# 6.851 Advanced Data Structures (Spring'14)

Prof. Erik Demaine  TAs: Timothy Kaler, Aaron Sidford

Problem 4  *Due: Monday, Mar. 10*

Be sure to read the instructions on the assignments section of the class web page.

### 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:

- `next()`: Move the finger to the next element and return it.
- `prev()`: Move the finger to the previous element and return it.
- `insert`($x$): Insert element $x$ immediately after the finger.
- `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 *head*, and assume that initially the list has no other elements. The operations `insert`($x$), and `delete()` should cost amortized $O(1)$ memory transfers each, and the operations `next()` and `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 `prev` or `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 `next` and `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.