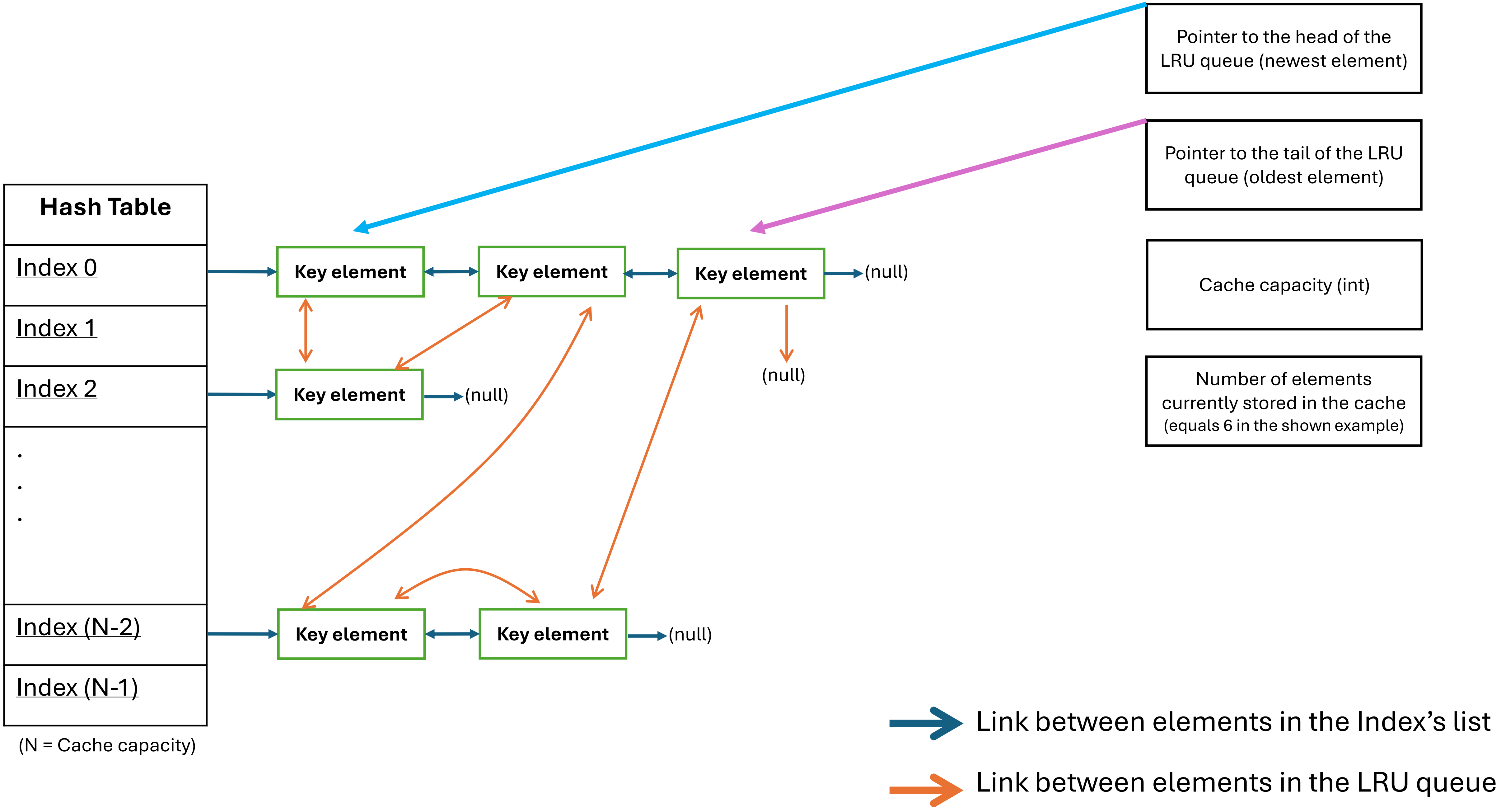
Time Based LRU Cache



Hash Table

Index 0

Index 1

Index 2

.
.
.

Index (N-2)

Index (N-1)

(N = Cache capacity)

Key element

Key element

Key element

(null)

Key element

(null)

(null)

Key element

Key element

(null)

Pointer to the head of the LRU queue (newest element)

Pointer to the tail of the LRU queue (oldest element)

Cache capacity (int)

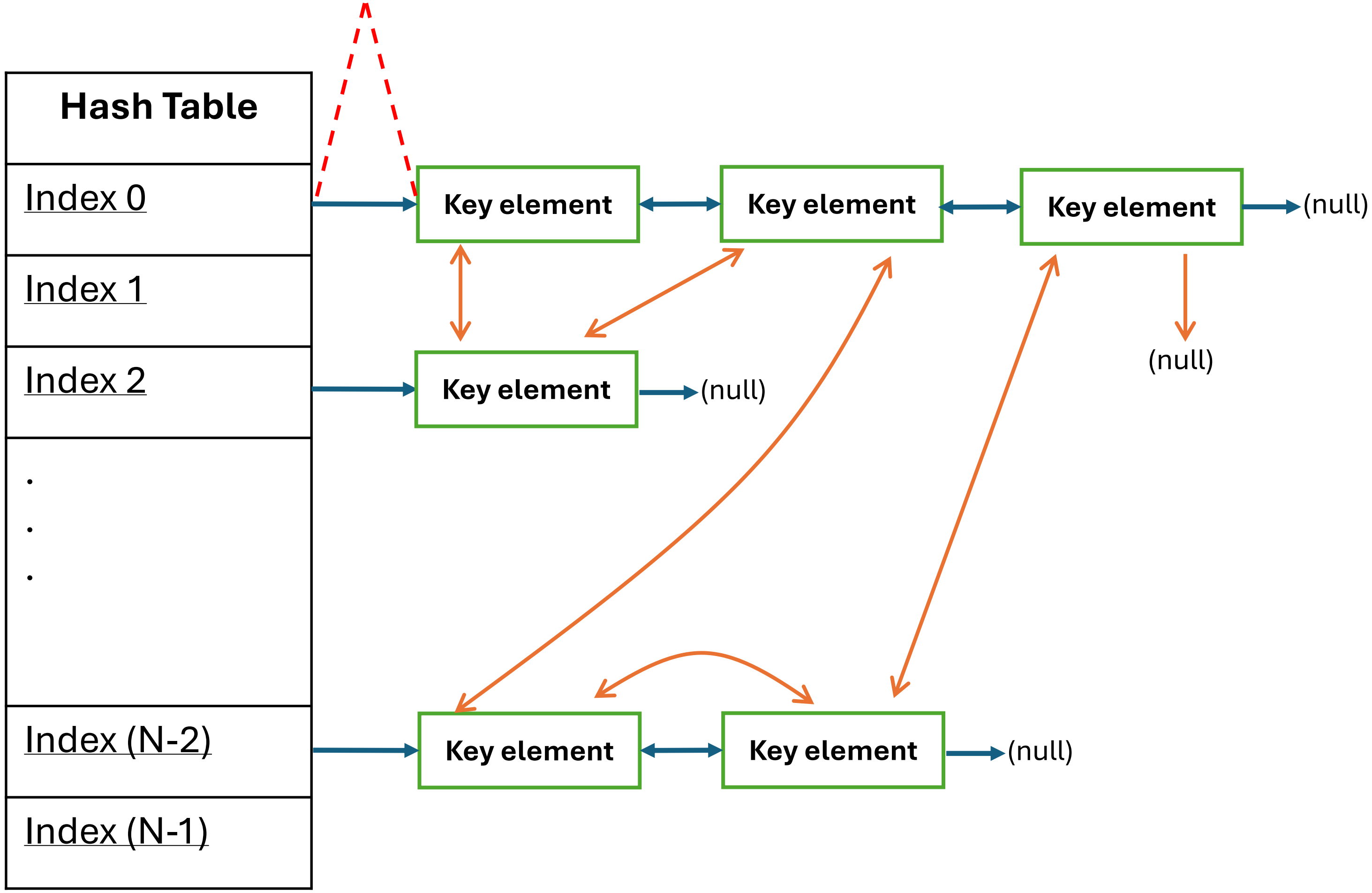
Number of elements currently stored in the cache (equals 6 in the shown example)

Link between elements in the Index’s list

Link between elements in the LRU queue

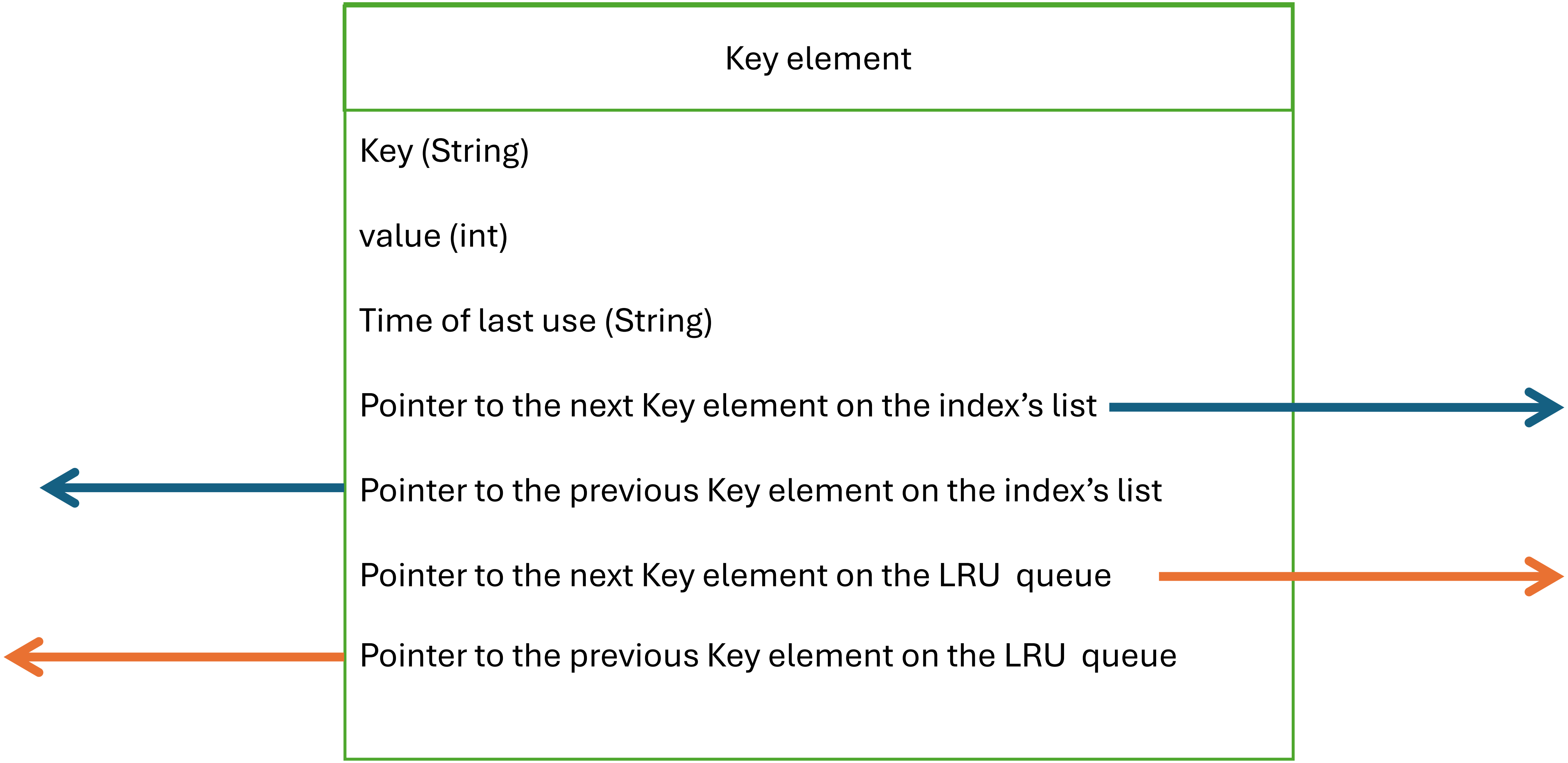
The Hash Table

Each cell (index) in the hash table holds a pointer to the head oh it’s list of elements



(N = Cache capacity)

The Key Element



Key element

Key (String)

value (int)

Time of last use (String)

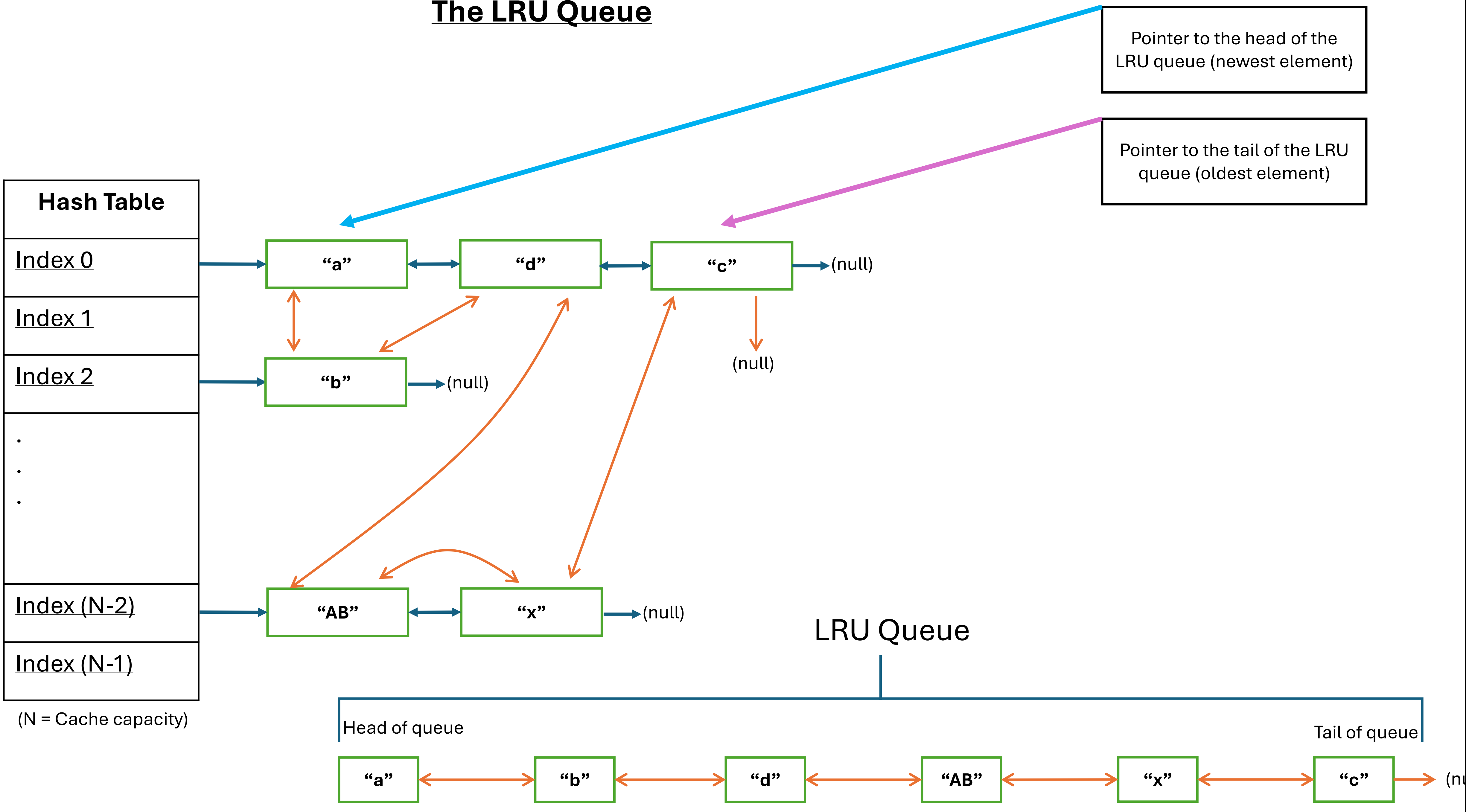
Pointer to the next Key element on the index’s list

Pointer to the previous Key element on the index’s list

Pointer to the next Key element on the LRU queue

Pointer to the previous Key element on the LRU queue

The LRU Queue



Pointer to the head of the LRU queue (newest element)

Pointer to the tail of the LRU queue (oldest element)

(N = Cache capacity)

Complexity analysis

Time complexity:

Insertion, search, and deletion operations in a hash table have an average time complexity of O(1).

In the Time Based LRU Cache data structure, these operations involve updating a fixed number of pointers, so the time complexity remains O(1).

When the clear_expired() method is called, it searches the LRU queue to find the most recent expired item. Since the LRU queue is implemented as a linked list, this search operation has a time complexity of O(N), where N is the number of elements in the cache.

Space complexity:

The hash table contains O(N) cells, and the cache maintains N elements, each with a unique key. Therefore, the space complexity of this structure is O(N).