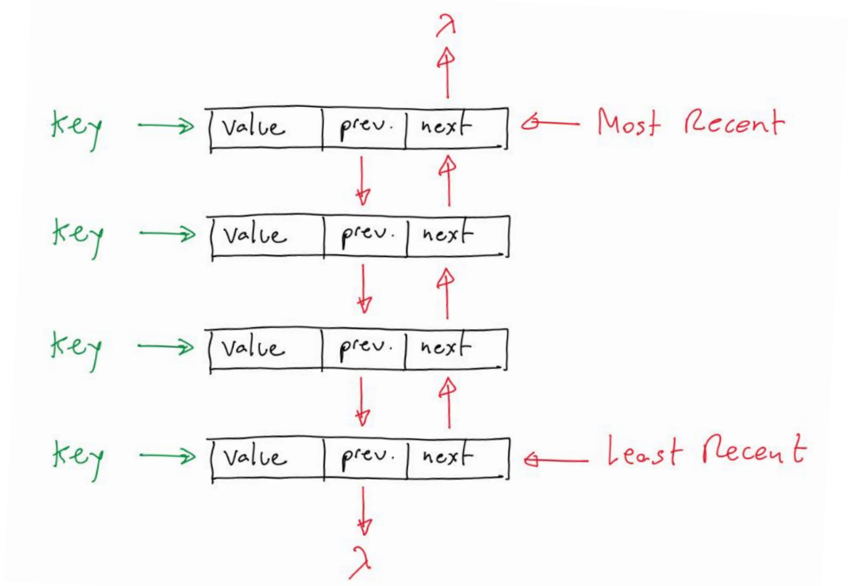
# Problem 1: LRU Cache

The LRU problem was addressed by using a dictionary data type in which a pair key-value corresponds to an address-register memory cache entry.

To keep track of the memory entries accessed more and less recently, each dictionary value carries, in addition to the actual cached value, two reference fields that store the dictionary key of two other pseudo adjacent registers: one less recently accessed, and another one accessed more recently.



The problem input size ***n*** will consist of a given LRU size as a parameter. All registers together form a chain that resembles a linked list. Although, the ***previous*** and ***next*** fields do not store references (or names) to other objects. They simply carry the dictionary keys for the "adjacent" registers in sequence from the least recently accessed entry (tail end) and most recently accessed entry (head end).

At each cache memory access, ***get()*** or ***set()*** methods, the positional reference fields are updated to relocate that register as the topmost recently accessed. If the ***set()*** method results in a cache miss, the least recently used register is deleted to make room for the new value which is also included as the most recently access record. There is no need for list traversal to find the LRU register or to rearrange data sequence upon each memory access. The algorithm takes constant time and to keep track of the access sequence which configures a ***O(1)*** time complexity.

This approach also leverages from the dictionary datatype ***O(1)*** time complexity for data search, insertion and deletion.

Regarding memory usage, the data register implemented for this algorithm uses a constant number of fields for each cache entry, so it has a ***O(n)*** space complexity.