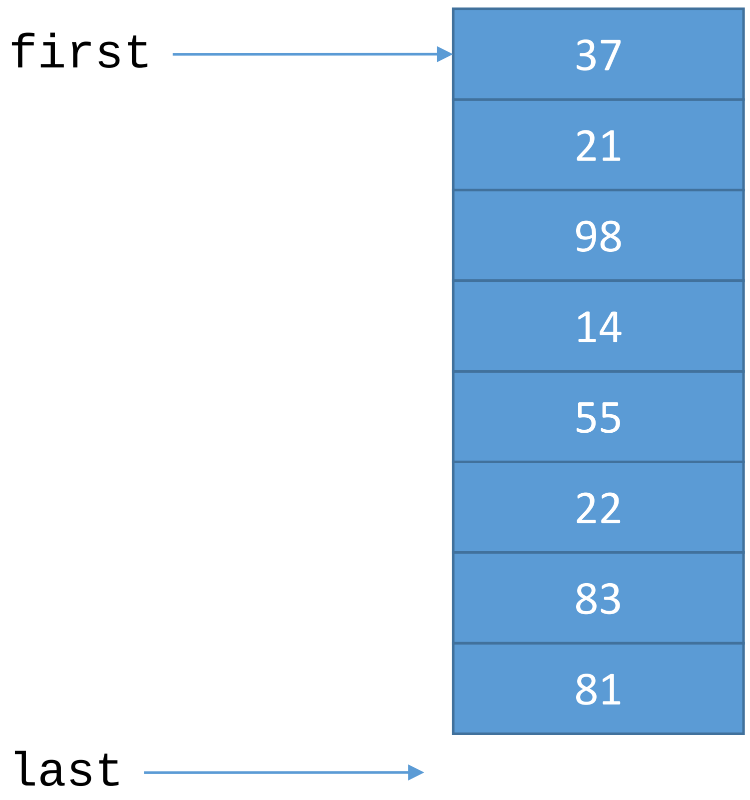
Merge Sort

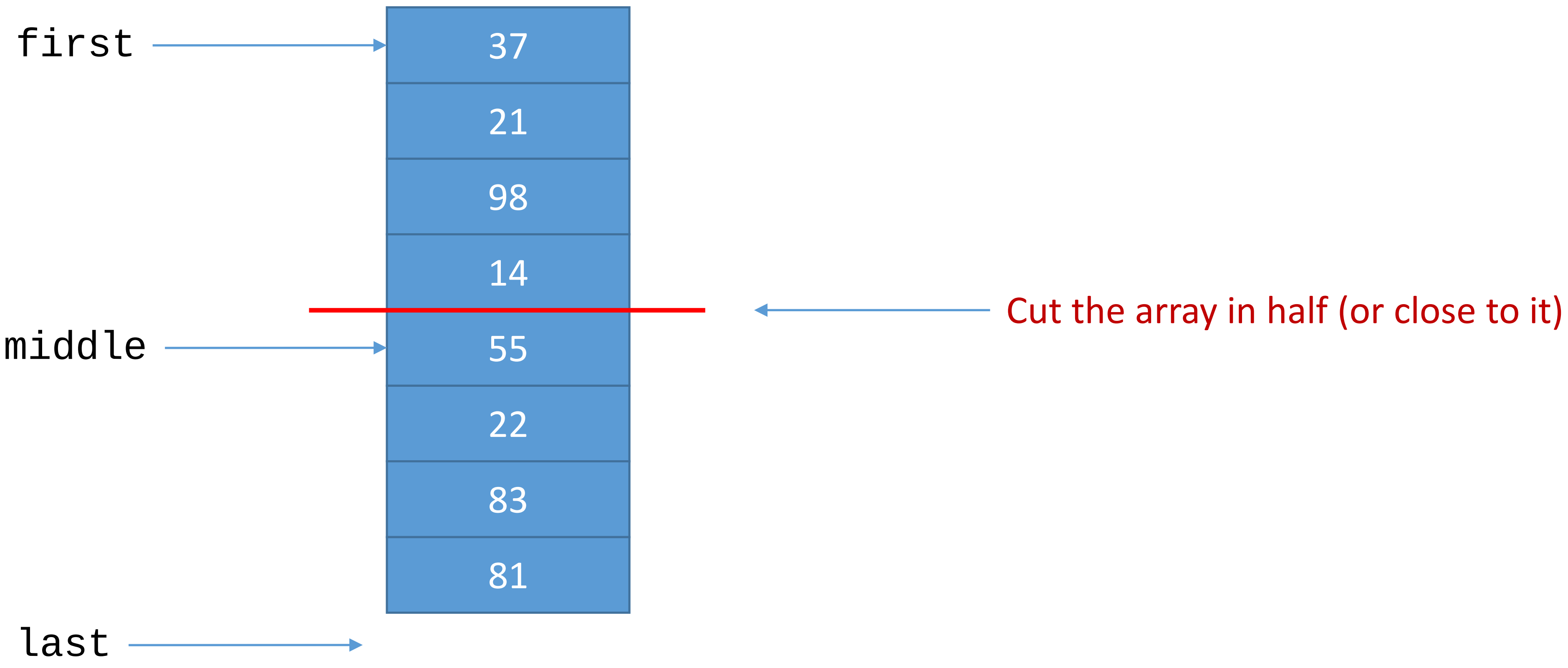
Peter Chapin

Vermont Technical College

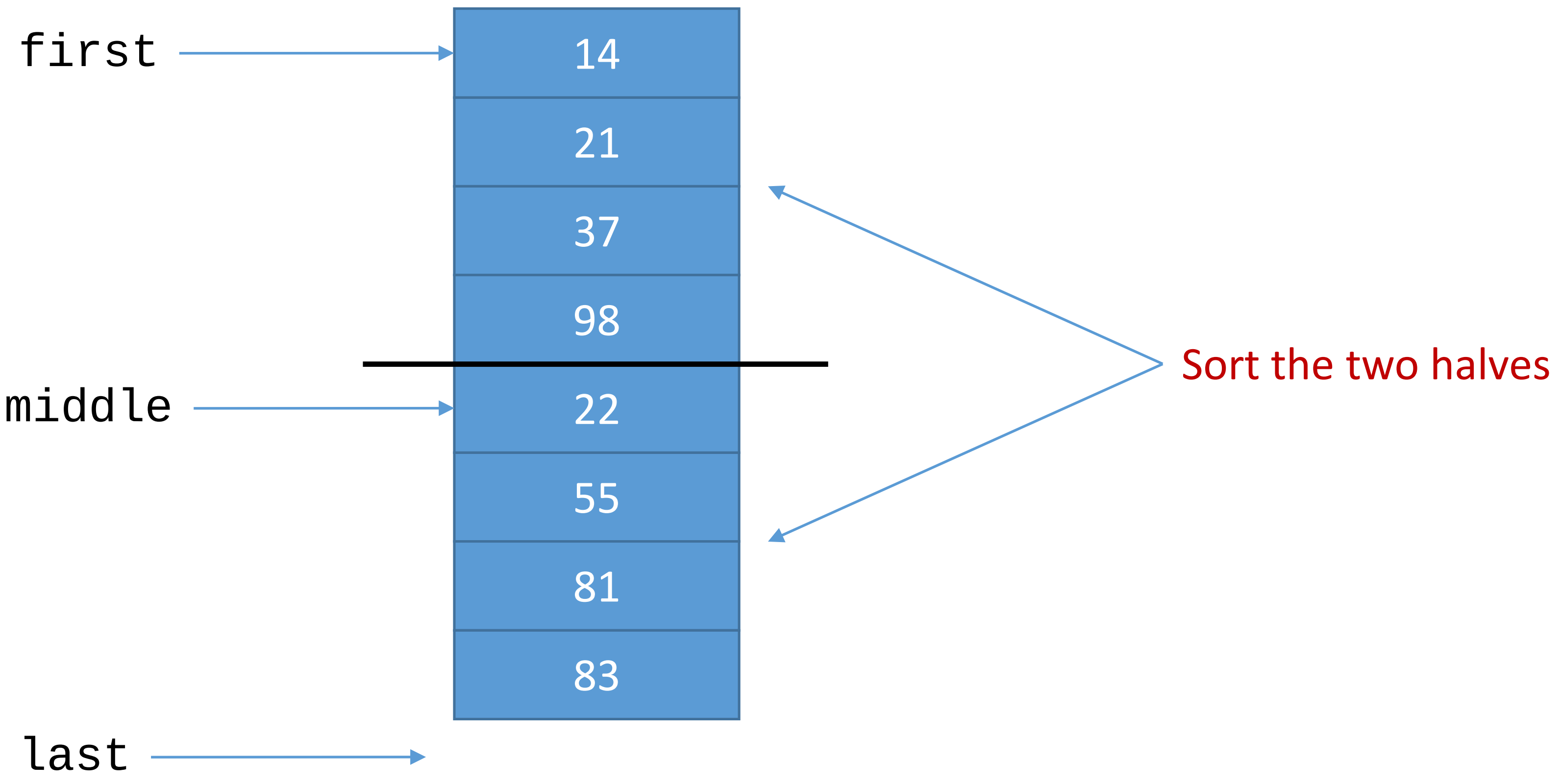
Starting Configuration



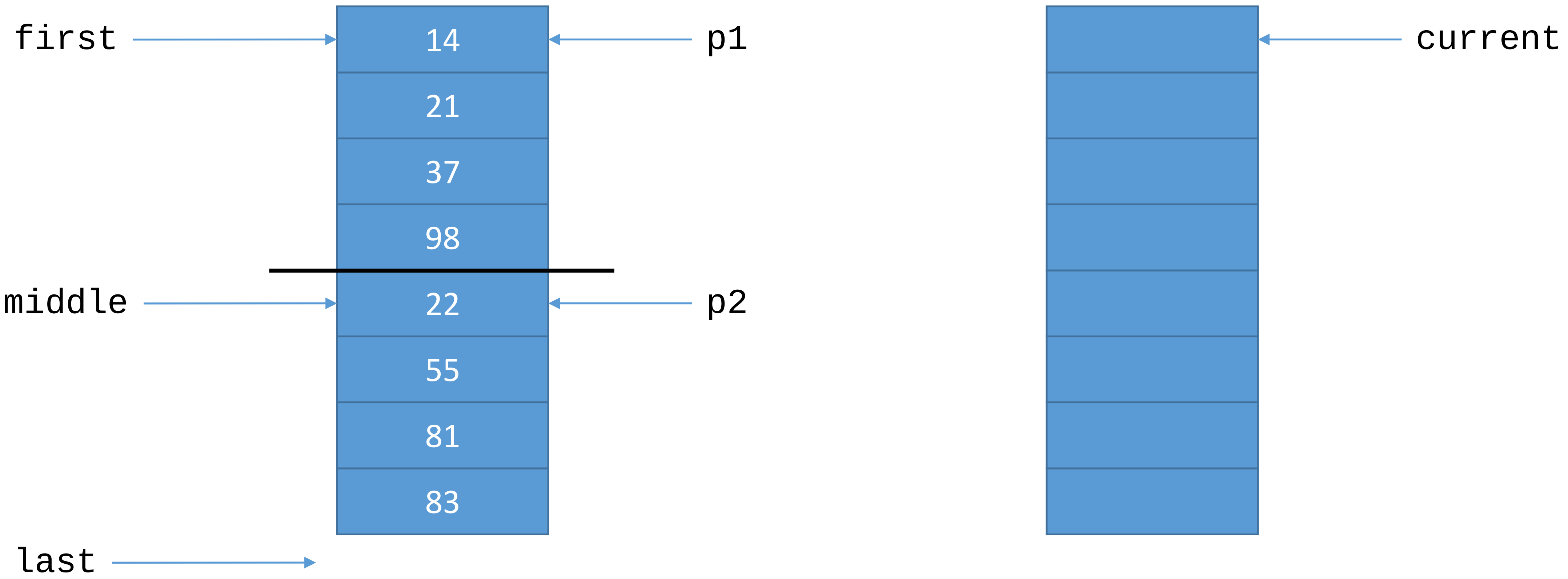
Split Into Subproblems



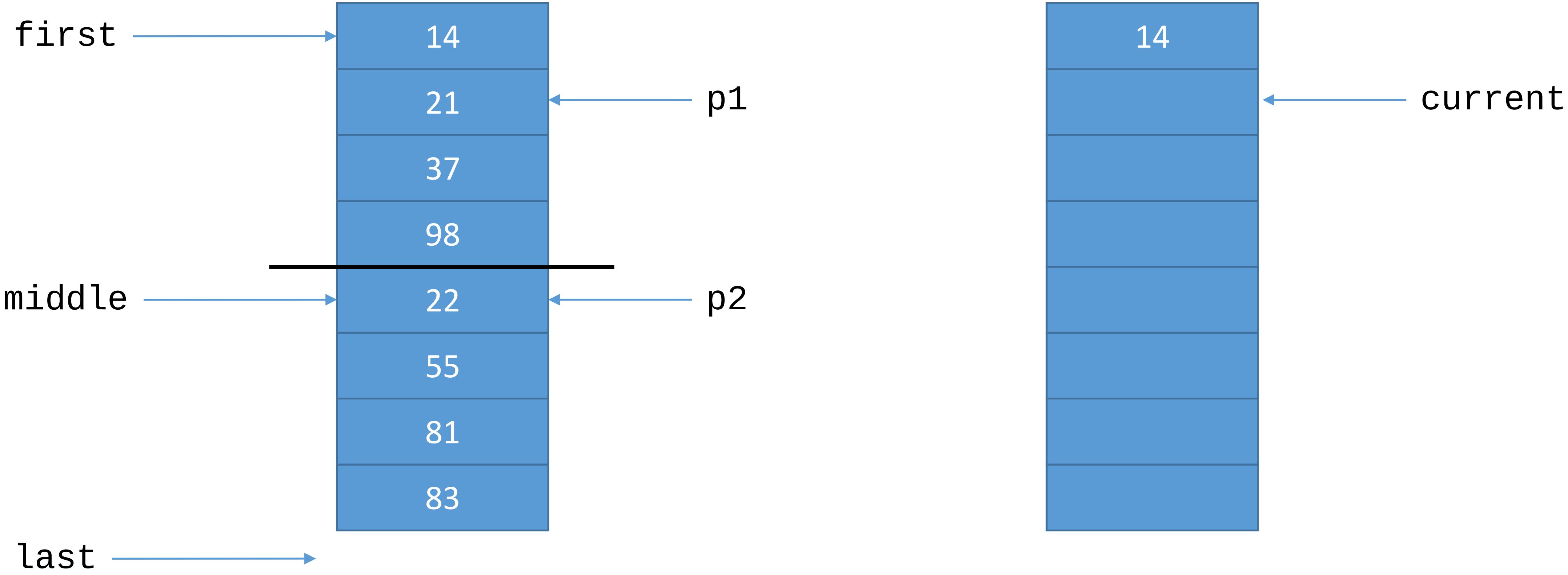
Recursively Solve Subproblems



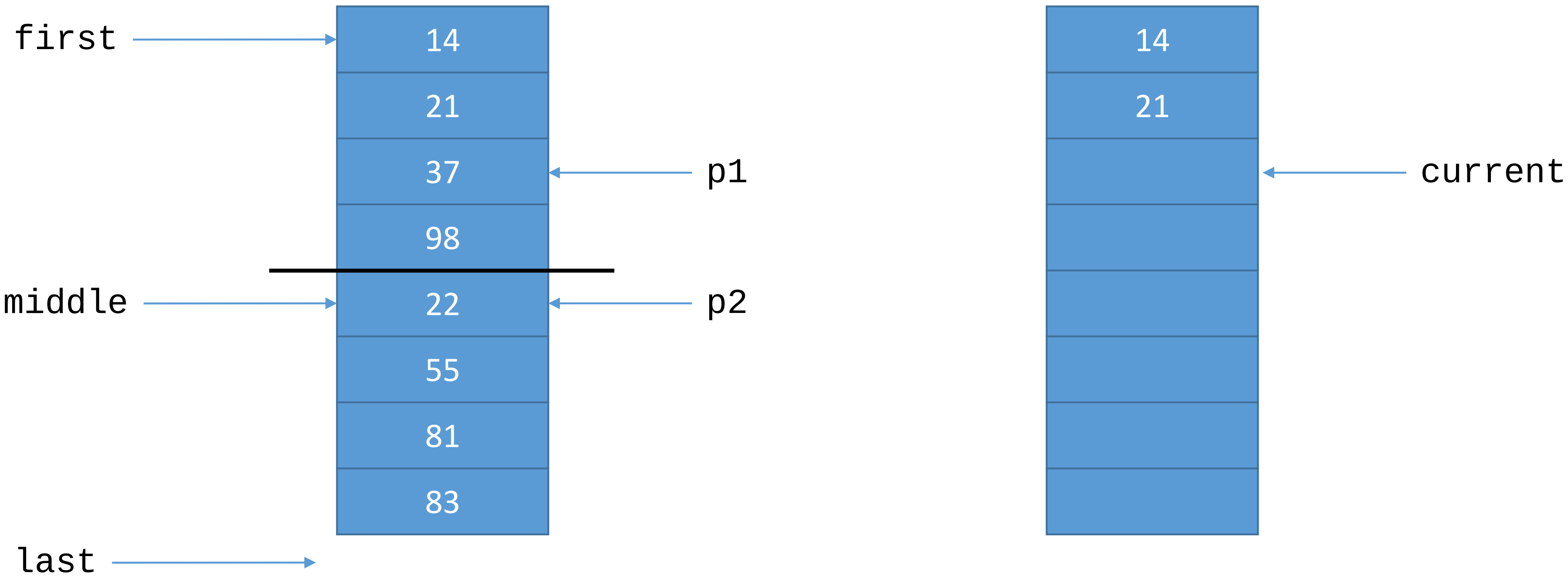
Merge the Subproblems



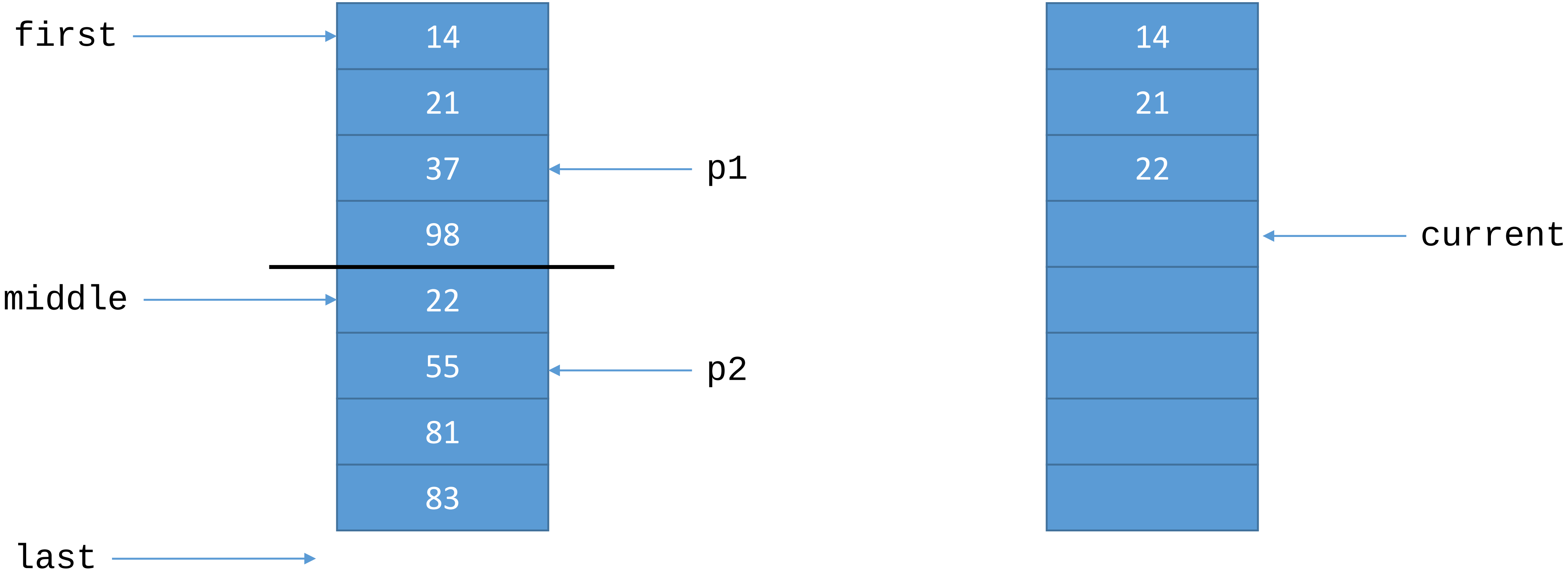
Merge the Subproblems



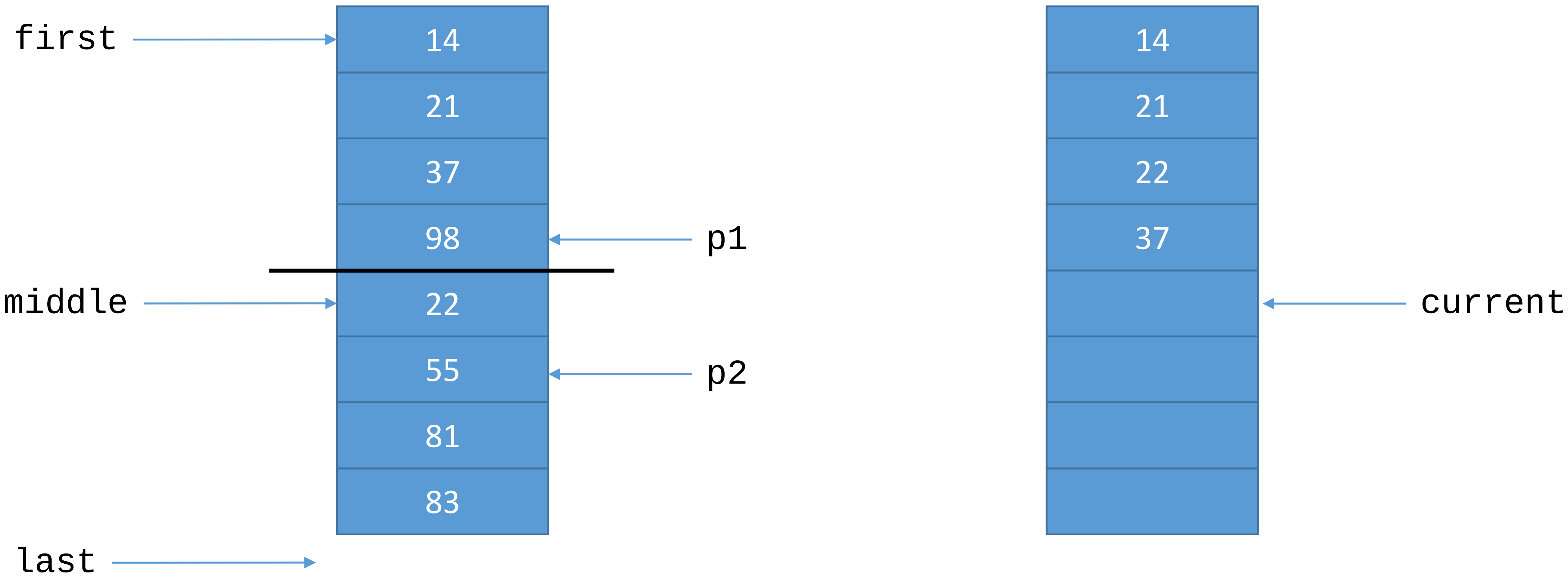
Merge the Subproblems



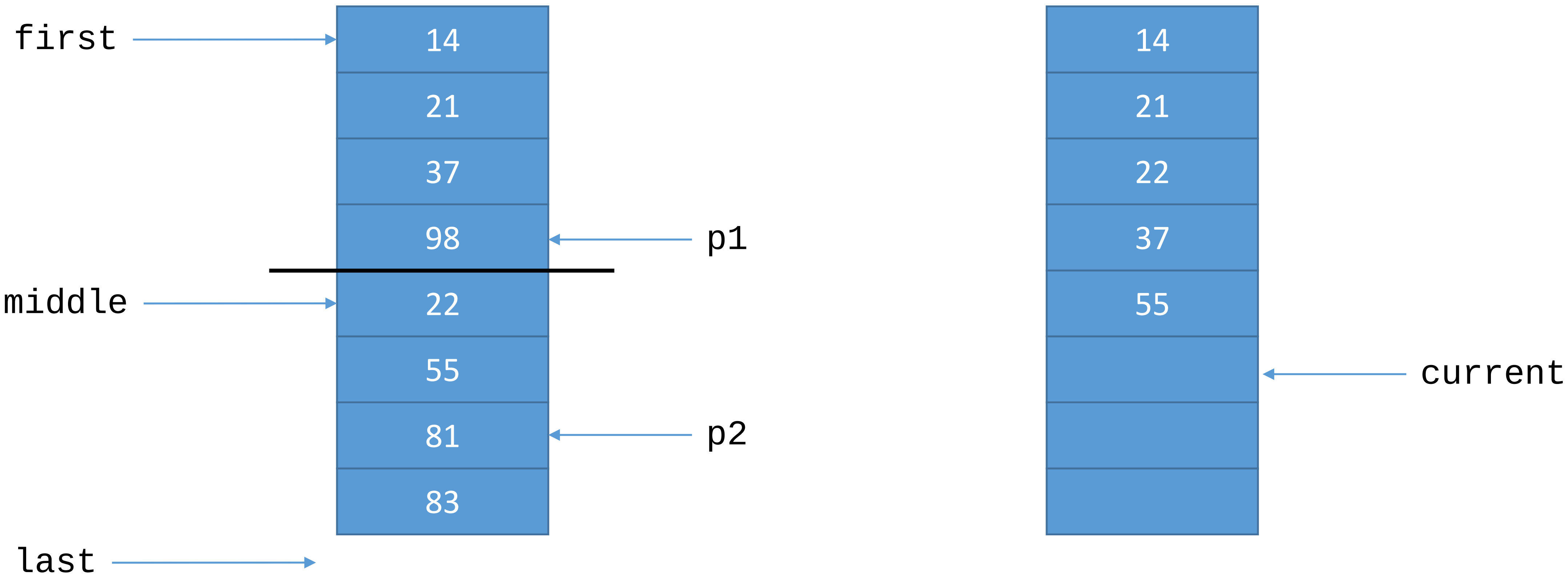
Merge the Subproblems



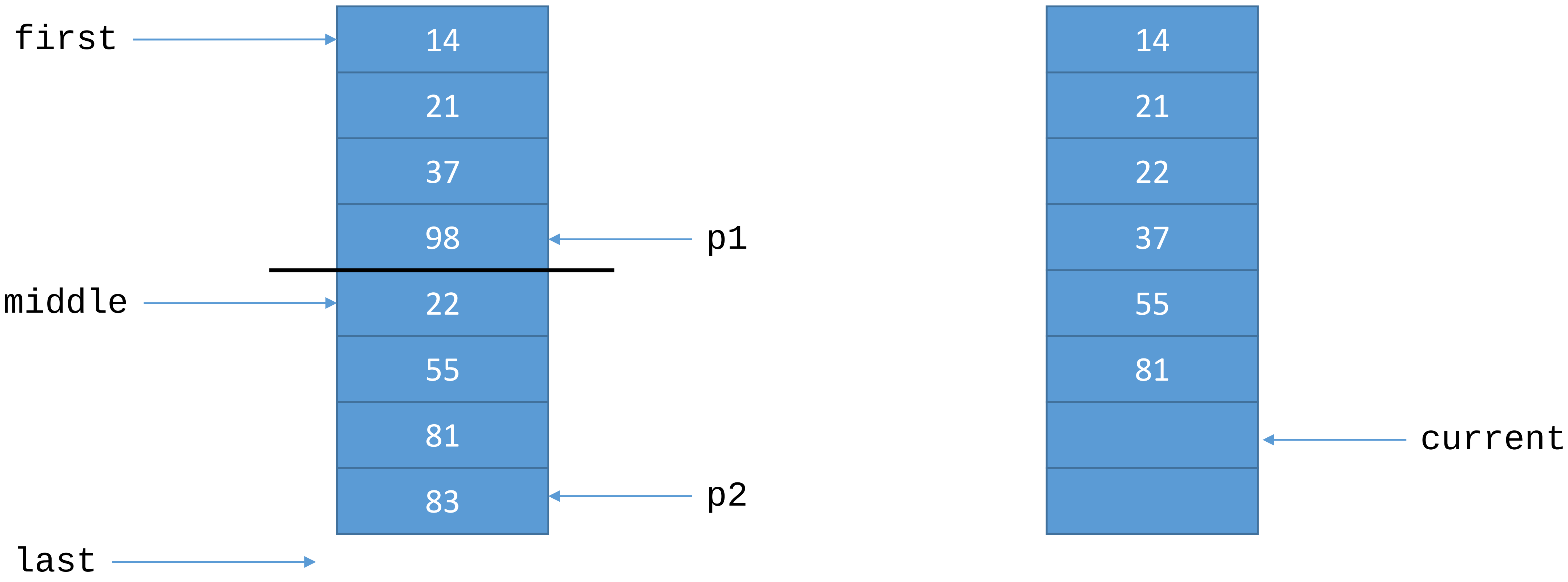
Merge the Subproblems



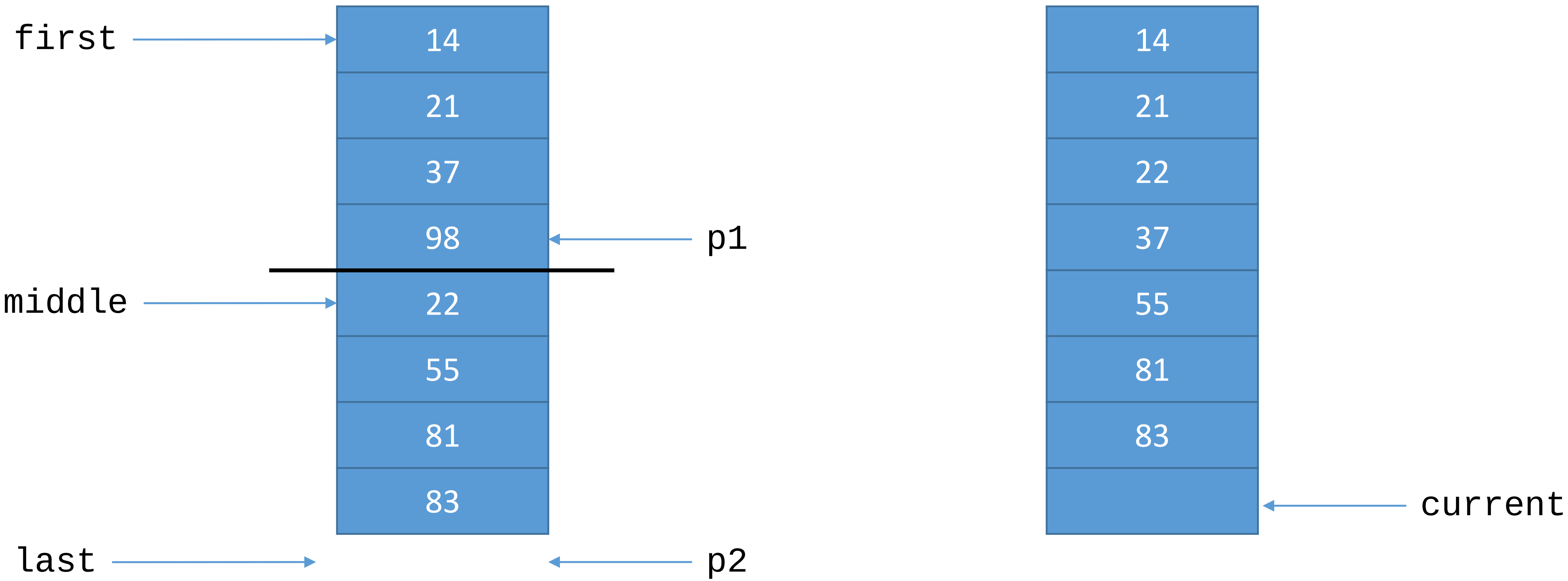
Merge the Subproblems



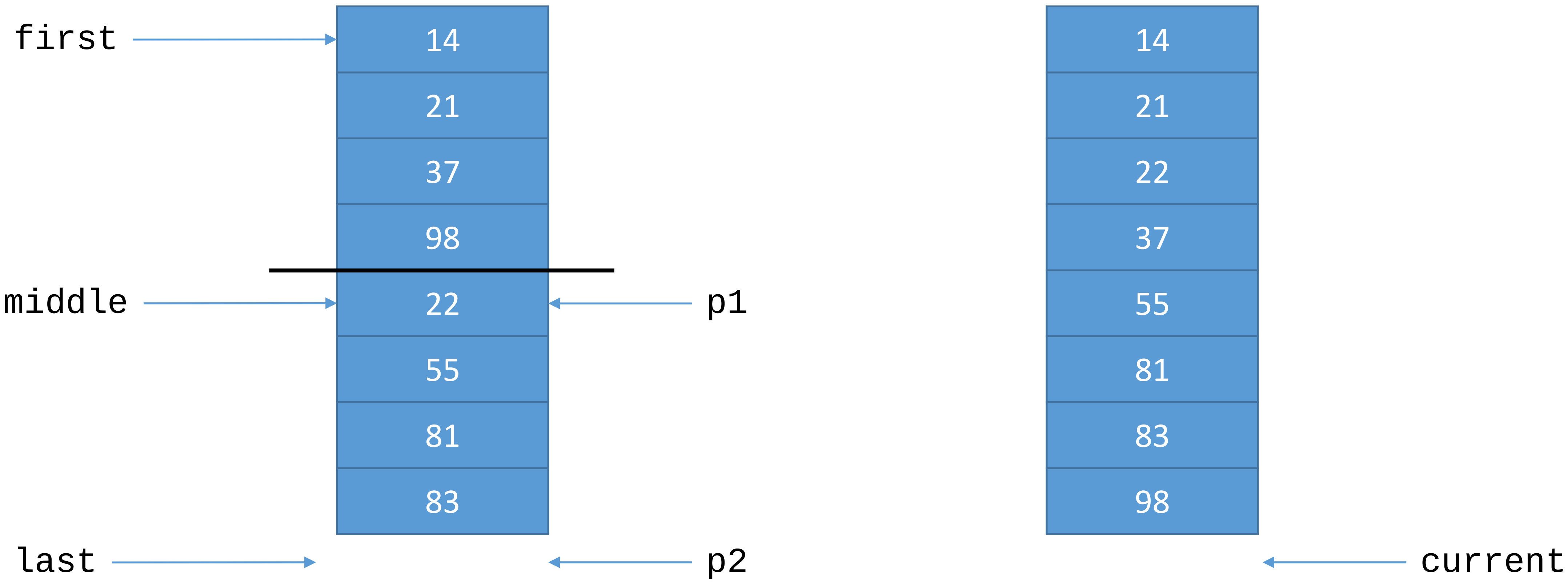
Merge the Subproblems



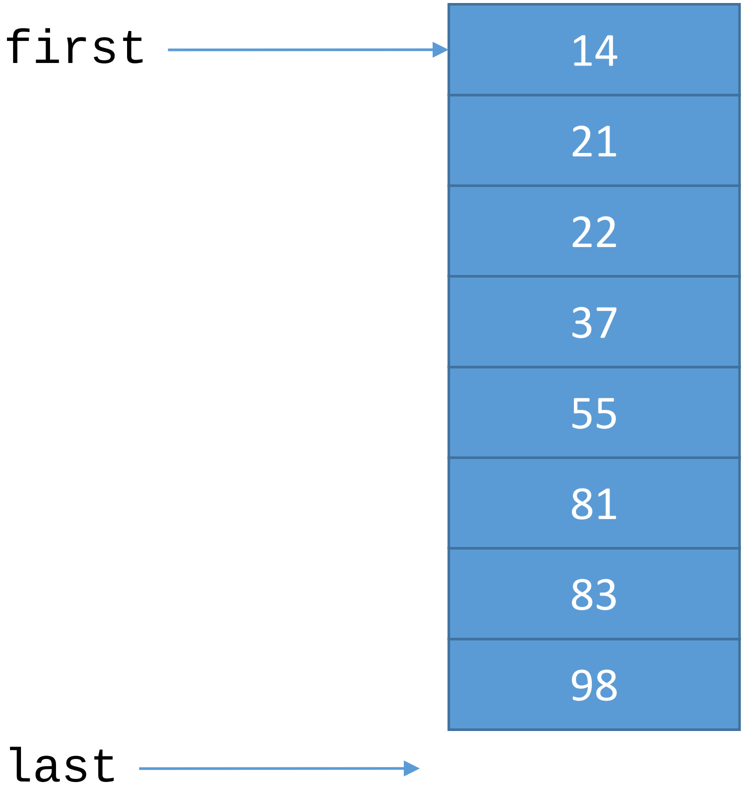
Merge the Subproblems



Merge the Subproblems



Copy Back



Pseudo-Code

```
IF <size of array is one or less> THEN
  <do nothing… array is already sorted>
ELSE
  <find midpoint of array>
  <sort first half>
  <sort second half>
  <merge the sorted halves together>
END IF
```

Requires allocating (and freeing!) a temporary array

Space and Time

- Merge Sort requires O(n) additional space beyond array.
  - Thus the method is expensive on space
  - Compare: Insertion Sort requires O(1) additional space!
- Time?
  - Not immediately obvious:
    - `T(n) = 2*T(n/2) + O(n)` ← Linear time to merge
    - A *recurrence* formula
  - Works out to O(n log(n))
  - Far superior to Insertion Sort's $O(n^2)$

Overhead of Recursion

- Using recursion down to subarray sizes of 1 is excessive
  - Huge overheads slow down the algorithm (though it remains O(n log(n))).
- Switch to another algorithm for small subarrays.

```
IF <size of array is less than threshold> THEN
  <use Insertion Sort on the array>
ELSE
  <find midpoint of array>
  <sort first half>
  <sort second half>
  <merge the sorted halves together>
END IF
```

Ideal threshold value found experimentally