Cache Oblivious Linked List

Your goal is to develop a cache-oblivious data structure for maintaining a linked list of elements with a single finger (pointer to one of the elements) that supports very fast motion of the finger and insertion/deletion at the node pointed to by the finger. More precisely, you should support the following operations:

- \texttt{next}(): Move the finger to the next element and return it.
- \texttt{prev}(): Move the finger to the previous element and return it.
- \texttt{insert}(x): Insert element \(x\) immediately after the finger.
- \texttt{delete}(): Remove the finger element, and move the finger to the previous element.

For simplicity, assume that the linked list always starts with a special undeletable element called the \textit{head}, and assume that initially the list has no other elements. The operations \texttt{insert}(x), and \texttt{delete}() should cost amortized \(O(1)\) memory transfers each, and the operations \texttt{next}() and \texttt{prev}() should cost an amortized \(O(1/B)\) memory transfers each. Your data structure must be cache-oblivious and occupy \(O(N)\) space, where \(N\) is the current number of elements in the list.

Technical Notes

- If \texttt{prev} or \texttt{next} tries to go beyond the first or last element, respectively, assume that the finger does not move.
- Note that the number of memory transfer time of \texttt{next} and \texttt{prev} are subconstant amortized.
- Because your solution must be cache-oblivious, you do not know the value of \(B\), yet you must achieve the necessary bounds in terms of \(B\).
- You can assume that you can allocate an array of size \(K\) (initialized to the value 0) in \(O(K/B)\) memory transfers. The total space of your data structure (which must be \(O(N)\)) is then the sum of the array sizes.