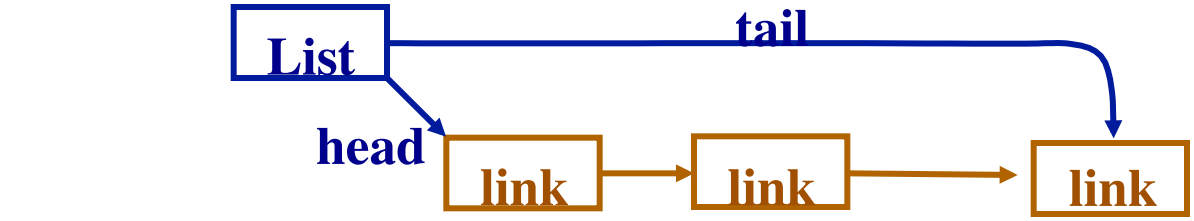
Linked List Implementation of the Deque

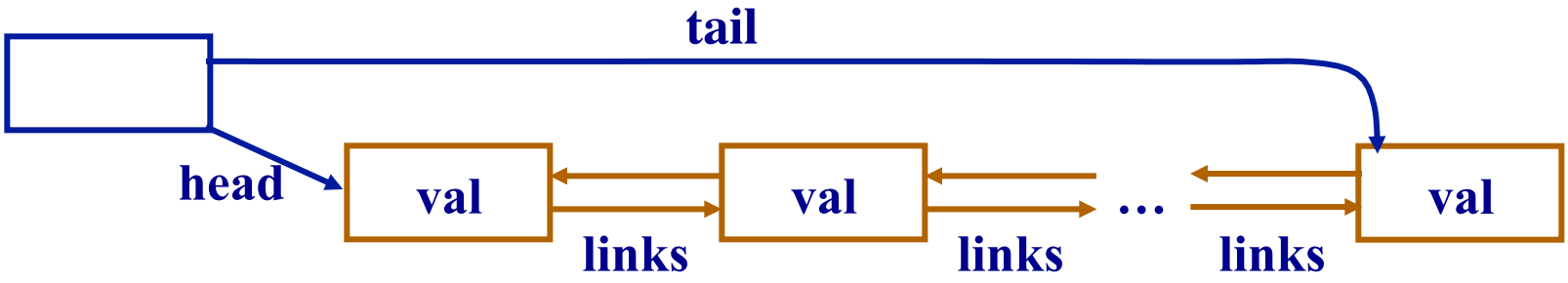
Deque Interface (Review)

```
int      isEmpty();

void    addFront(TYPE val);   // Add value at front of deque.
void    addBack (TYPE val);   // Add value at back of deque.

void removeFront();           // Remove value at front.
void removeBack ();           // Remove value at back.

TYPE       front();           // Get value at front of deque.
TYPE        back();           // Get value at back of deque.
```

Linked List Deque

- What if we want to add and remove elements from both front and back?

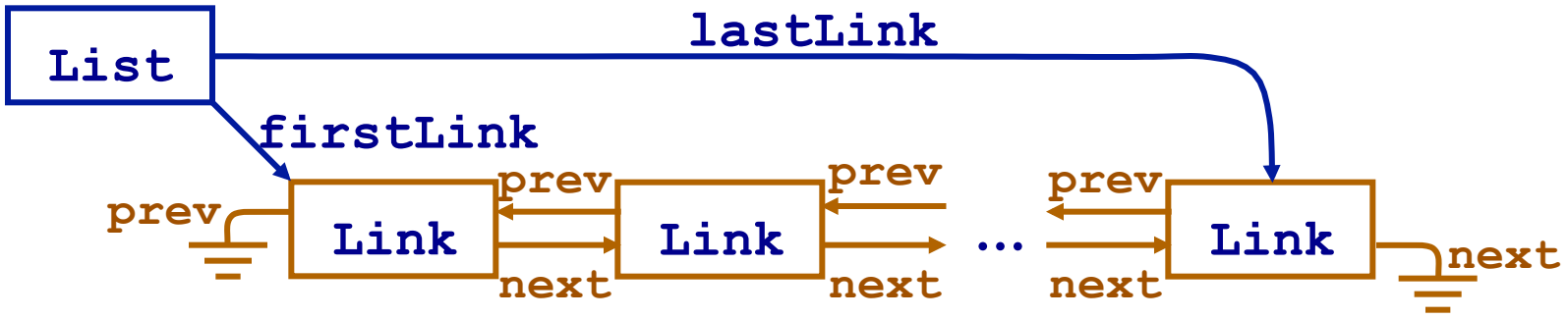




Modification #3: Double Links

- Point forward to the **next** element
- Point backwards to the **prev**ious element

```
struct DLink {
  TYPE           val;
  struct DLink *next;   /* Link to prev node. */
  struct DLink *prev;   /* Link to next node. */
};
```

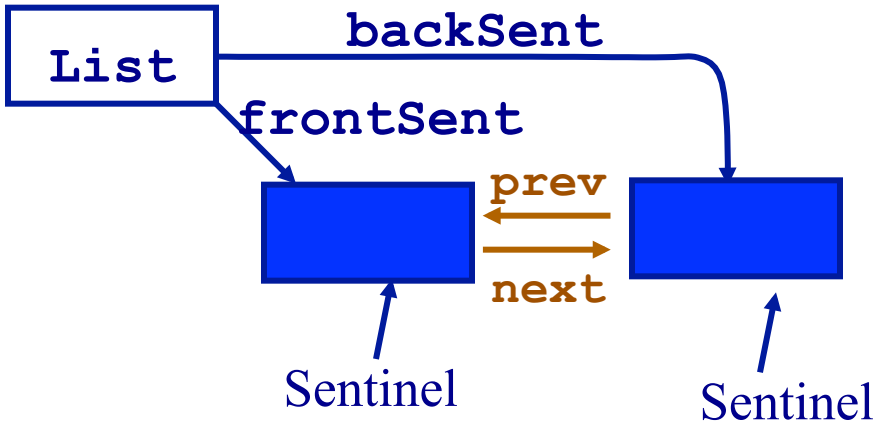


linkedListDeque Struct

```
struct linkedList {
  int size;
  struct dlink * frontSentinel;
  struct dlink * backSentinel;
};
```

linkedListDequeInit

```
void LinkedListInit (struct linkedList *q) {
   q->frontSentinel = malloc(sizeof(struct dlink));
   assert(q->frontSentinel != 0);
   q->backSentinel = malloc(sizeof(struct dlink));
   assert(q->backSentinel);
   q->frontSentinel->next = q->backSentinel;
   q->backSentinel->prev = q->frontSentinal;
   q->size = 0;
}
```

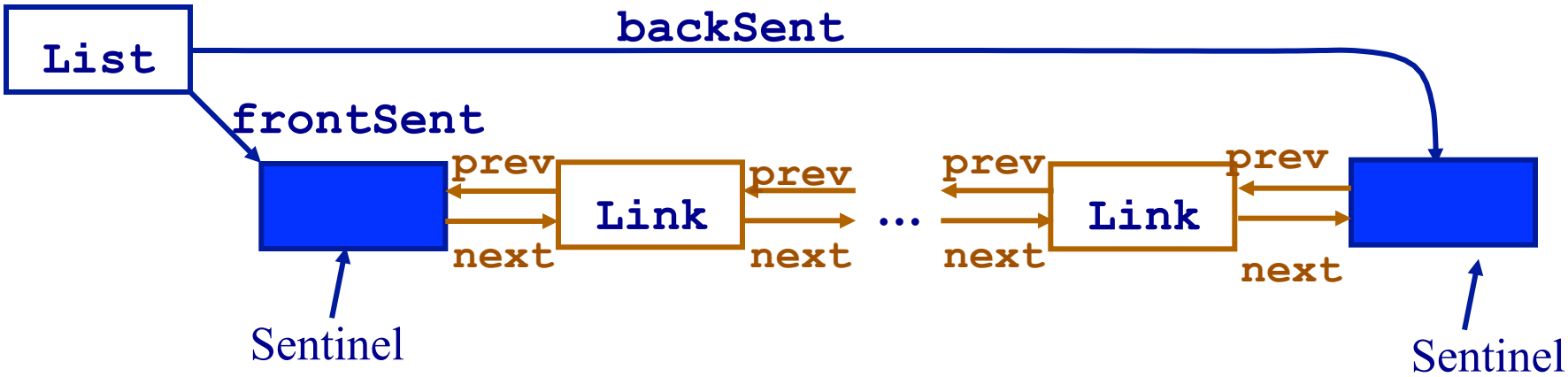


Advantage of Sentinels

- Consider a deque, with two sentinels:
  - Pointer to front sentinel: **frontSent**
  - Pointer to back sentinel: **backSent**

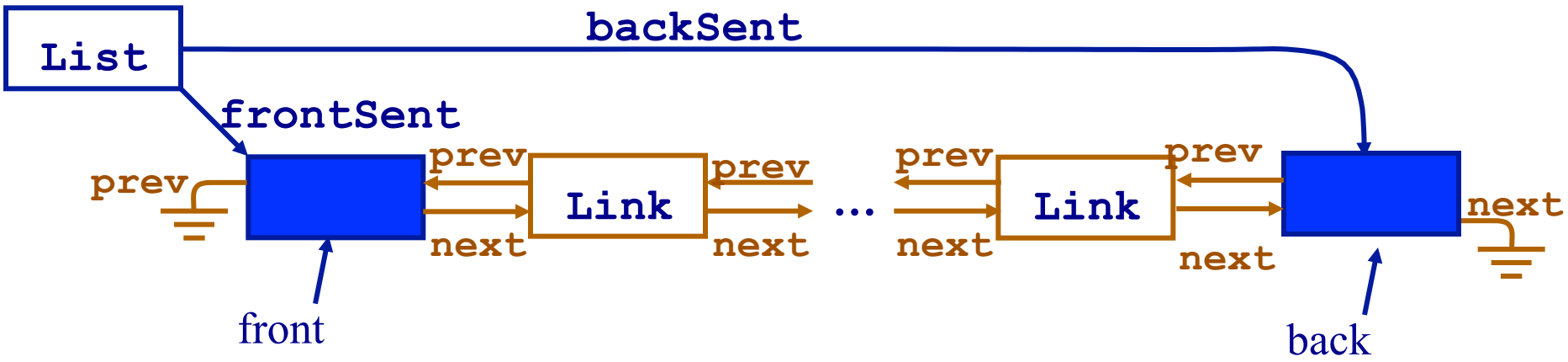
This is similar to most standard library Deque implementations (Java LinkedList)

- Add to front and add to back are now special cases of more general "add before" operation



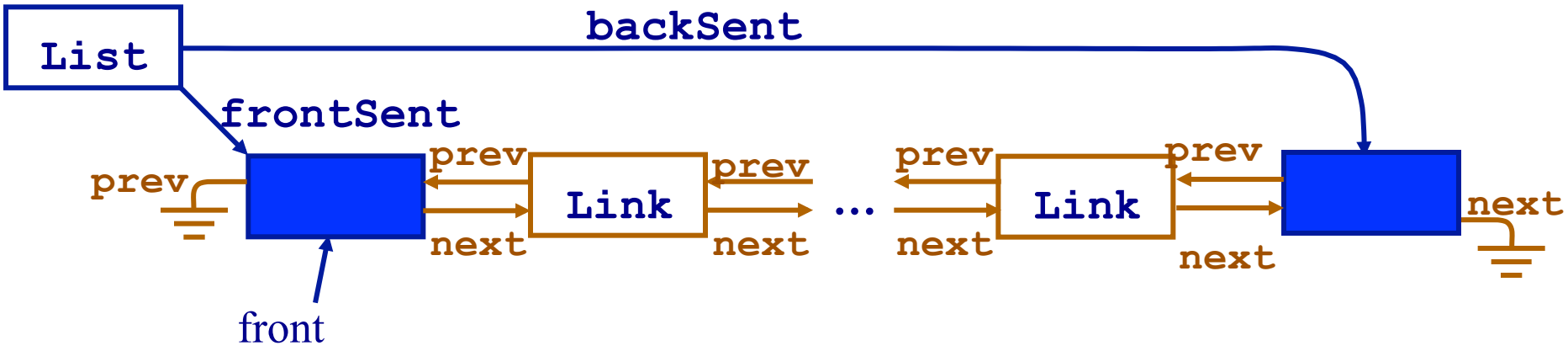
Adding to the LL Deque

```
void addBackListDeque(struct ListDeque *q,TYPE
  val) {
  _addBefore(q->lastLink, val);
}

void addFrontListDeque(struct ListDeque *q,TYPE
  val) {
  _addBefore(q->firstLink->next, val);
}
```



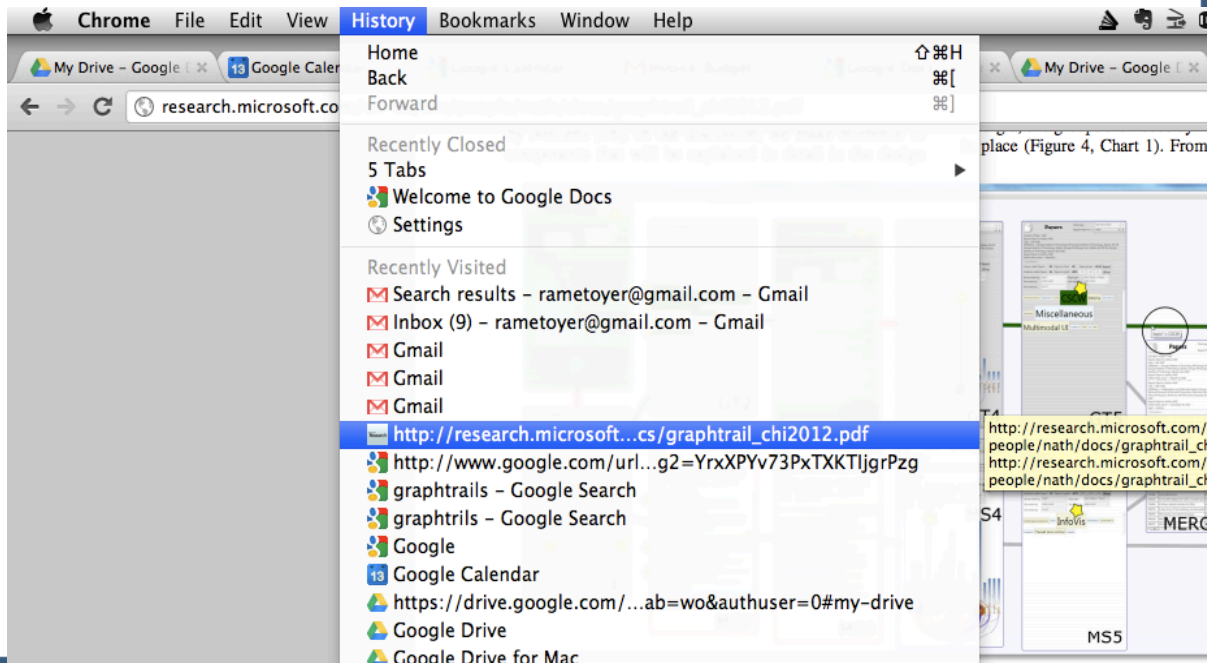
Removing from the LL Deque

```
void removeFirstListDeque(struct ListDeque *q) {
  assert(!isEmptyListDeque(q));

  _removeLink(q->firstLink->next);
}

void removeLastListDeque(struct ListDeque *q) {
  assert(!isEmptyListDeque(q));

  _removeLink (q->lastLink->prev);
}
```



Deque: DynArray vs. Linked List

- Remember: Finite length undo
- Which would you use?
- Which would support this kind of history operation?



Your Turn…

Worksheet #19: _addBefore, _removeLink

	DynArrDeque best, ave, worst	LLDeque best, ave, worst
addLast	O(1),O(1+), O(N)	O(1),O(1),O(1)
removeLast	O(1), O(1),O(1)	O(1),O(1),O(1)
addFirst	O(1),O(1+), O(N)	O(1),O(1),O(1)
removeFirst	O(1),O(1),O(1)	O(1),O(1),O(1)