

```
21  bool find(int key, Node<T>*&prev, Node<T>*&curr, Node<T>*&next, MemManager* mm) {
22      prev = list;
23      mm->try_reserve(prev);
24      curr = prev->next.load();
25      while (curr != nullptr) {
26          if (mm->try_reserve(curr)) {
27              if (prev->next.load() != curr) {
28                  mm->unreserve(prev); mm->unreserve(curr);
29                  return find(key, prev, curr, next);
30              }
31          }
32          next = curr->next.load();
33          if (is_marked(next)) { // curr is logically deleted
34              Node<T> *tmp = unmark(next);
35              if (!prev->next.compare_exchange_strong(curr, tmp)) {
36                  mm->unreserve(prev); mm->unreserve(curr);
37                  return find(key, prev, curr, next);
38              }
39              mm->unreserve(curr);
40              mm->sched_for_reclaim(curr);
41              curr = tmp;
42          }
43      else {
44          int ckey = curr->key;
45          if (prev->next.load() != curr) {
46              mm->unreserve(prev); mm->unreserve(curr);
47              return find(key, prev, curr, next);
48          }
49          if (ckey >= key) {
50              return ckey == key;
51          }
52          mm->unreserve(prev);
53          prev = curr;
54          curr = next;
55      }
56  }
57  return false;
58 }
```

FIGURE 19.4 Traversing a nonblocking linked list with safe reclamation.