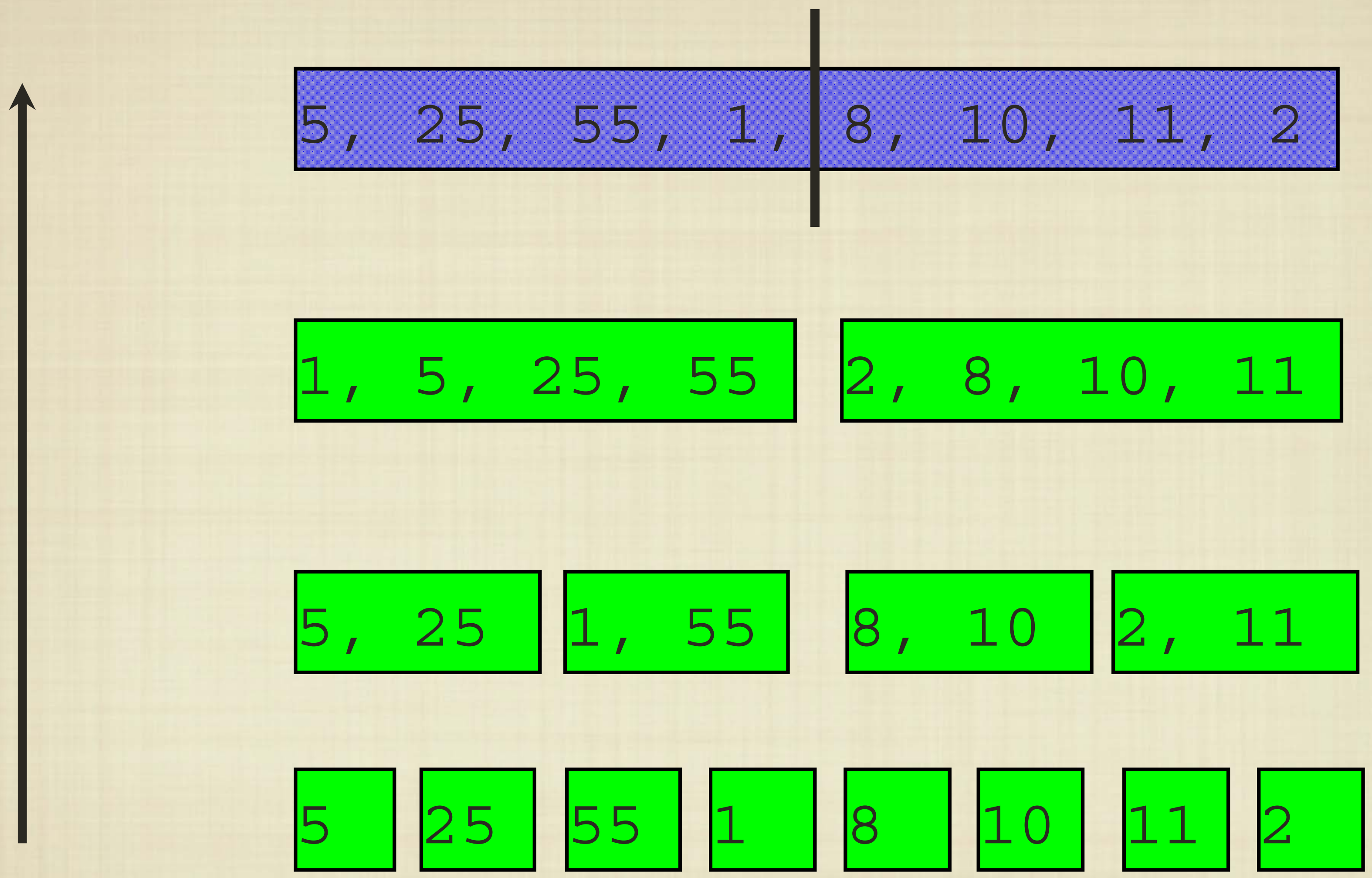
The merge step is actually doing all of the work!

Merge Sort



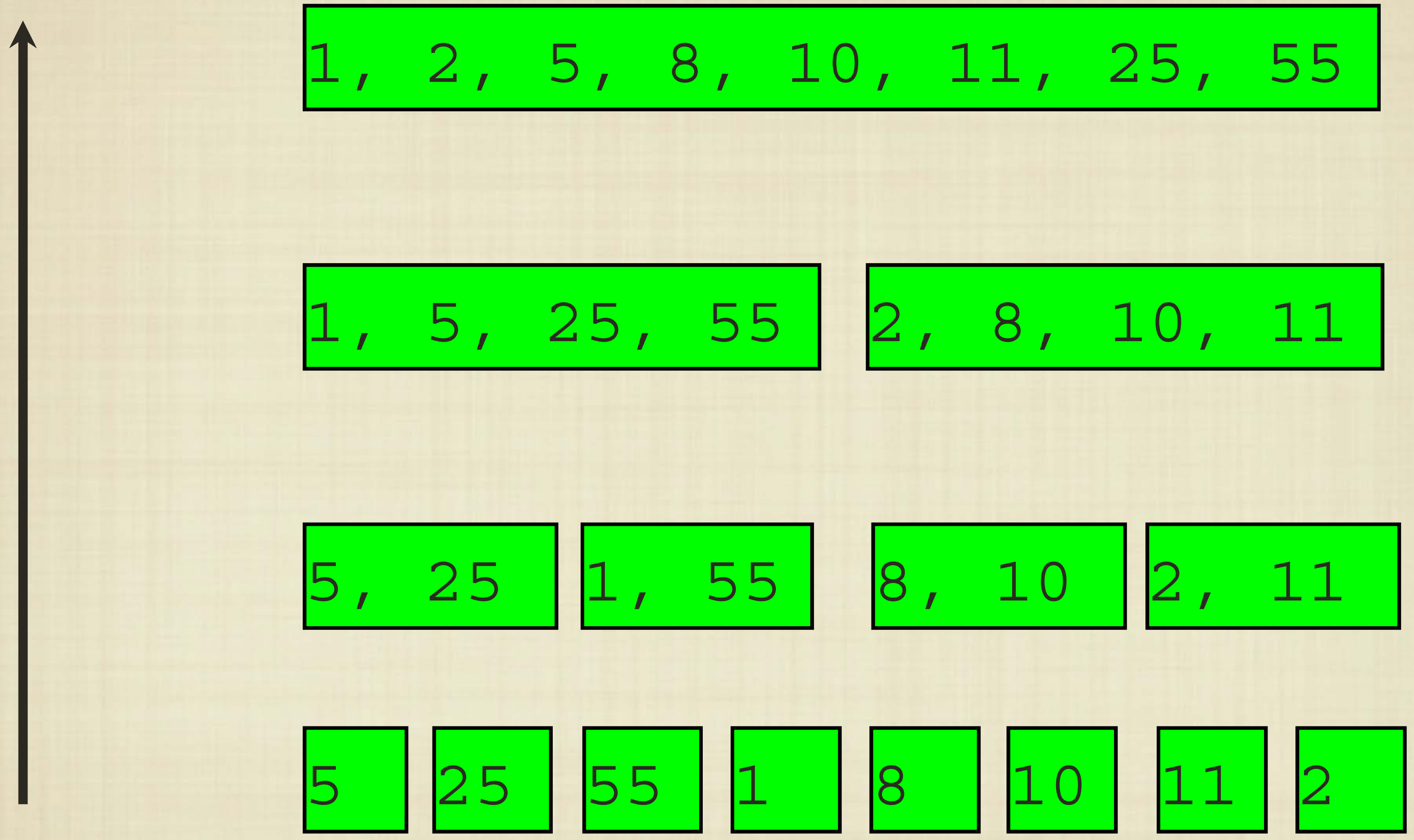
The merge step is actually doing all of the work!

Merge Sort



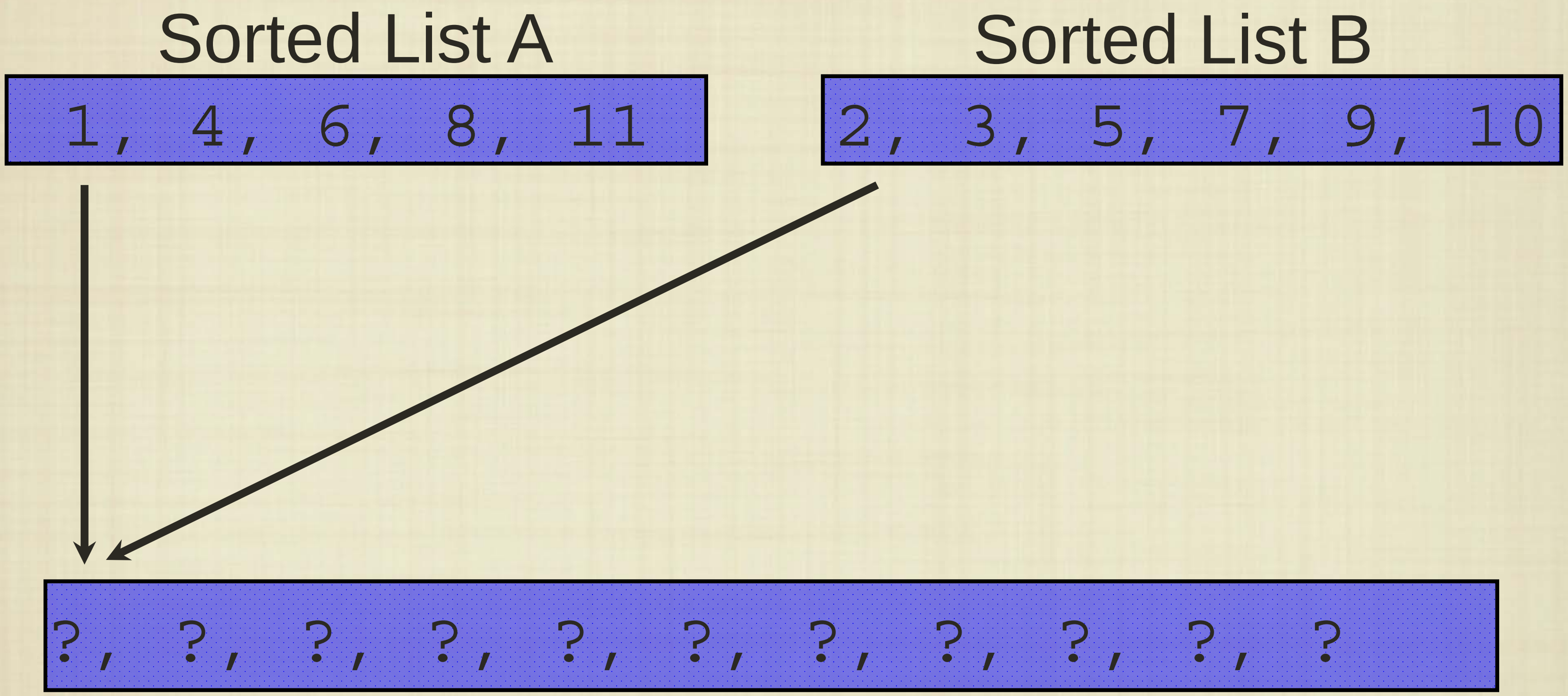
The merge step is actually doing all of the work!

Merge Sort

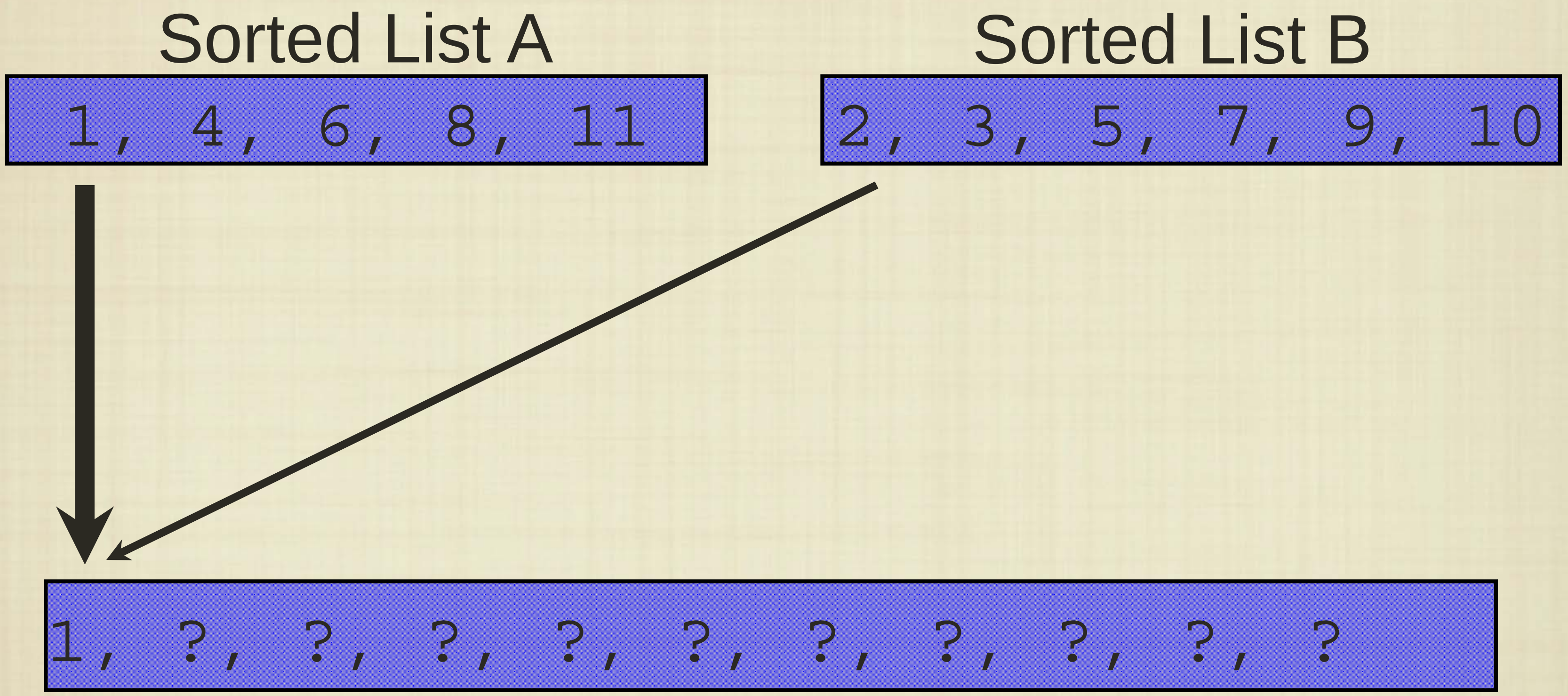


The merge step is actually doing all of the work!

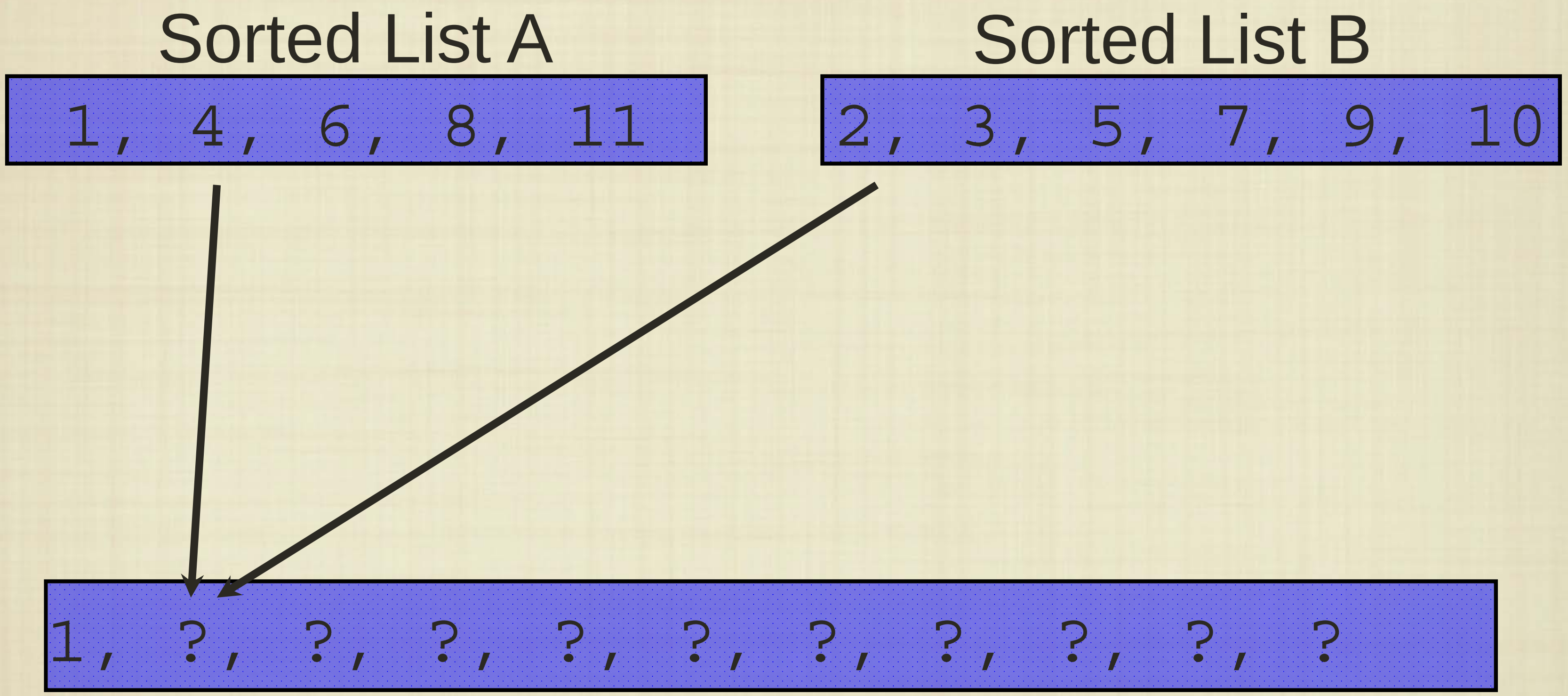
Merging Lists



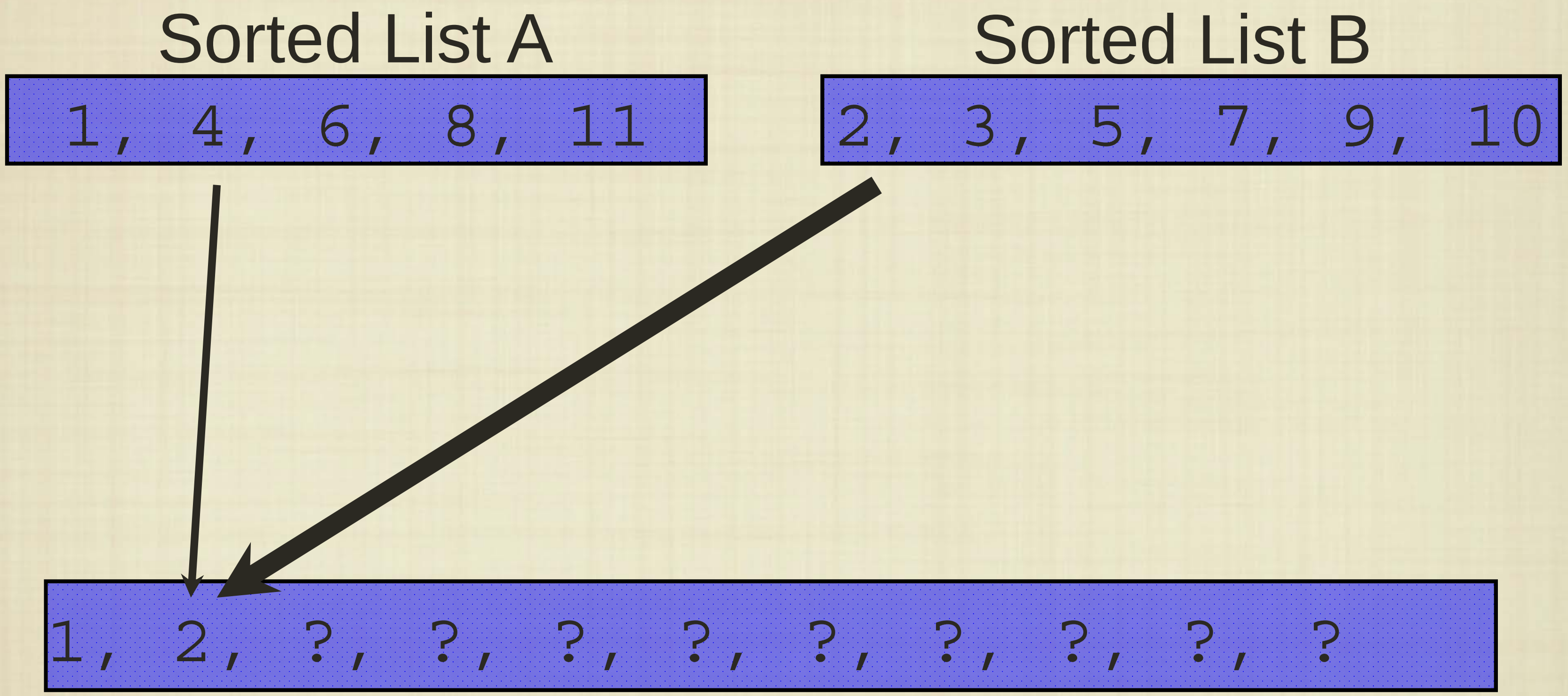
Merging Lists



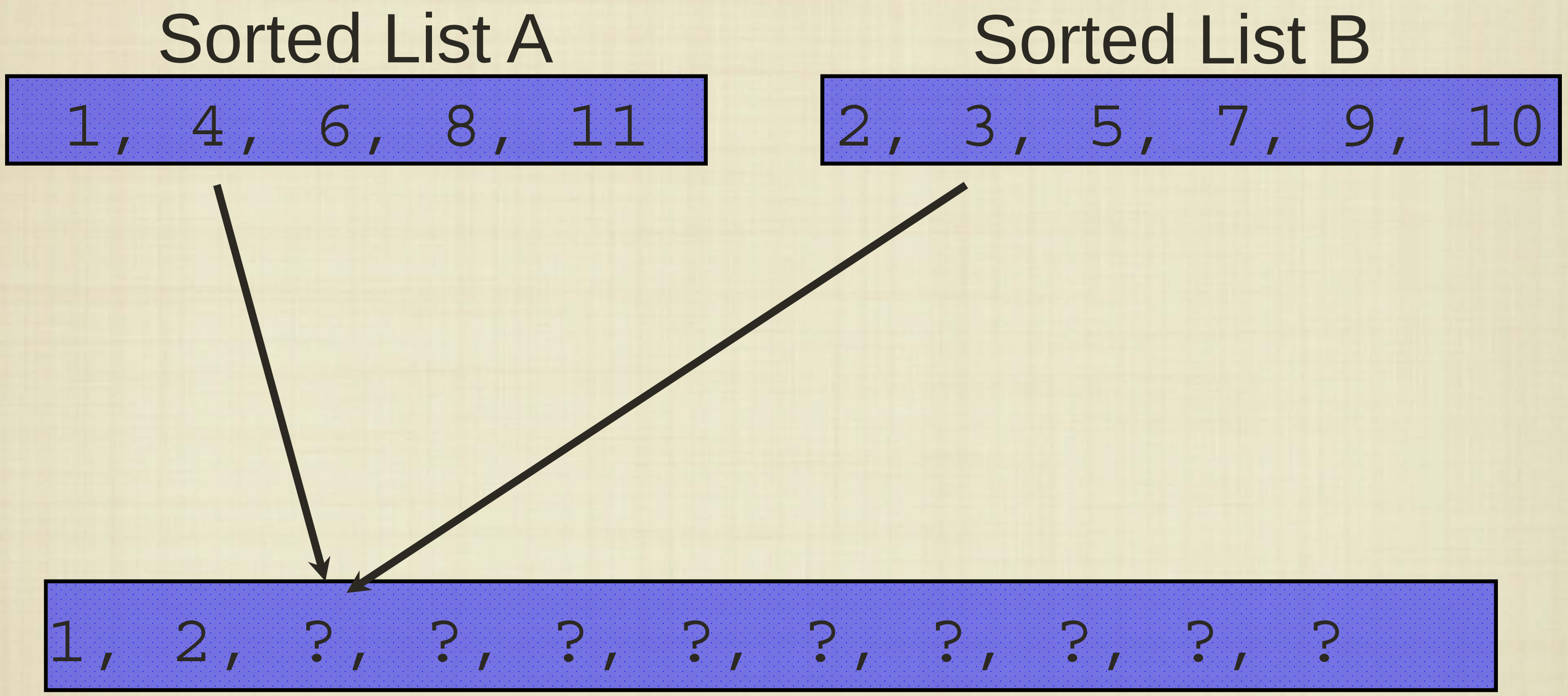
Merging Lists



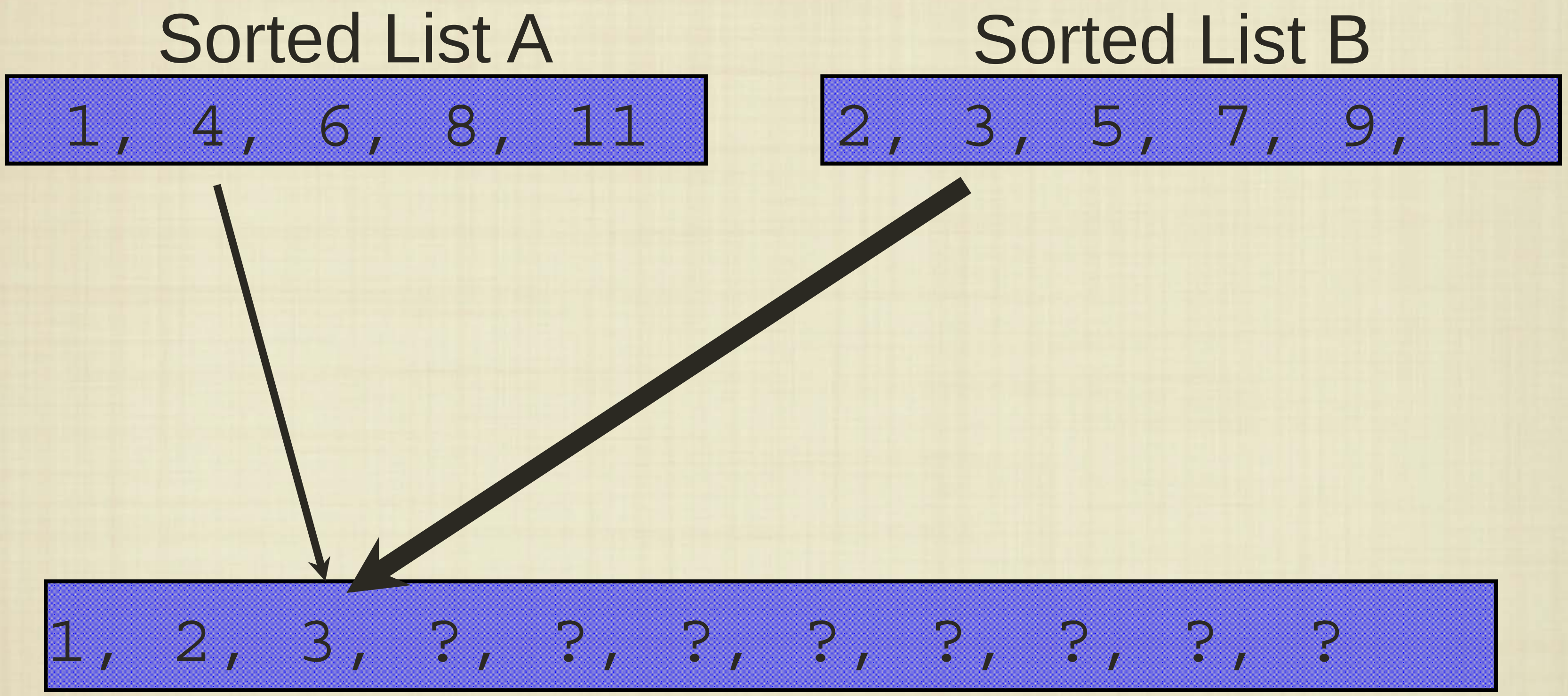
Merging Lists



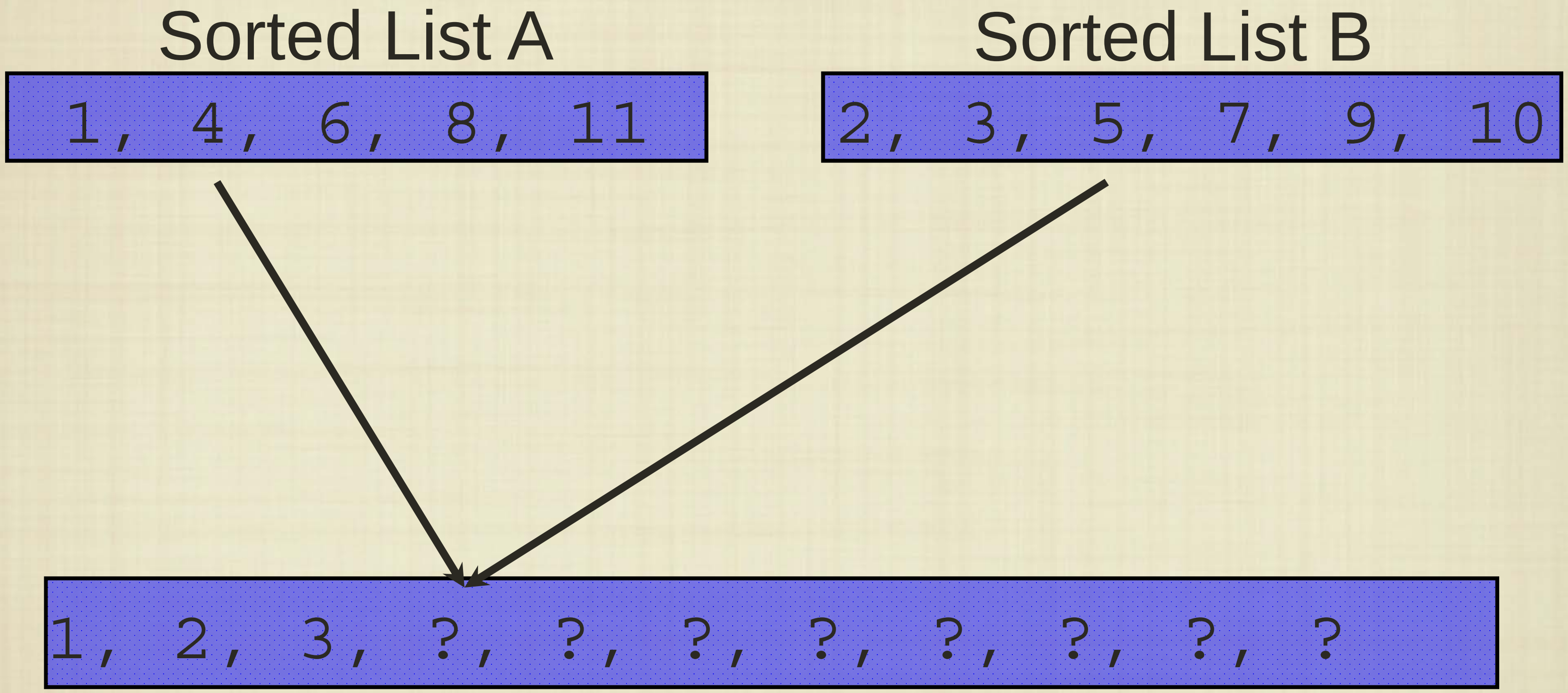
Merging Lists



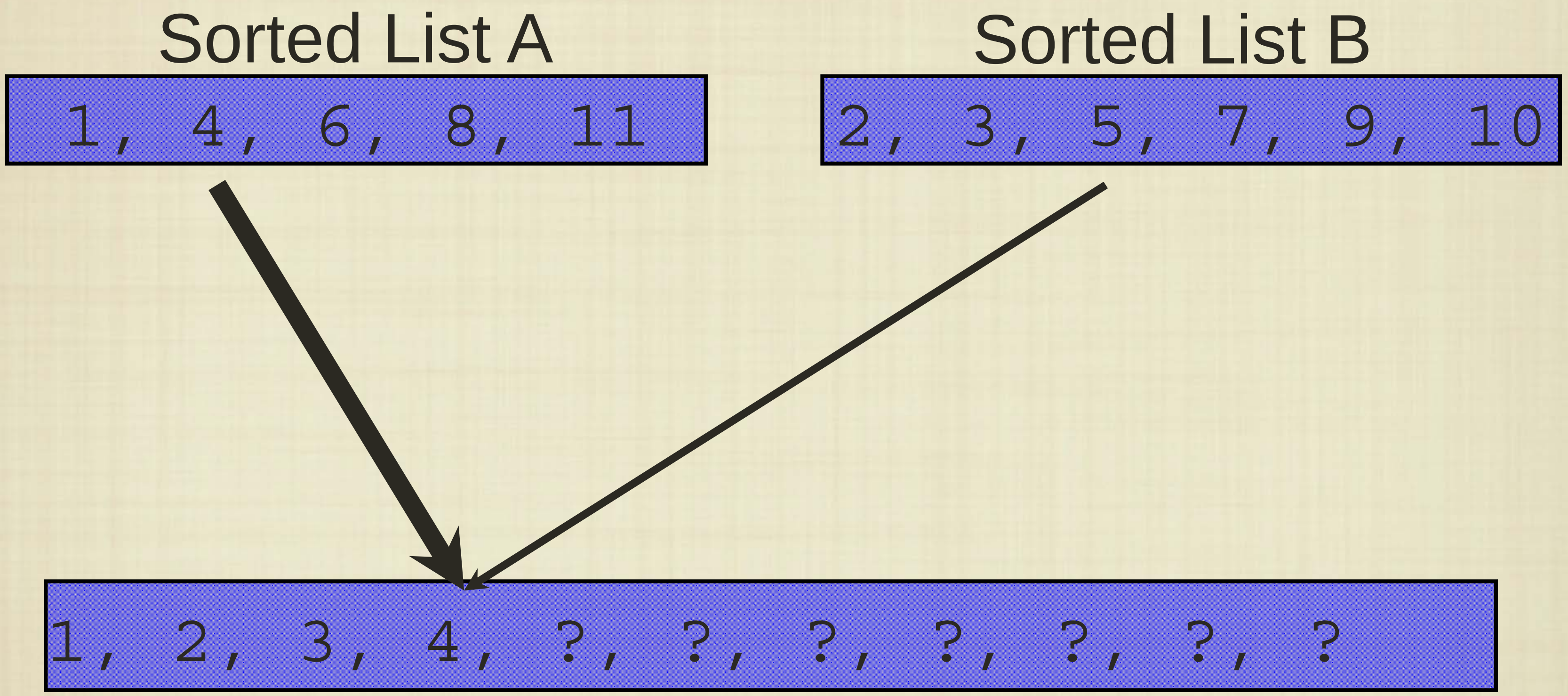
Merging Lists



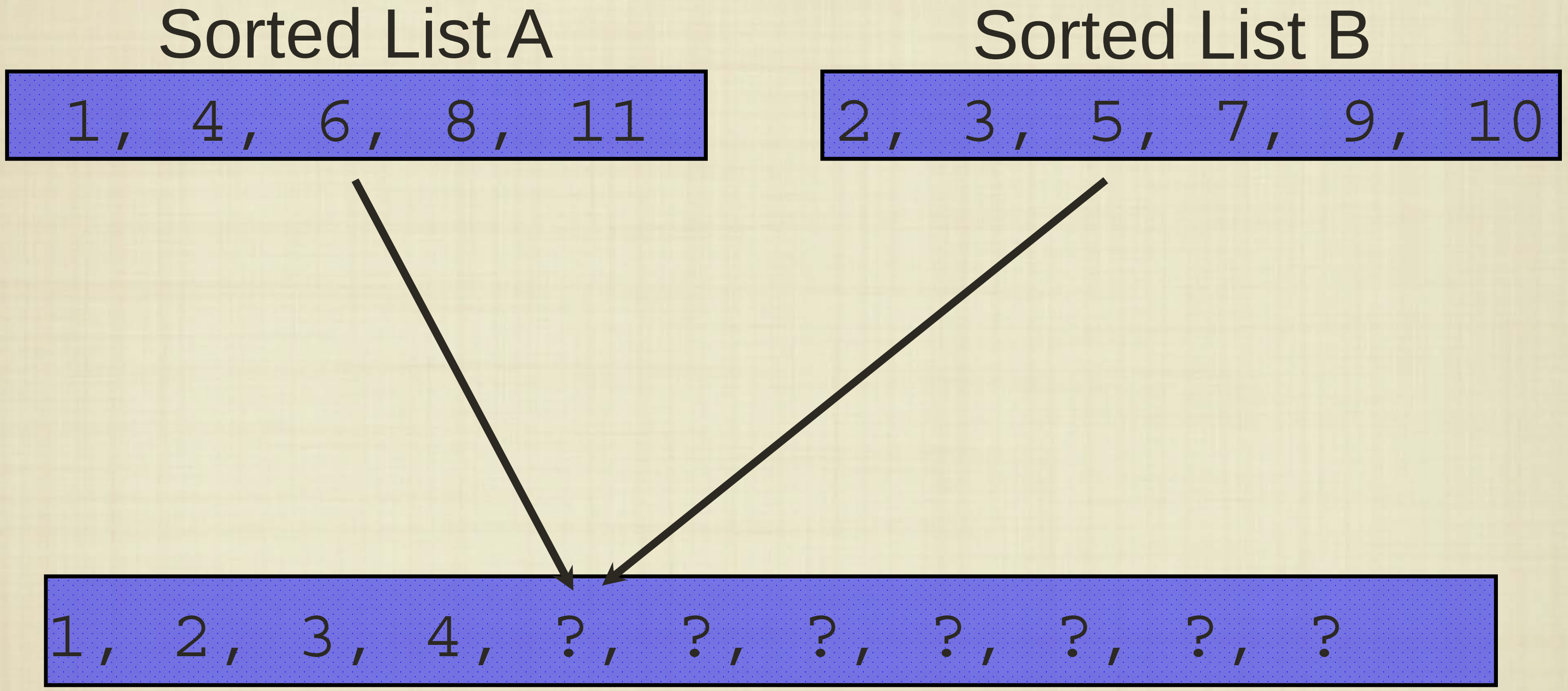
Merging Lists



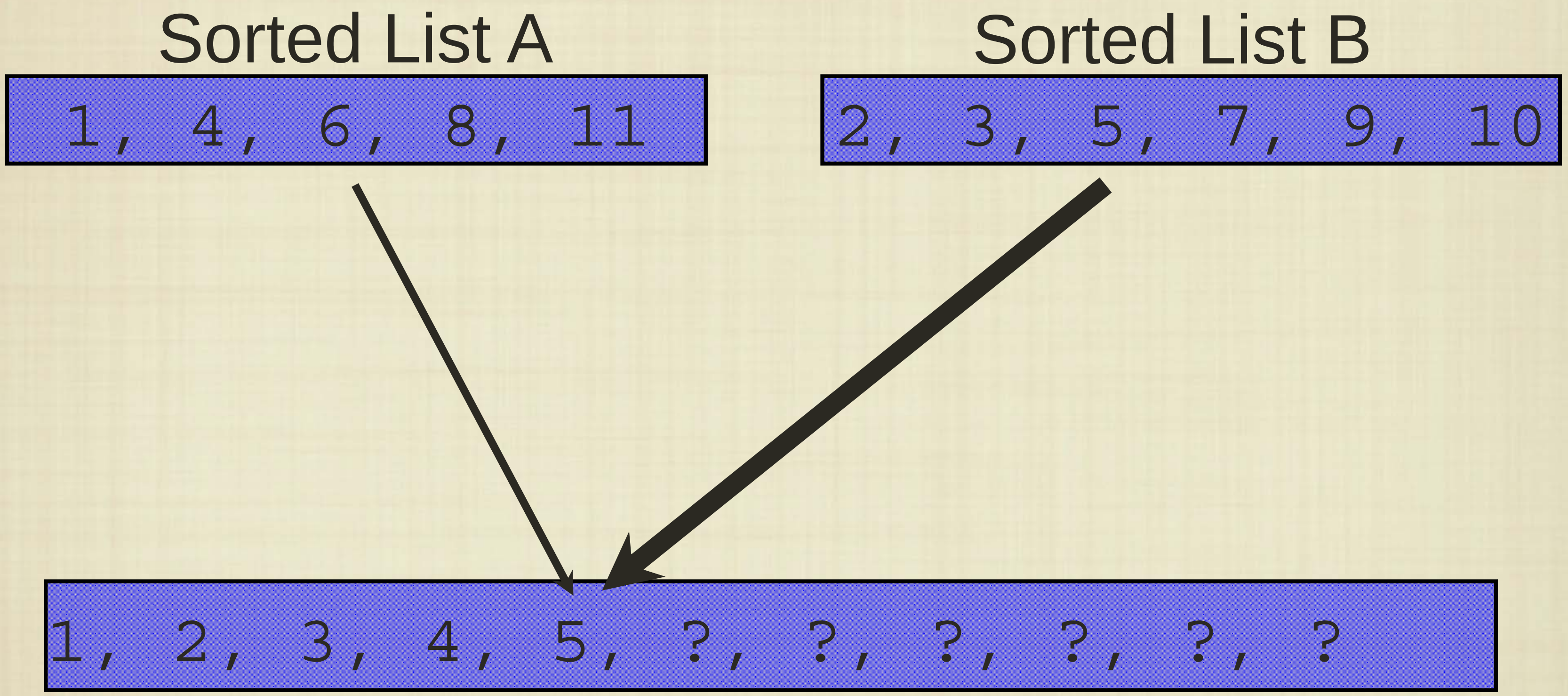
Merging Lists



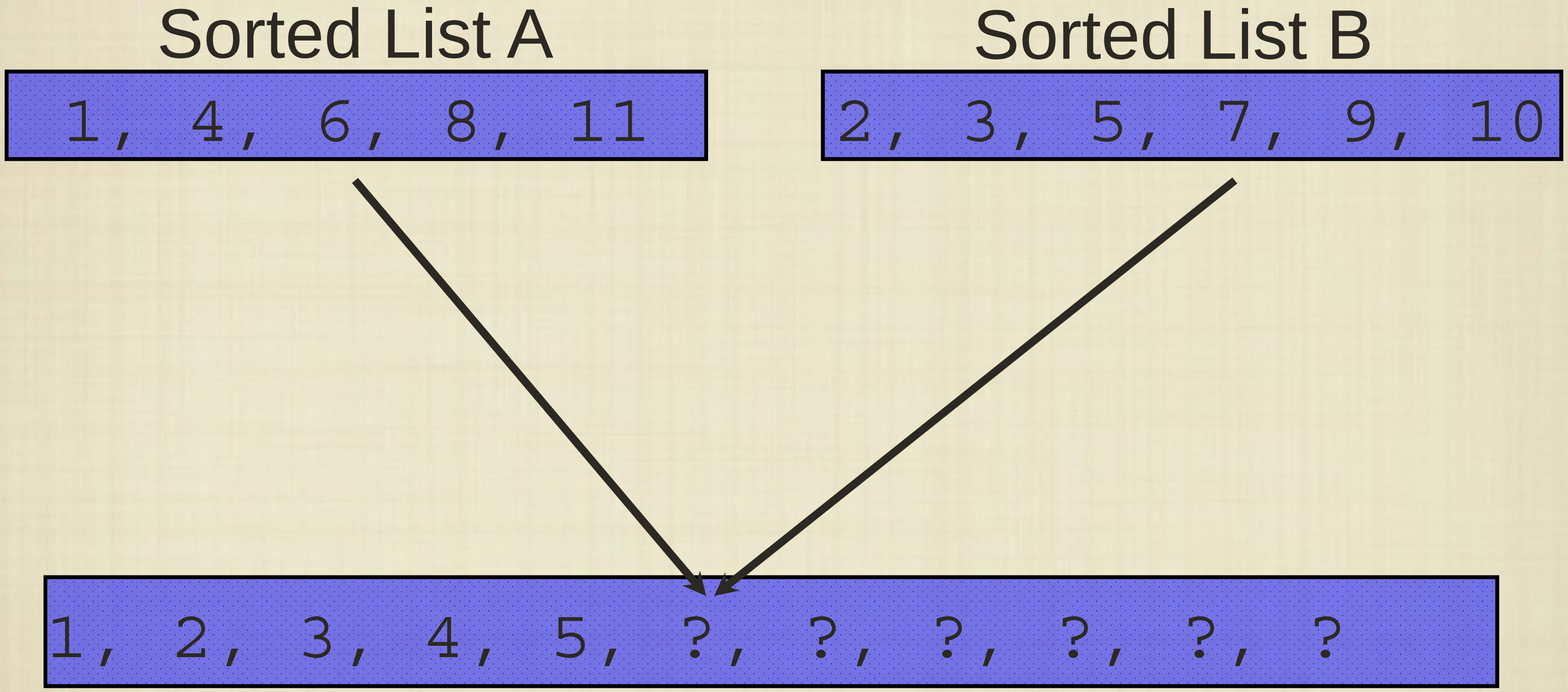
Merging Lists



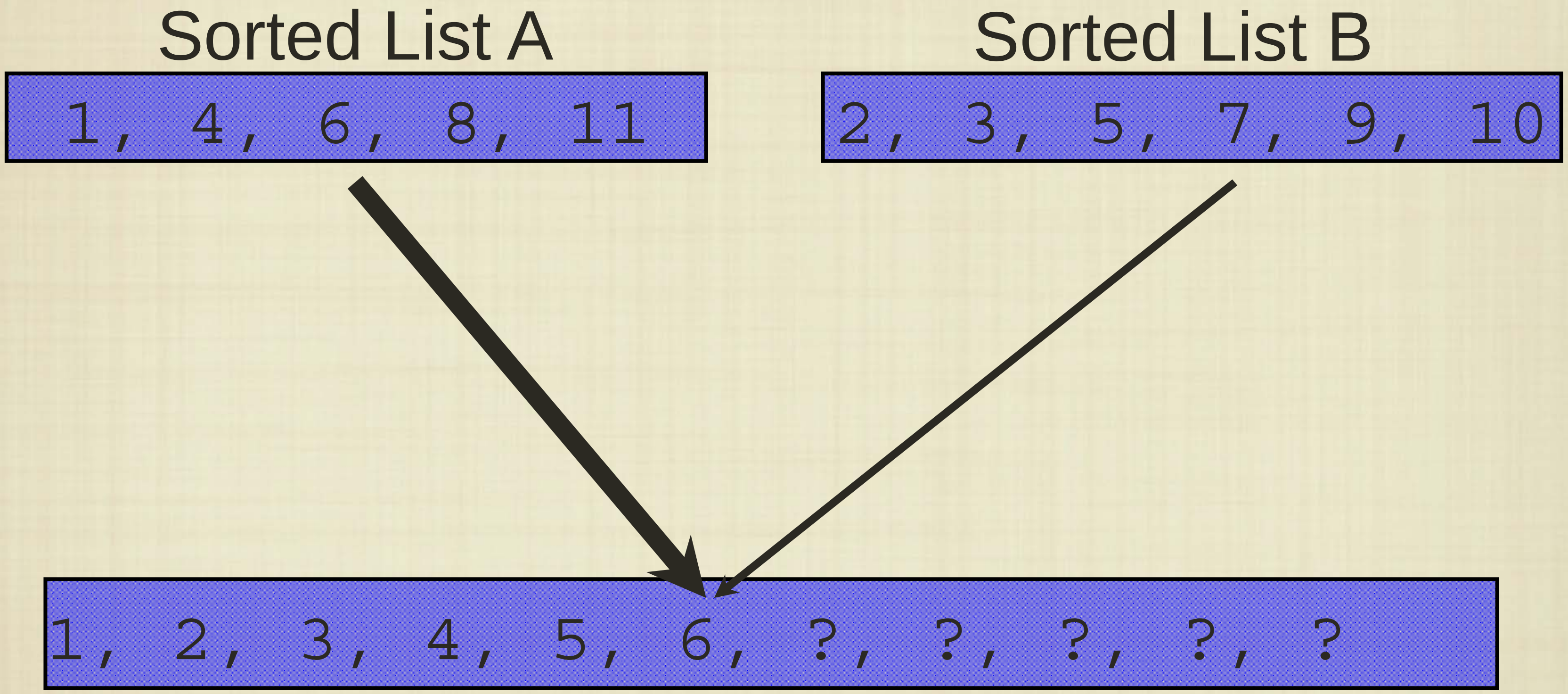
Merging Lists



Merging Lists

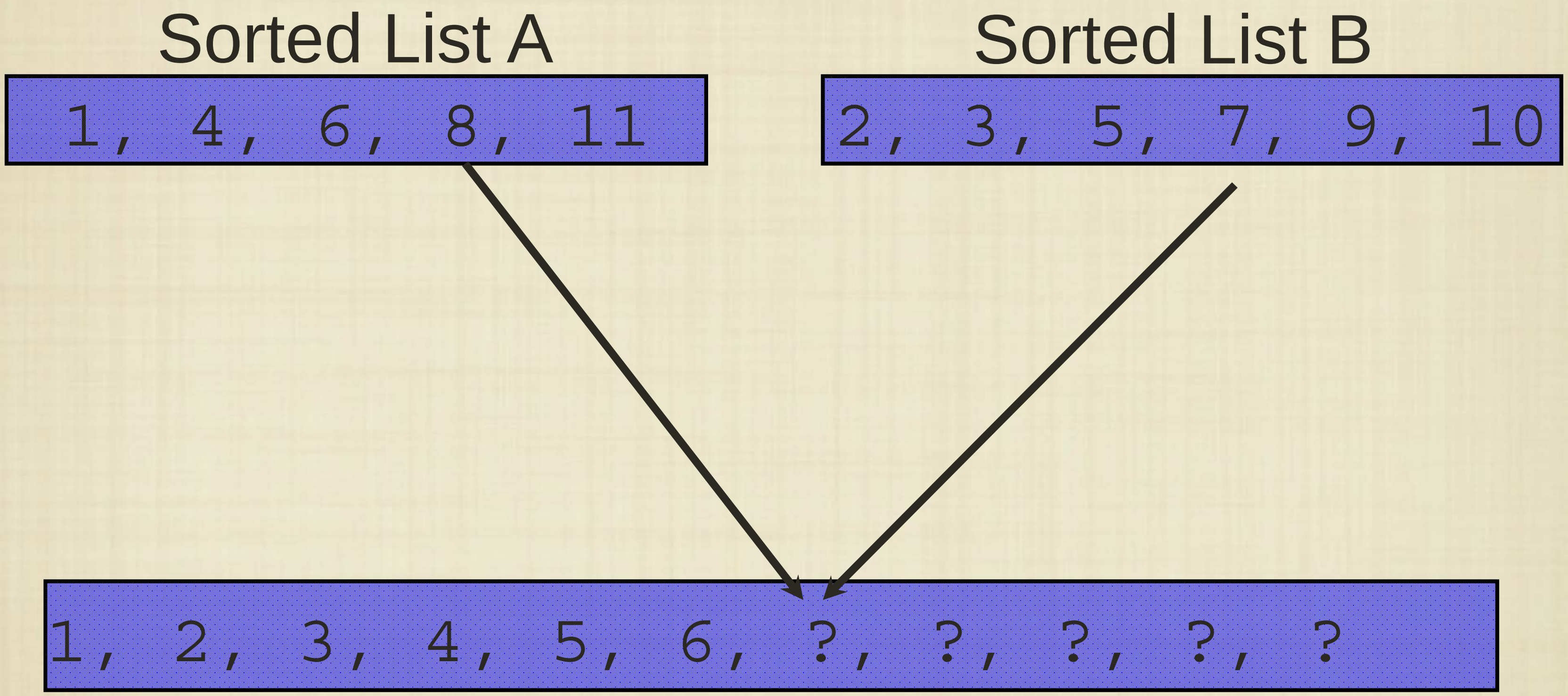


Merging Lists



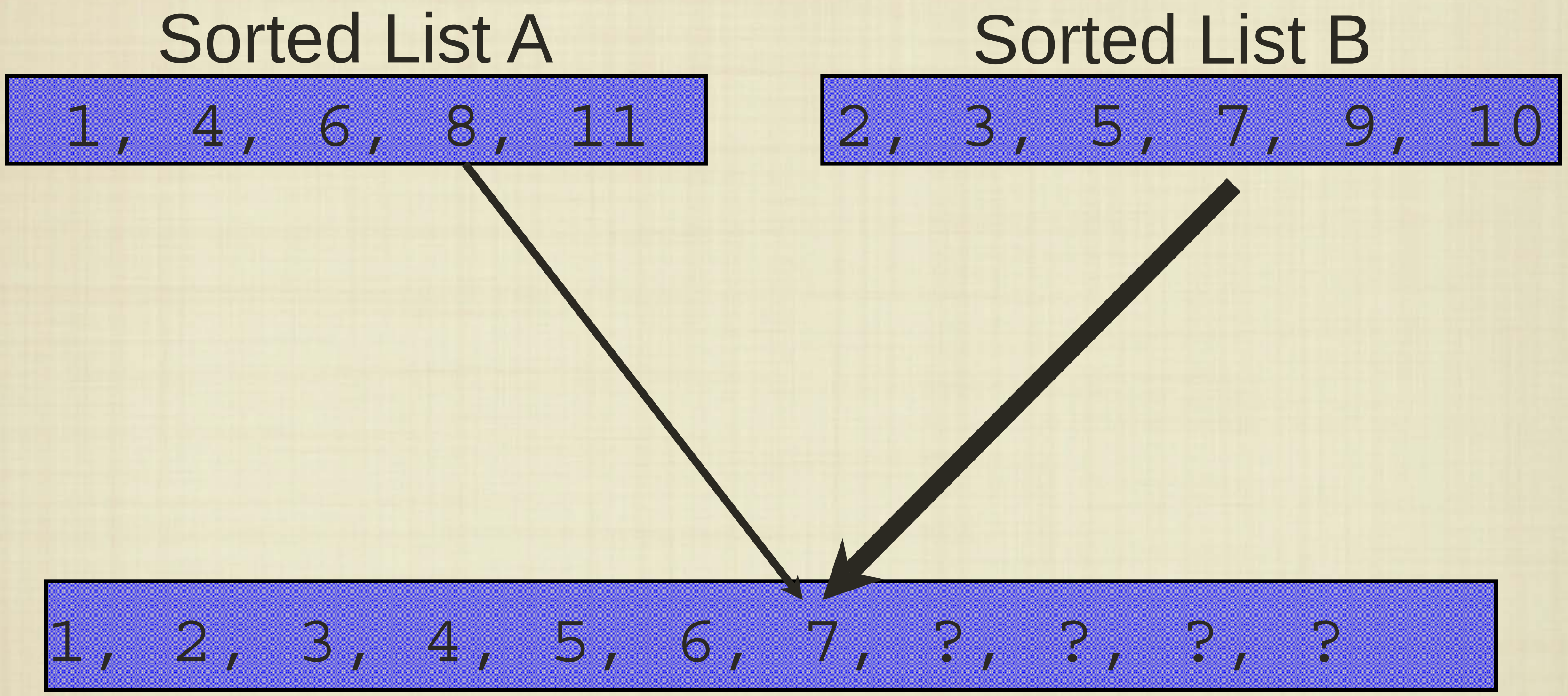
Merging Lists

- Suppose that we instead had a list that had two sorted halves. Could we do better?

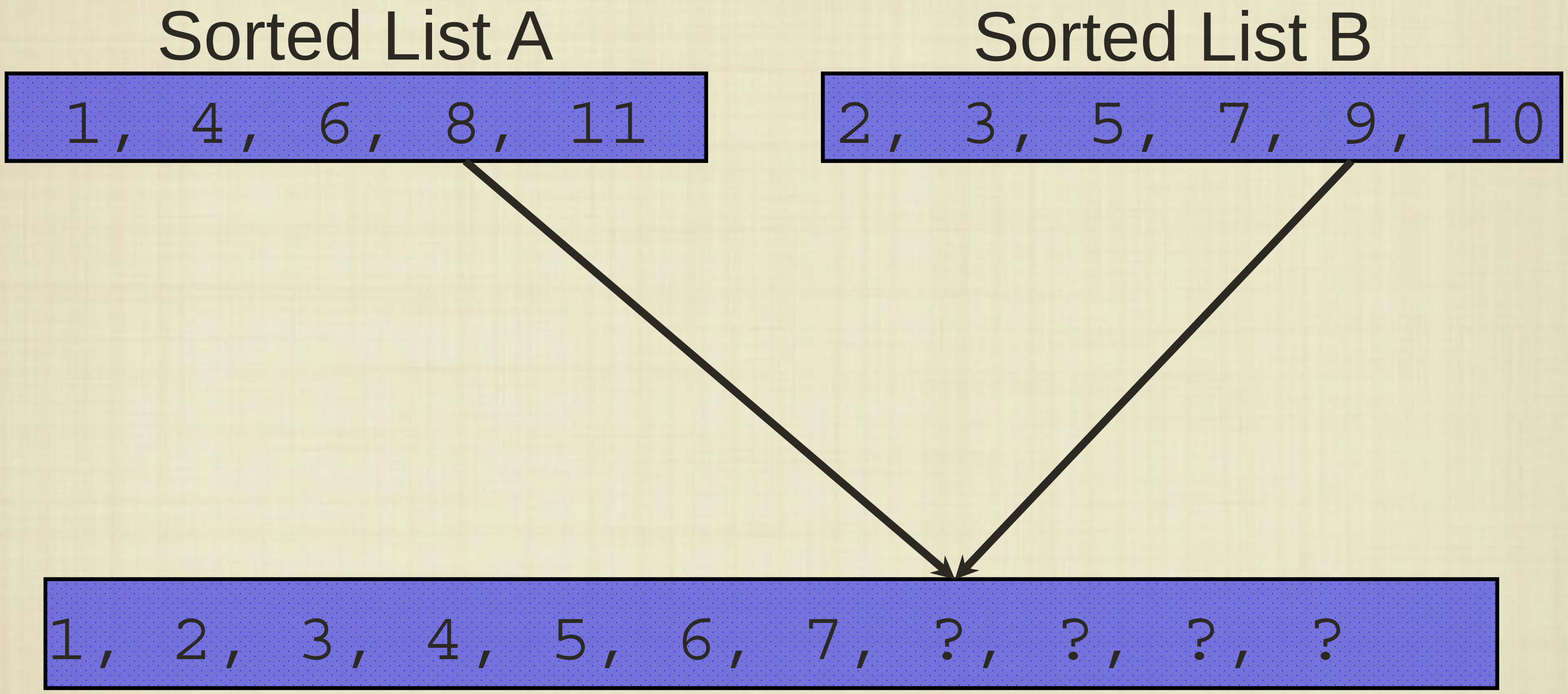


Merging Lists

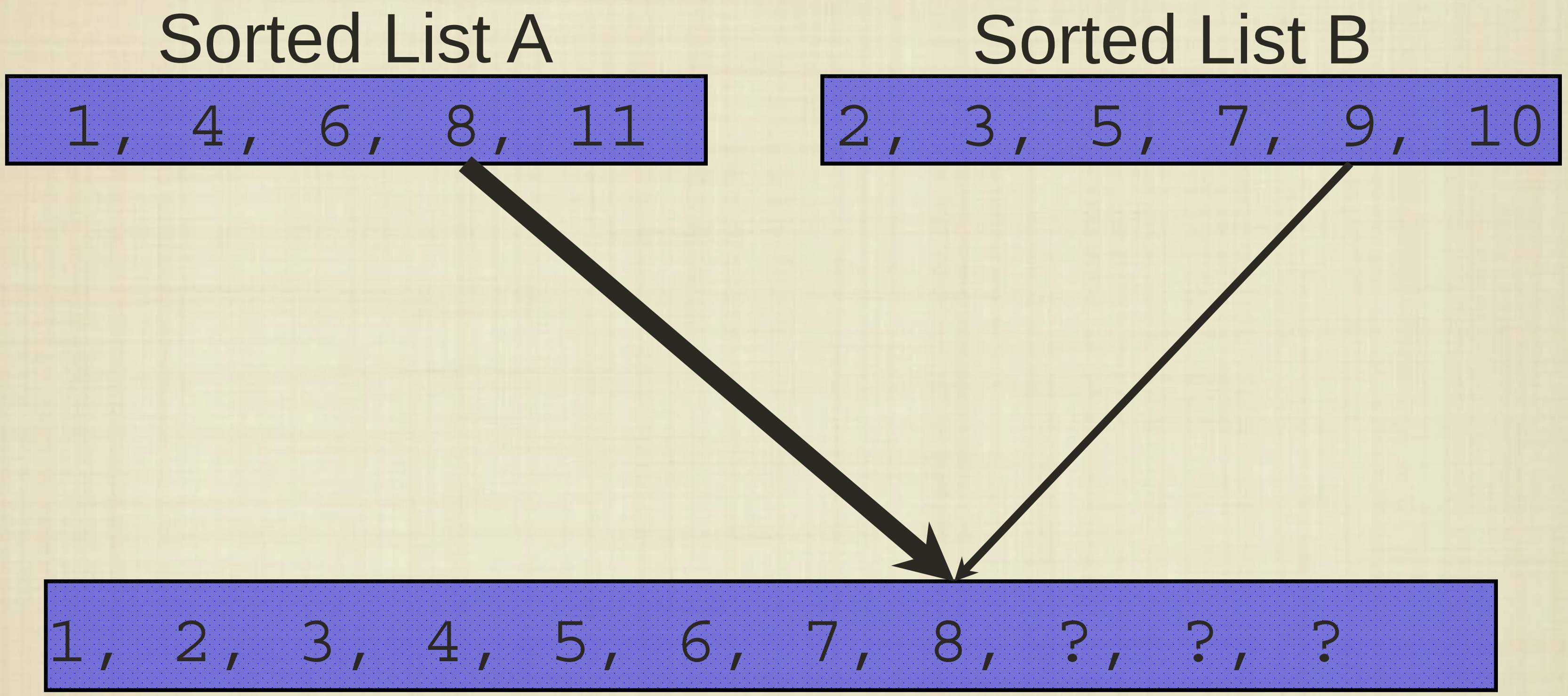
- Suppose that we instead had a list that had two sorted halves. Could we do better?



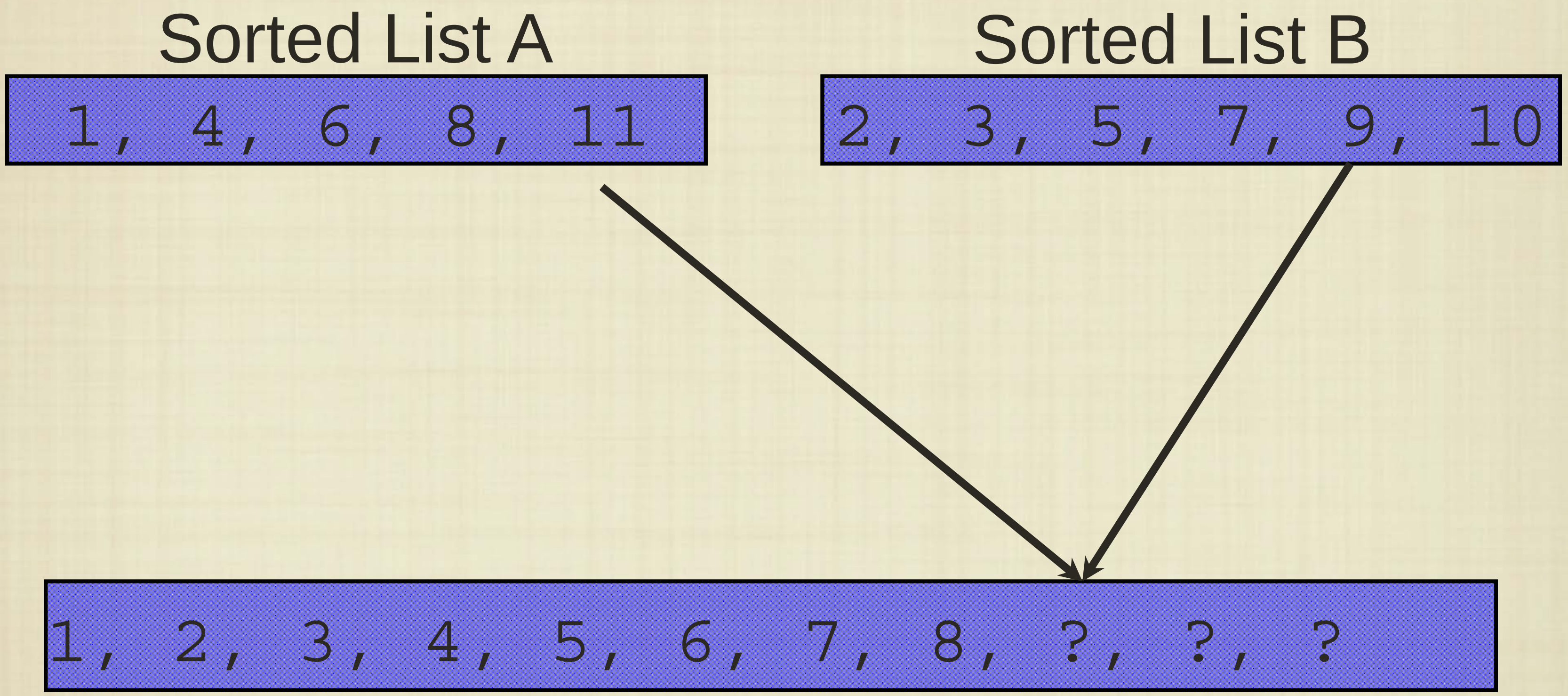
Merging Lists



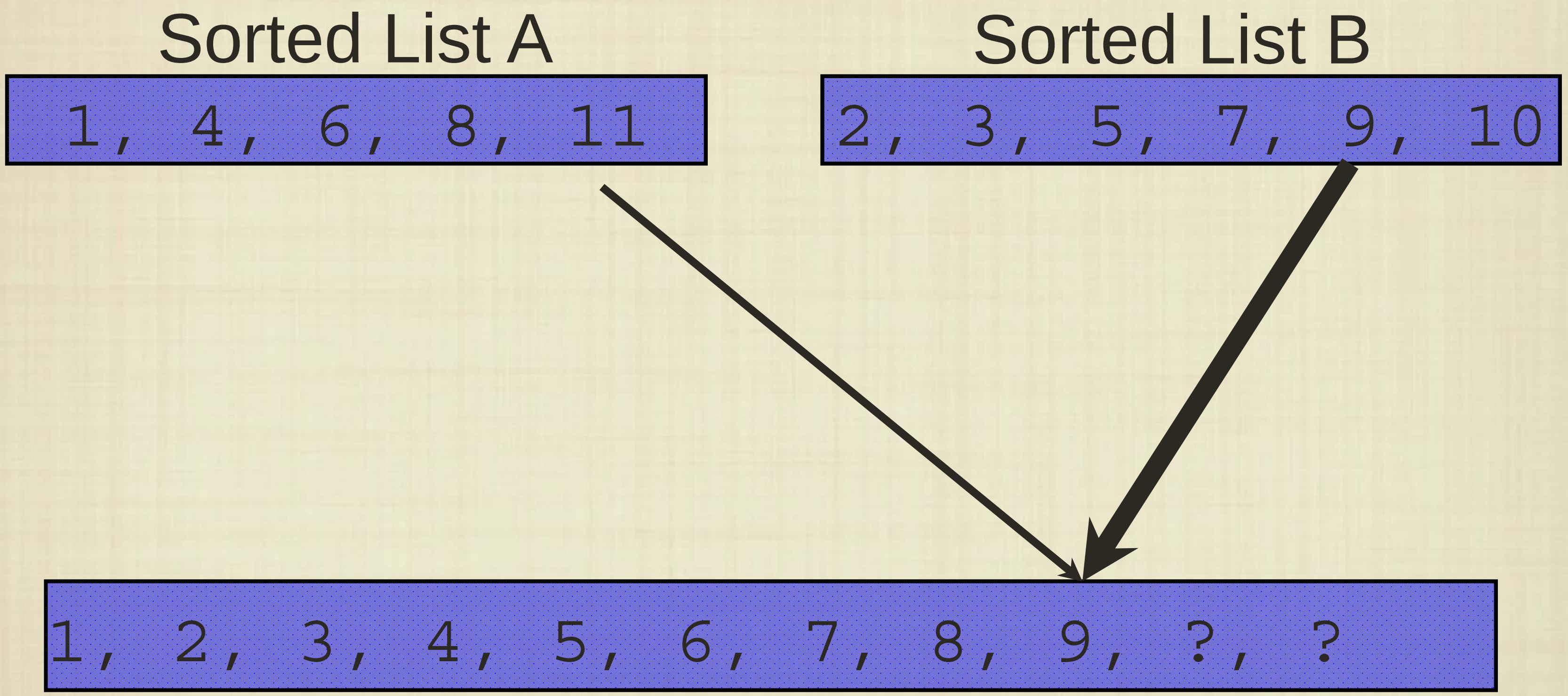
Merging Lists



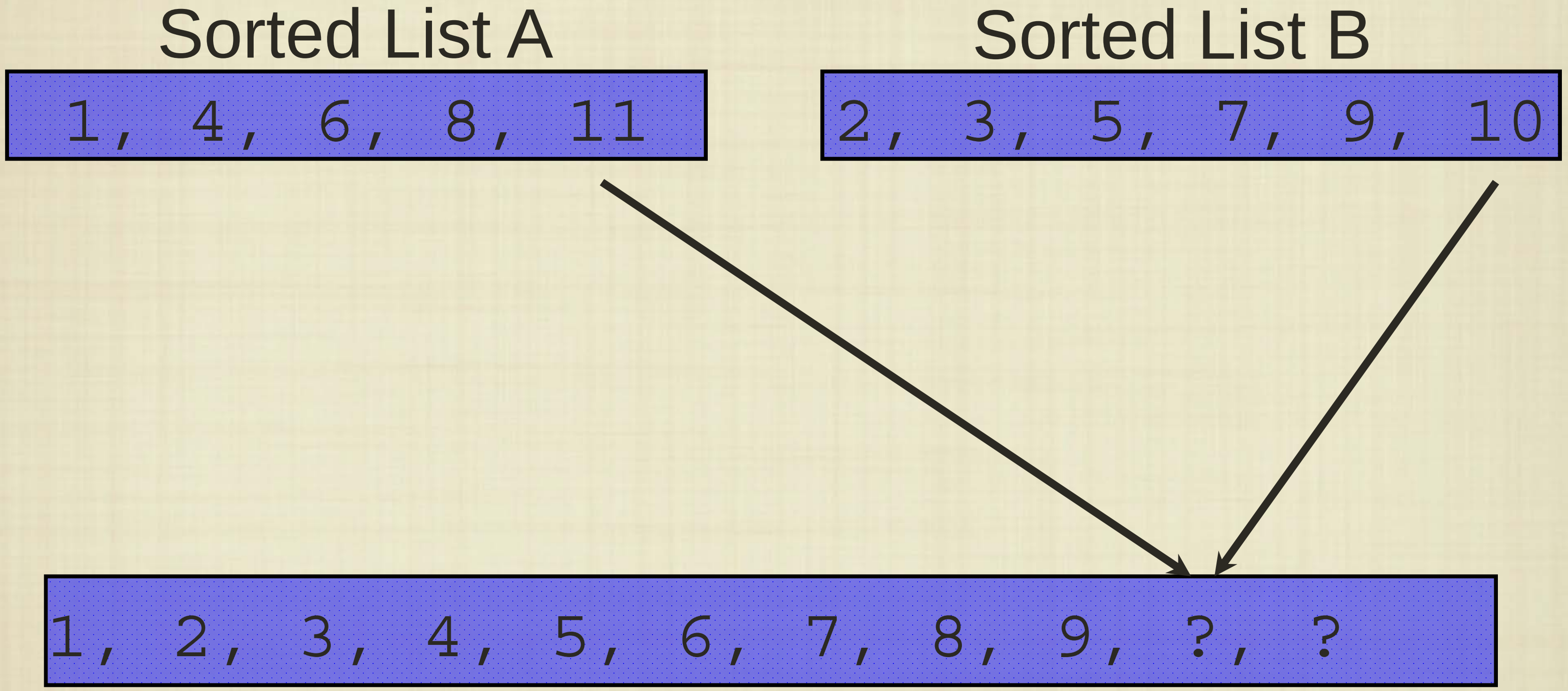
Merging Lists



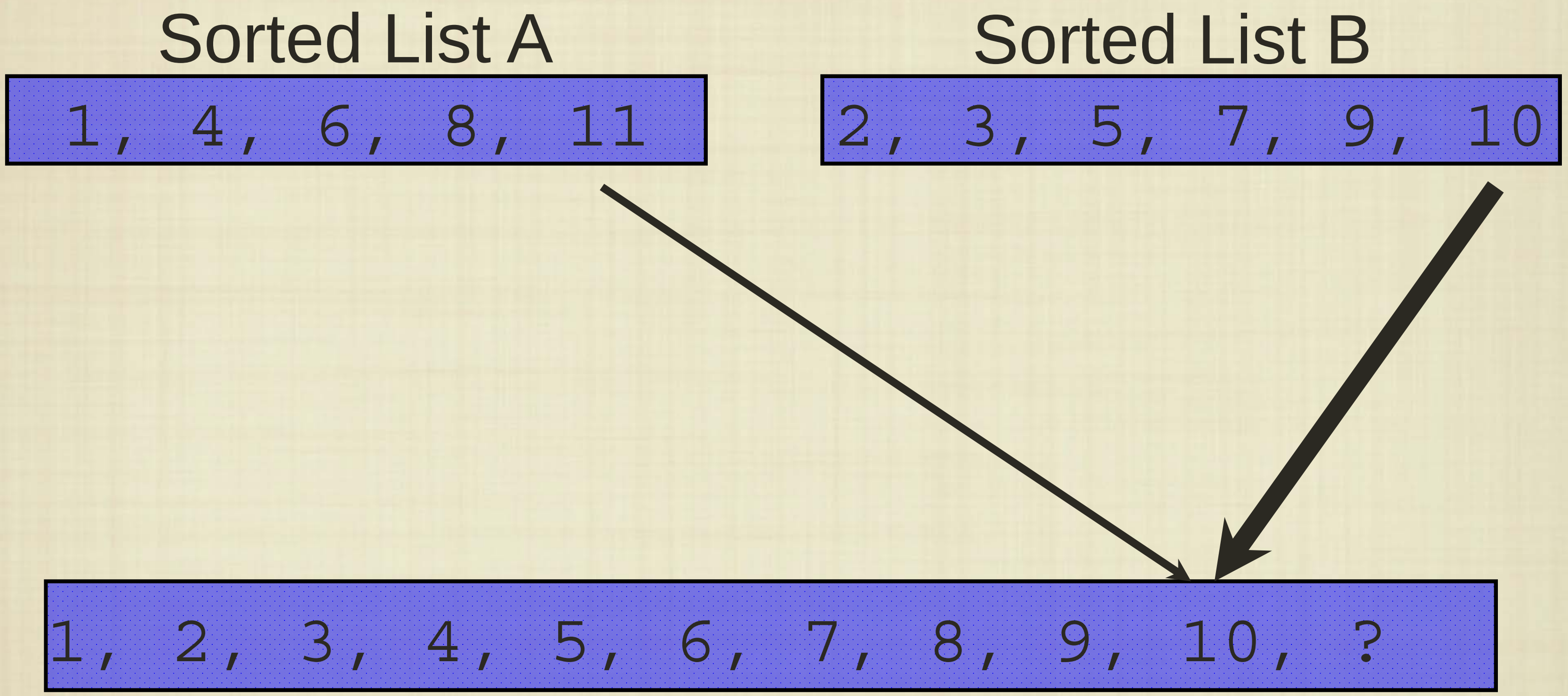
Merging Lists



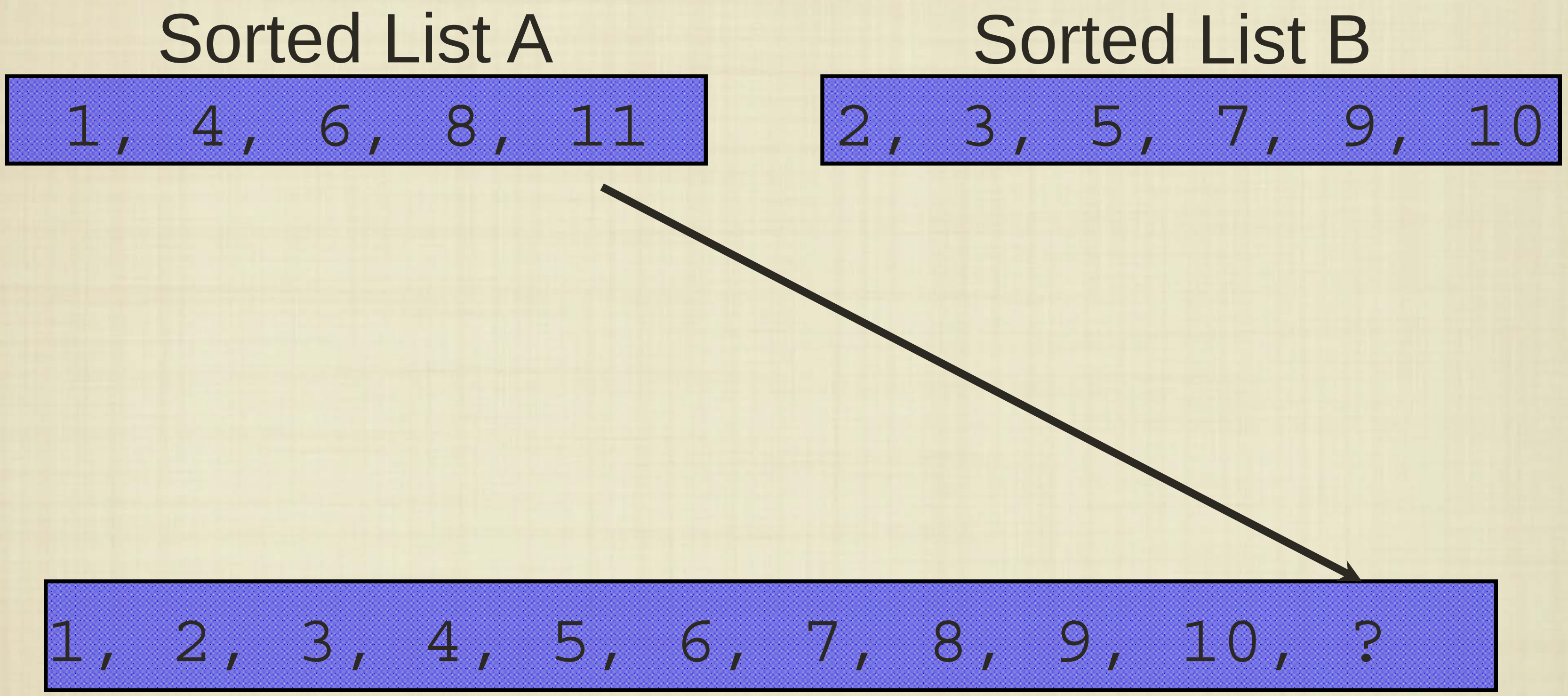
Merging Lists



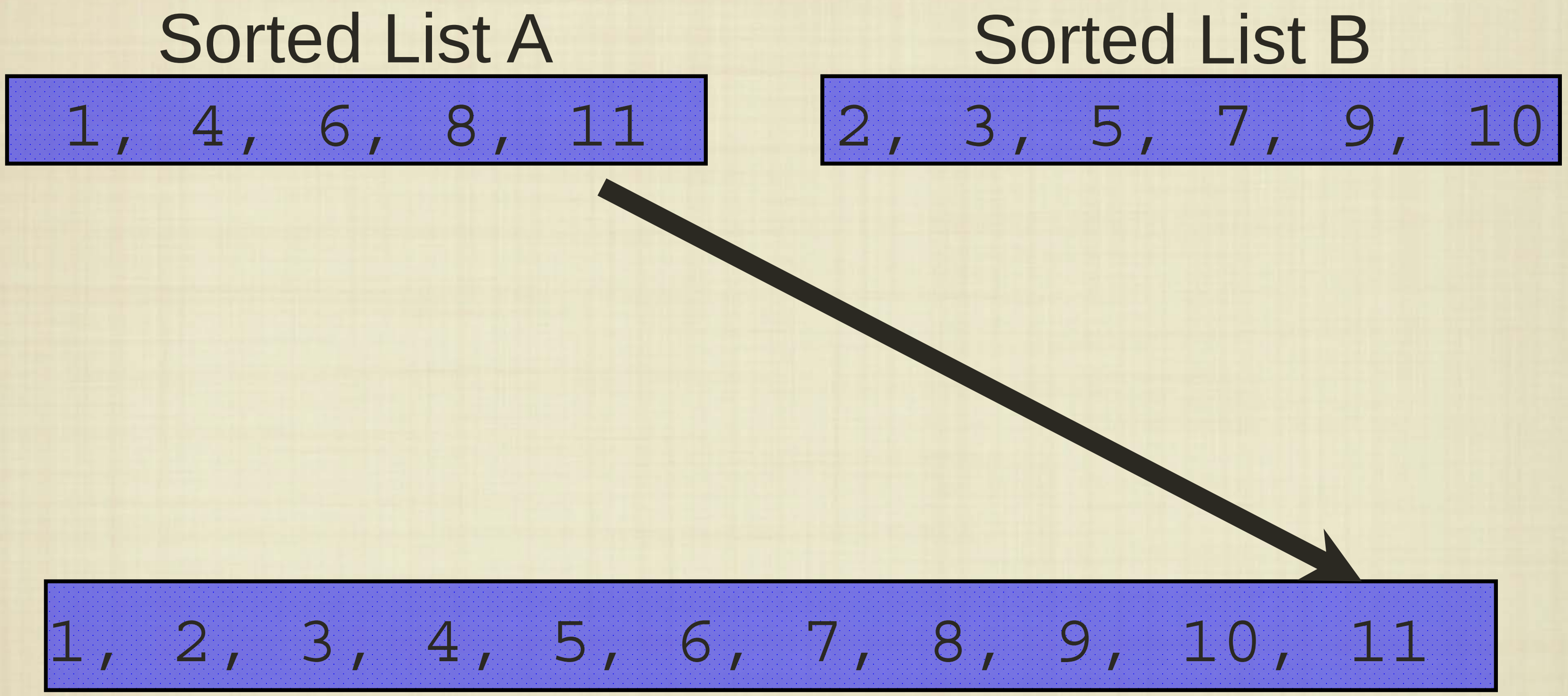
Merging Lists



Merging Lists



Merging Lists



Merging Lists

Sorted List A

`1, 4, 6, 8, 11`

Sorted List B

`2, 3, 5, 7, 9, 10`

`1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11`

The key idea is to scan through both lists, while moving the smallest element to a new list. If we finish scanning either list, the rest of the other list is appended to the result.

Merge Sort

- Functional programming languages are ideally suited to implement recursive algorithms. How would we implement merge sort?

```
(define (merge-sort L)
(if (equal? (length L) 1)
    L
    (let ([mid (quotient (length L) 2)])
        (merge (merge-sort (take L mid))
             (merge-sort (drop L mid))))))
```

Assuming `merge` is correct, is `merge-sort` correct?

How do we implement `merge`?