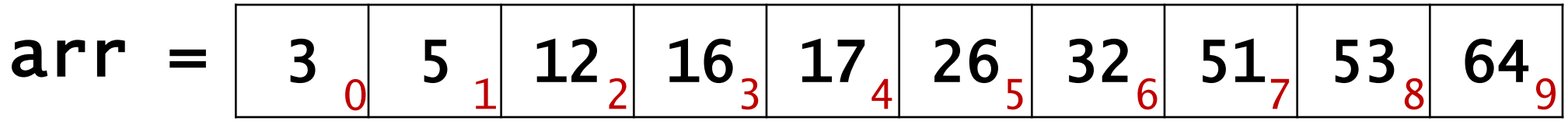
Binary Search

Shusen Wang

Problem of Search

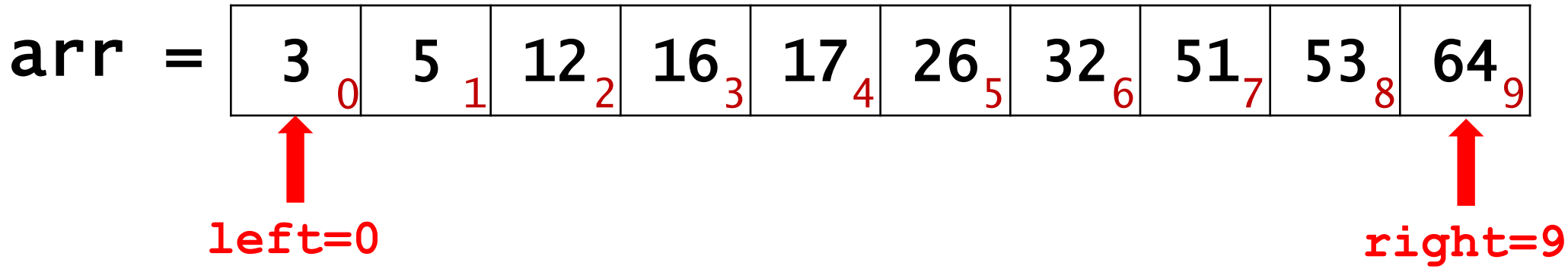


Problem of Search

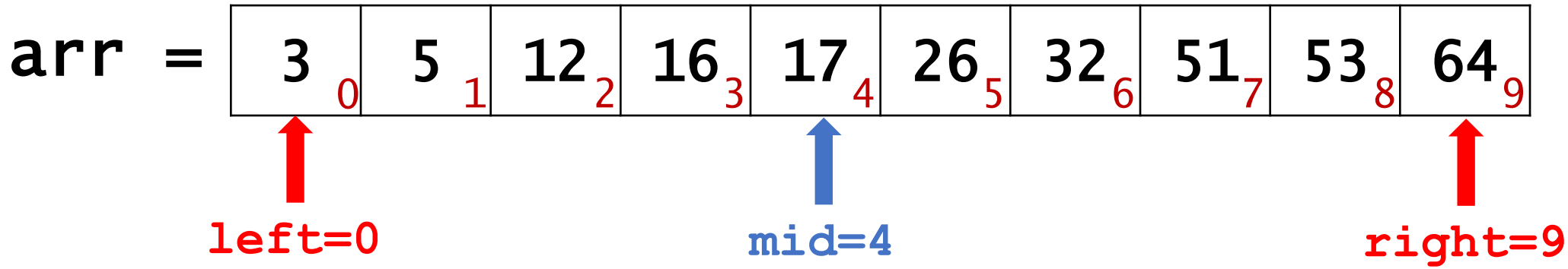
arr =	3 0	5 1	12 2	16 3	17 4	26 5	32 6	51 7	53 8	64 9

- **Inputs:** (i) an array whose elements are in the ascending order and (ii) a key.
- **Goal:** Search the key in the array. If found, return its index; if not found, return -1.

Example: key=26

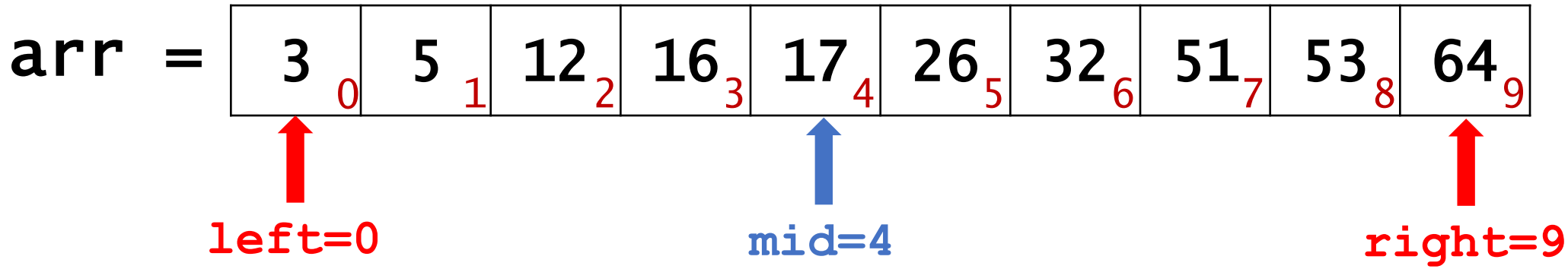


Example: key=26



- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- `If arr[mid]==key  ==> return mid.`
- `If arr[mid]>key  ==> right ← mid-1.`
- `If arr[mid]<key  ==> left ← mid+1.`

Example: key=26

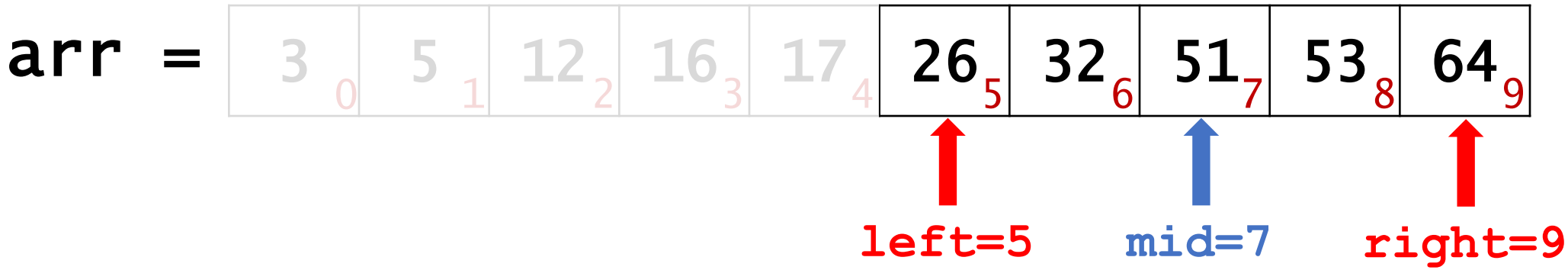


- mid ← [(left + right)/2].
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right ← mid-1.
- If arr[mid]<key ==> left ← mid+1.

Example: key=26

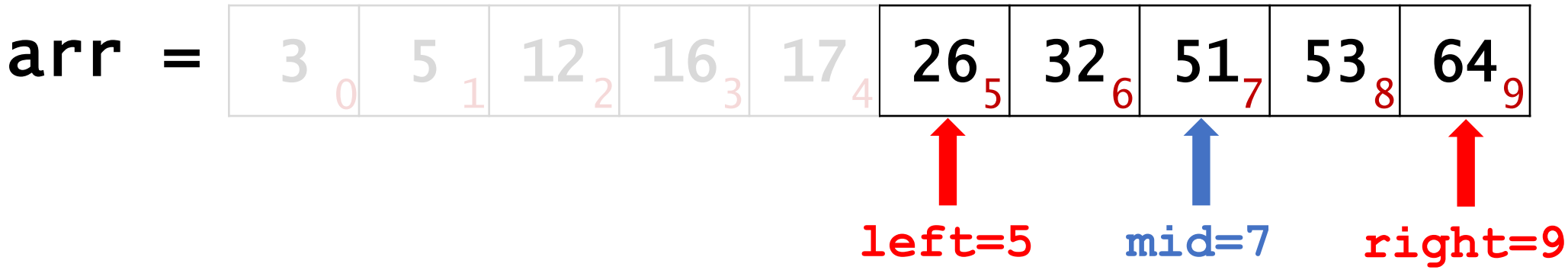
- mid ← ⌊(left + right)/2⌋.
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right ← mid-1.
- If arr[mid]<key ==> left ← mid+1.

Example: key=26



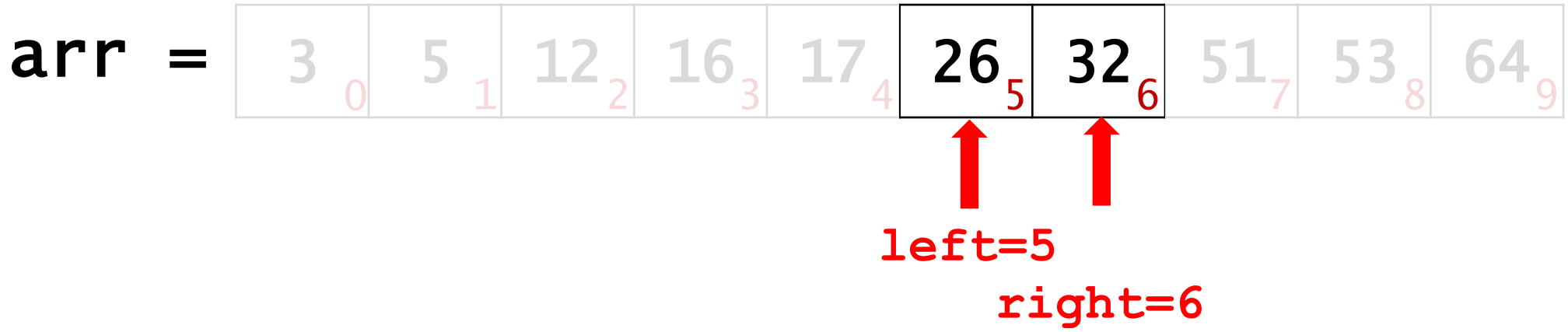
- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right ← mid-1.
- If arr[mid]<key ==> left ← mid+1.

Example: key=26



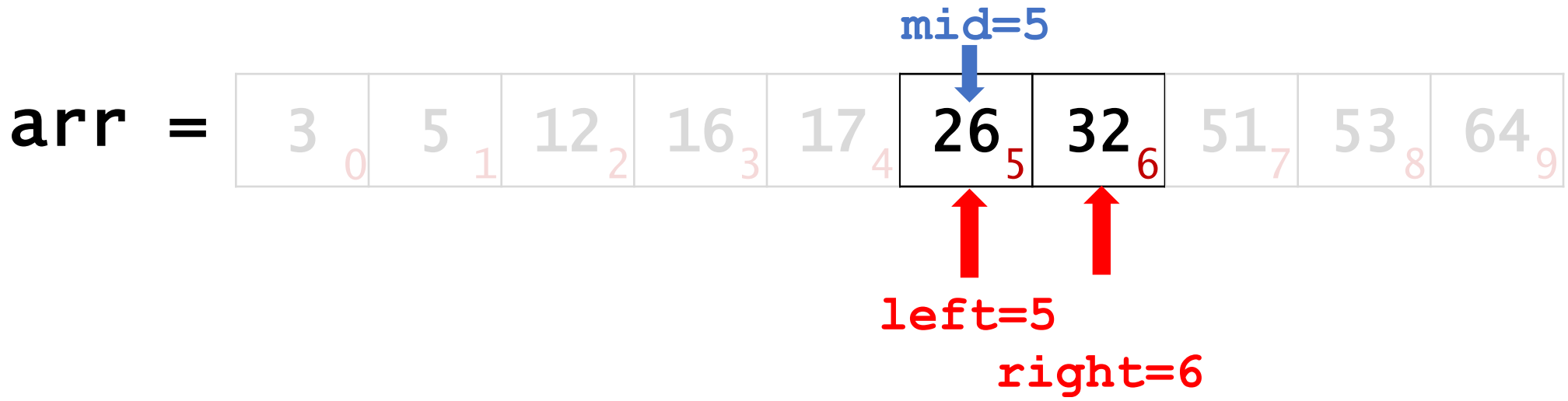
- mid ← [(left + right)/2].
- If arr[mid]==key ==> return mid.
- **If arr[mid]>key ==> right← mid-1.**
- If arr[mid]<key ==> left ← mid+1.

Example: key=26



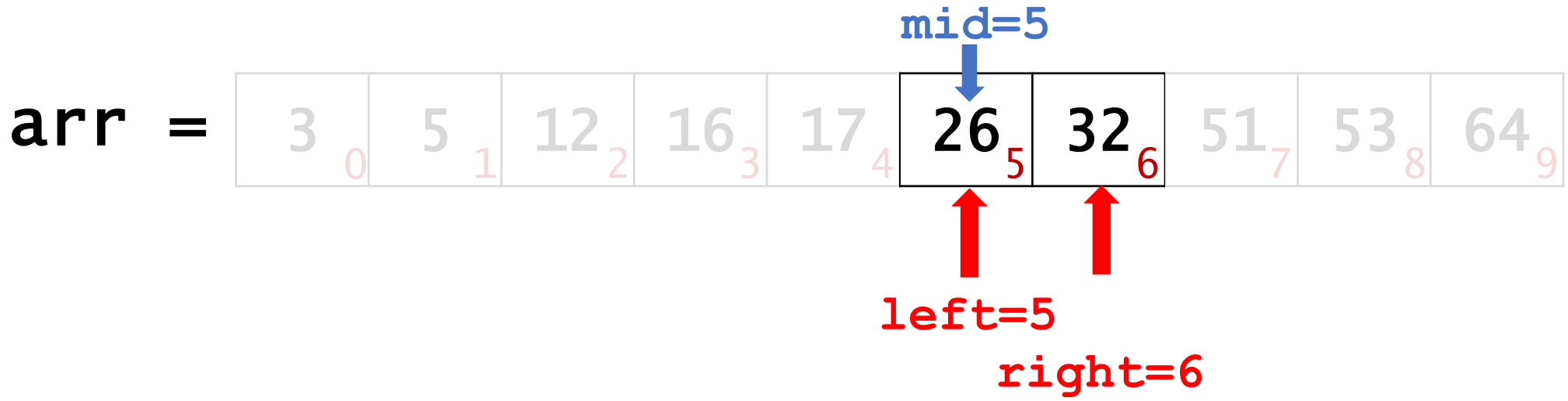
- mid ← [(left + right)/2].
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right← mid-1.
- If arr[mid]<key ==> left ← mid+1.

Example: key=26



- mid $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$.
- If arr[mid]==key ==> return mid.
- If arr[mid]>key ==> right← mid-1.
- If arr[mid]<key ==> left← mid+1.

Example: key=26



- mid ← [(left + right)/2].
- **If arr[mid]==key ==> return mid.**
- If arr[mid]>key ==> right← mid-1.
- If arr[mid]<key ==> left← mid+1.

Time Complexity

- Each iteration reduce the size of array by half.
- Let n be the size of array.
- The total number of iterations is at most $\log_2 n$.
- Time complexity: $O(\log n)$.

```
int search(int arr[], int left, int right, int key) {
      while (left <= right) {
            int mid = (left + right) / 2;
            if (arr[mid] == key)
                  return mid; // key is found
            if (arr[mid] < key)
                  left = mid + 1; // search in the right half
            else
                  right = mid - 1; // search in the left half
      }
      return -1; // key is not found
}
```

How to support both search and insertion?

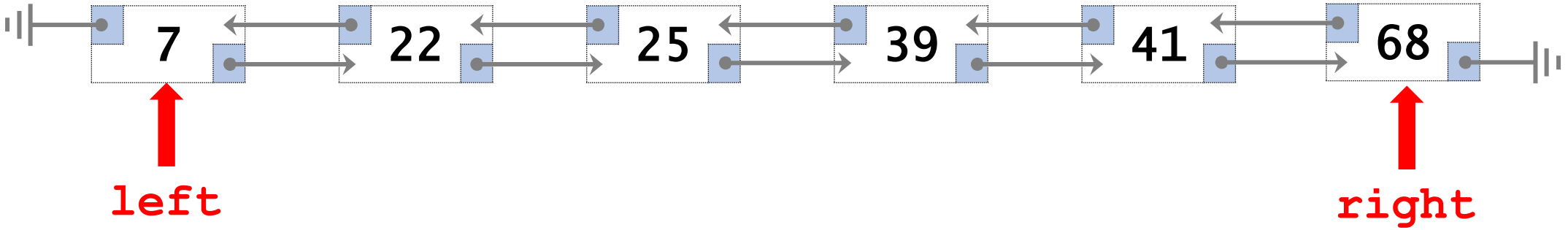
Vector

v =

3 0	5 1	12 2	16 3	17 4	26 5	32 6	51 7	53 8	64 9

- The ascending order must be kept; otherwise, search would take $O(n)$ time.
- Inserting item in the right position has $O(n)$ time complexity (on average).

List



- Can we perform binary search in the list?
- No. Given left and right, we cannot get mid efficiently.

Thank You!