CS 261: Data Structures

Linked Lists

List Stack

Dynamic Array -- Problems

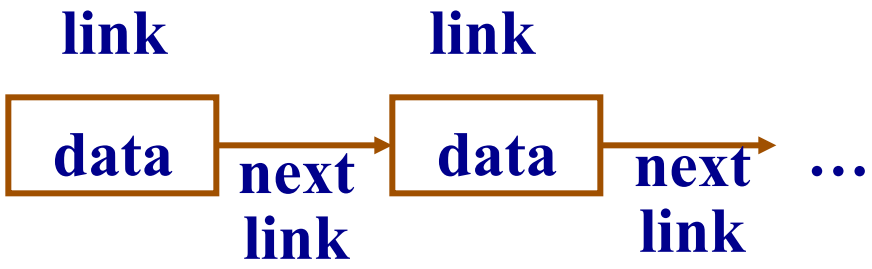
- Data kept in a single large block of memory
- Often more memory used than necessary
  - especially when repeatedly growing and shrinking the dynamic array

Linked List

- A good alternative
- The memory use is always proportional to the number of elements in the collection

Characteristics of Linked Lists

- Elements are held in objects called **Links**
- Links are 1-to-1 with data elements, allocated and released as necessary



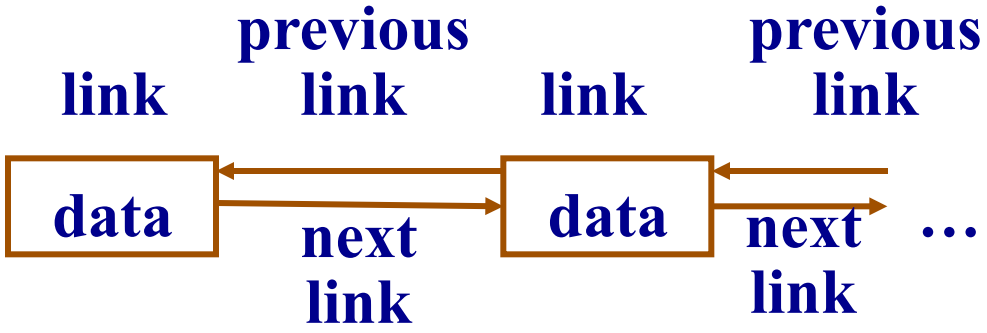
Single and Double Linked Lists

Each link points to

only next link → single linked list

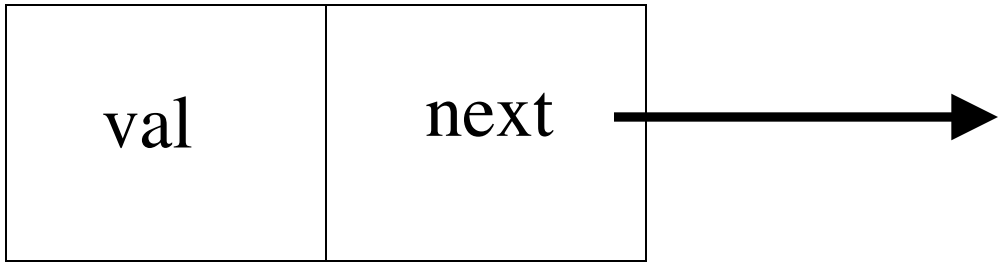
next and previous links → double linked list

in the sequence



Link Structure

```
struct Link {
    TYPE  value;
    struct Link *next;
};
```



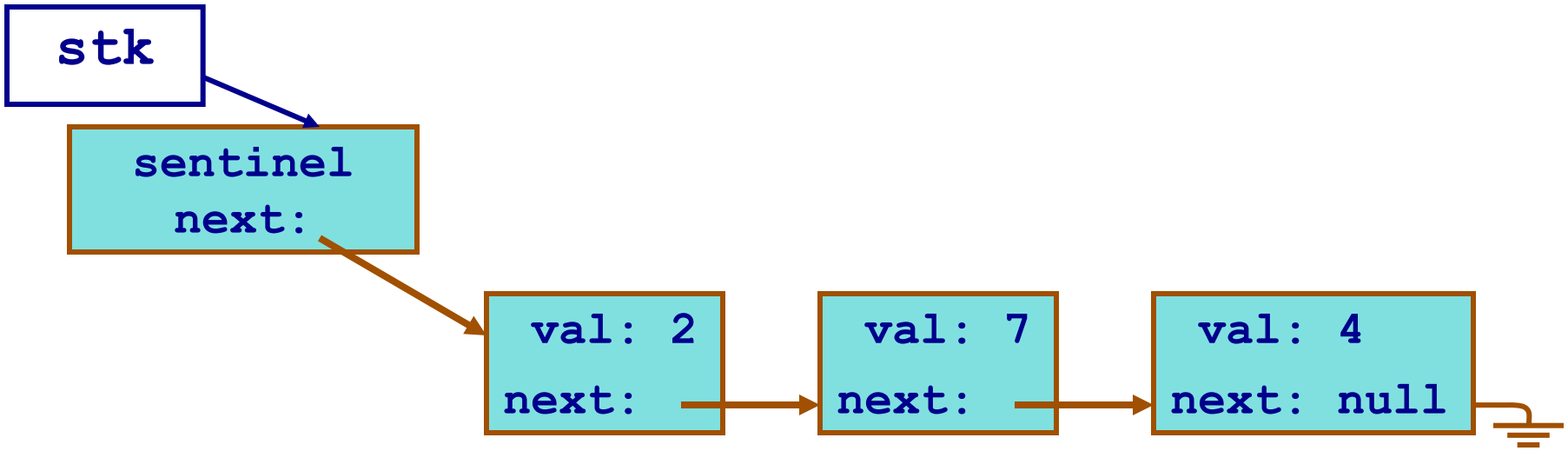
Elements of Linked Lists

- **Sentinel** -- special link for start or end
- Points to
  - first **or** last link only (single linked list)
  - first **and** last link (double linked list)

List Stack

- Sentinel points to the first element
- Sentinel points to NULL if stack empty
- Add or remove elements only from front
- Allow only singly linked list
- Can access only first element

List Stack



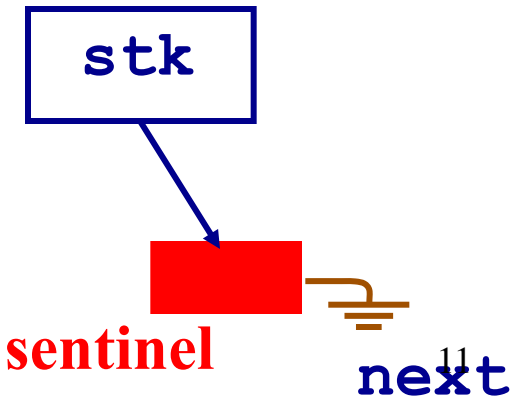
Implementation of List Stack

```
struct Link {
    TYPE  value;
    struct Link * next;
};

struct ListStack {
    struct Link * sentinel;
};
```

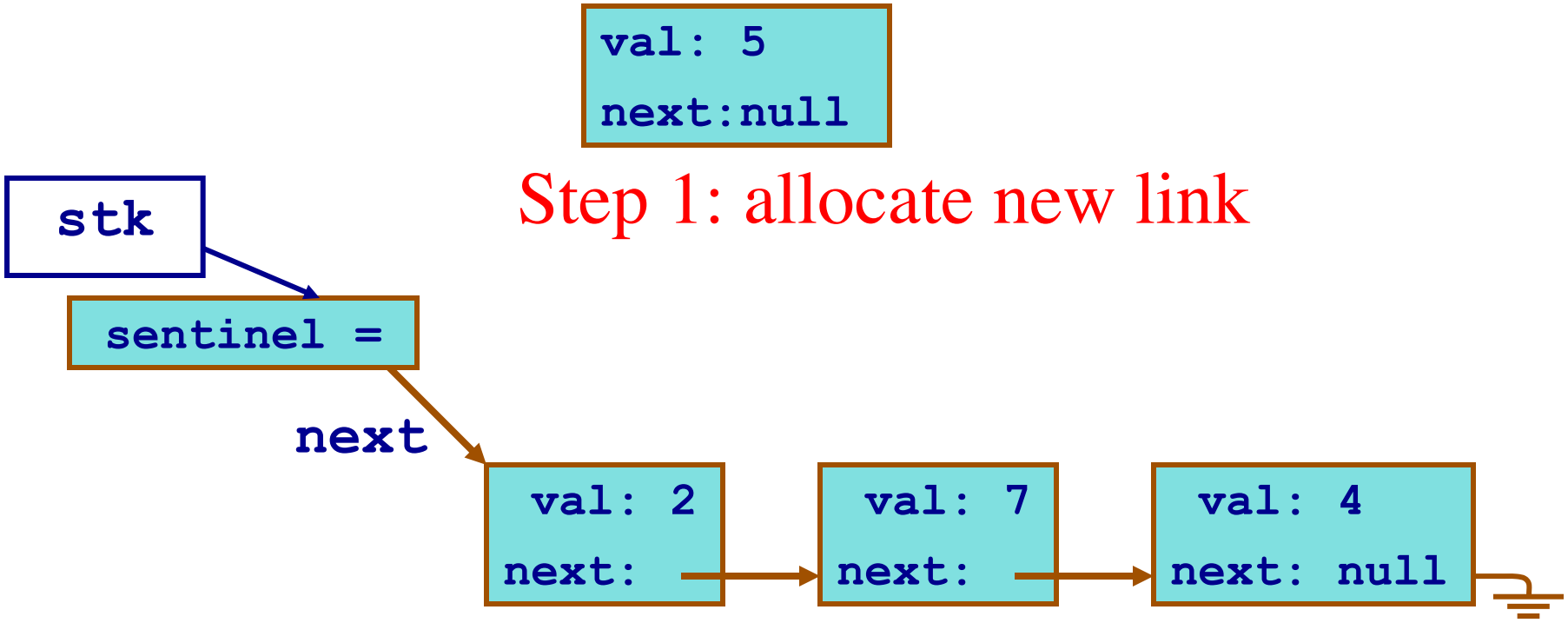
How to initialize List Stack?

InitStack

```
void InitStack(struct ListStack * stk){
    /*initialize the sentinel*/
    struct Link *sentinel =
        (struct Link *)malloc(sizeof(struct Link));

     assert(sentinel != 0);

    /*linked list is empty*/
     sentinel->next = NULL;
     stk->sentinel = sentinel;
}
```

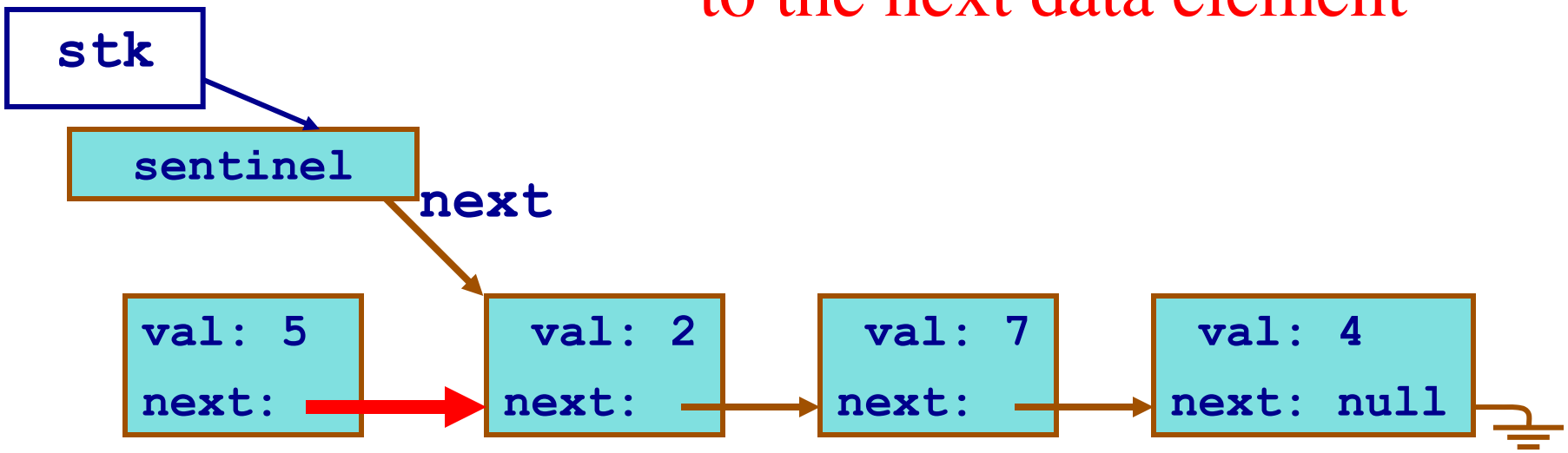


Push List Stack: 3 Steps



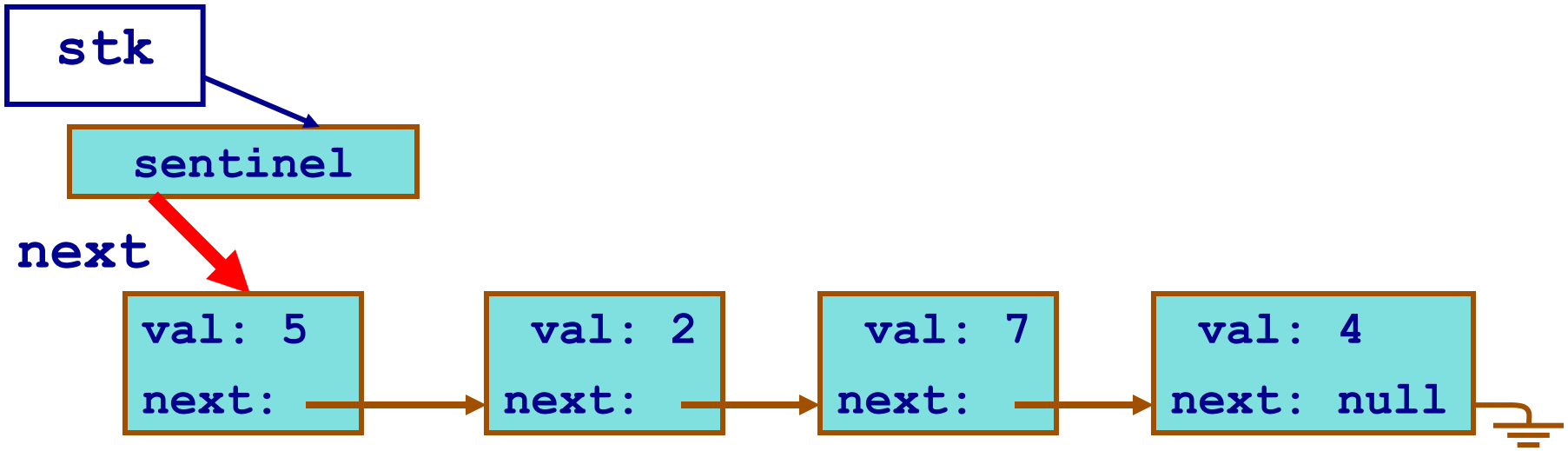
Push List Stack: 3 Steps

Step 2: link the new element to the next data element



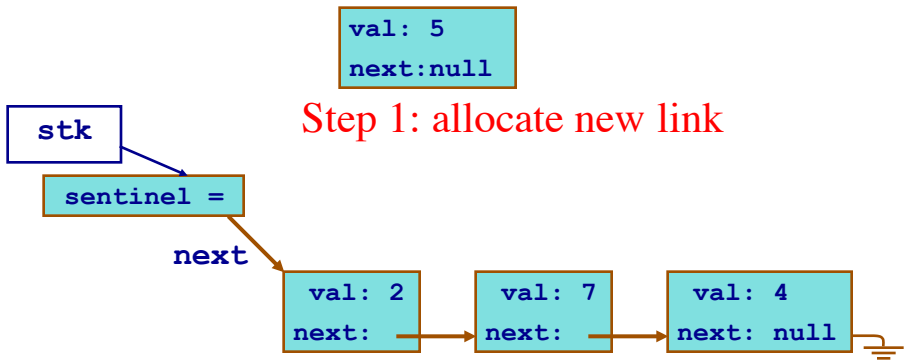
Push List Stack: 3 Steps

Step 3: add the new element to the top



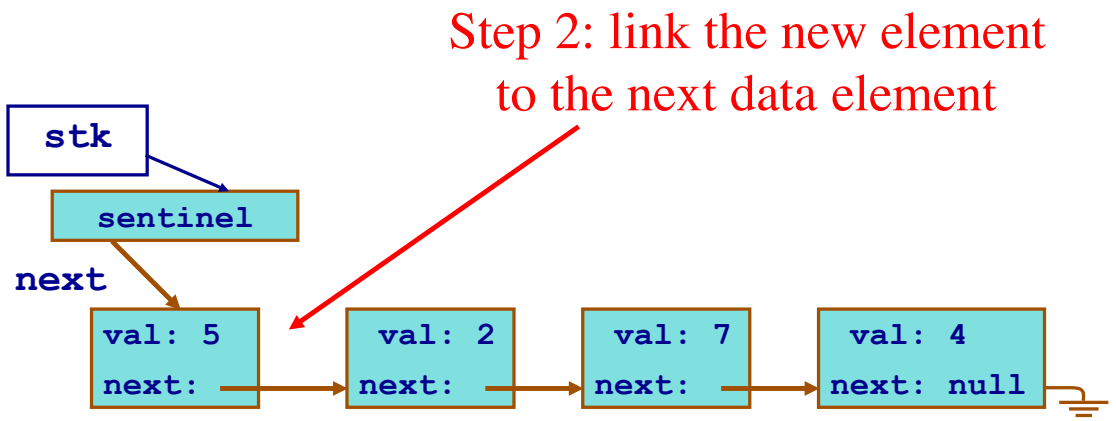
Push List Stack: 3 Steps

```
void pushStack (struct listStack *stk, TYPE val){
    struct Link * new =
        (struct Link *) malloc(sizeof(struct Link));
    assert (new != 0);
    new->value = val;


}
```



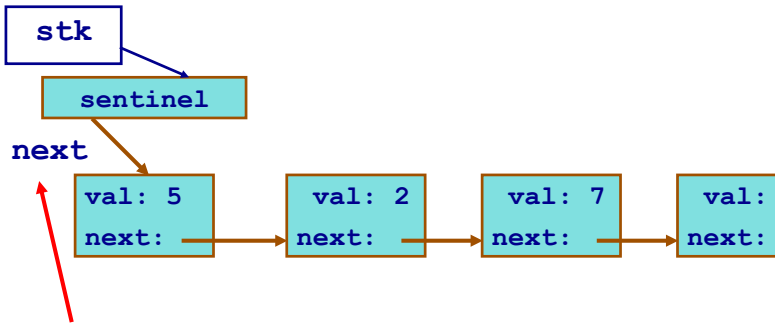
Push List Stack: 3 Steps

```
void pushStack (struct listStack *stk, TYPE val){
    struct Link * new =
        (struct Link *) malloc(sizeof(struct Link));
    assert (new != 0);
    new->value = val;
    new->next =  stk->sentinel->next;

}
```



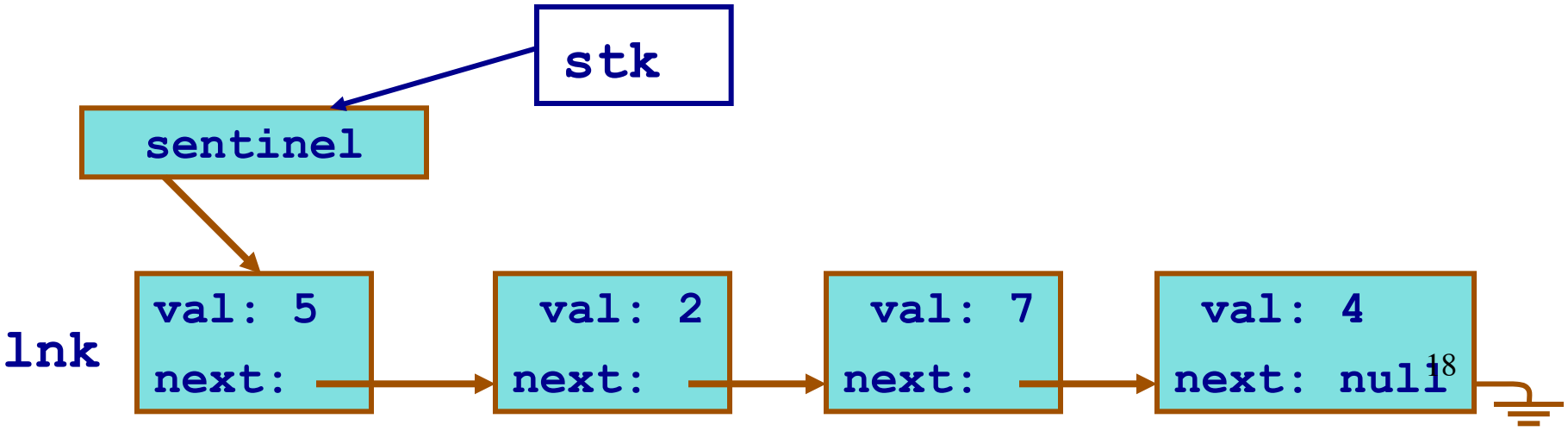
Push List Stack: 3 Steps

```
void pushStack (struct listStack *stk, TYPE val){
    struct Link * new =
        (struct Link *) malloc(sizeof(struct Link));
    assert (new != 0);
    new->value = val;
    new->next =  stk->sentinel->next;
    stk->sentinel->next = new;
}
```



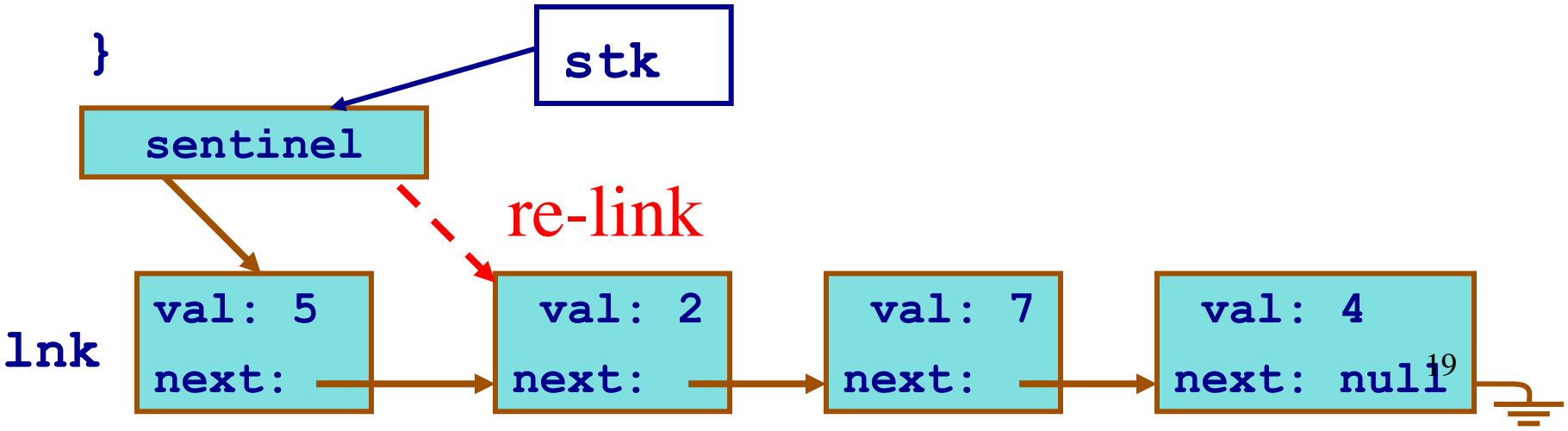
PopStack

```
/*move the top to the next element*/
void PopStack (struct ListStack *stk) {
    struct Link * lnk = stk->sentinel->next;
    if(lnk!=NULL){ /*the top element exists*/


}
```



PopStack

```
/*move the top to the next element*/
void PopStack (struct ListStack *stk) {
    struct Link * lnk = stk->sentinel->next;
    if(lnk!=NULL){ /*the top element exists*/
      stk->sentinel->next = lnk->next;
      free(lnk);
    }
}
```



topStack, isEmpty...

- Should be done on your own
- Worksheet 17

List Stack vs. Dyn. Array Stack

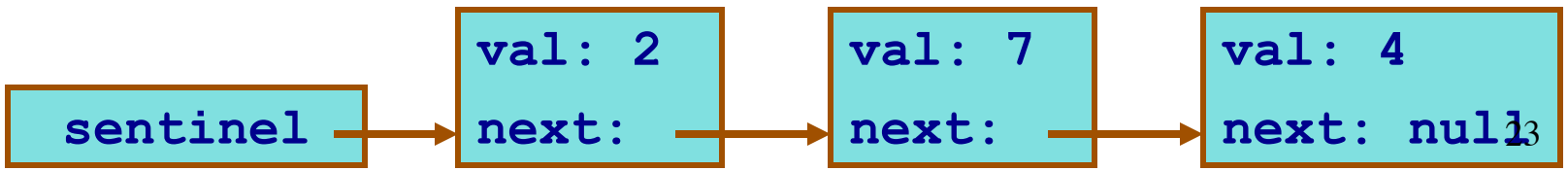
	List	Dyn. Array
pushStack	O(1)	O(1)
popStack	O(1)	O(1)
topStack	O(1)	O(1)

List Bag

- Init, Add operations are similar to List Stack
- Contains and Remove operations are tricky
- How to patch up links after removing an element?

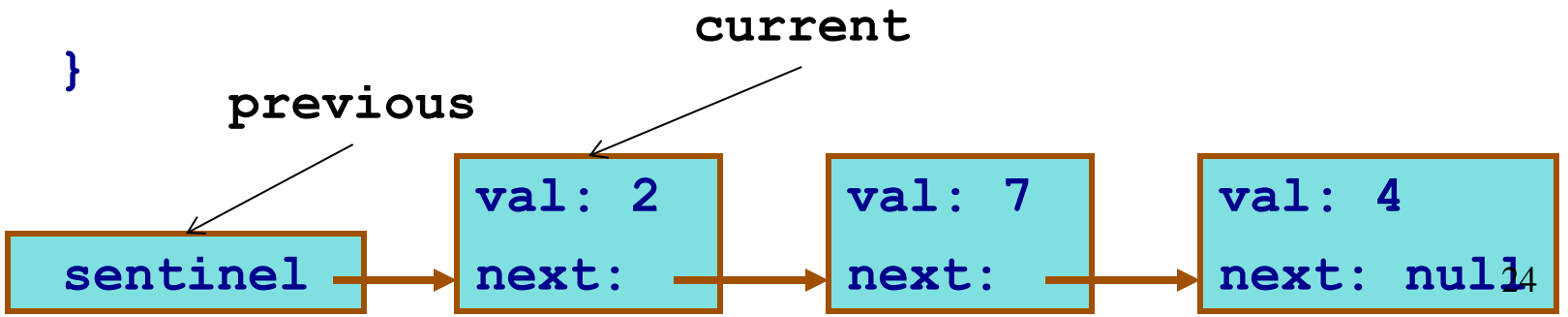
Remove

```
void removeListBag(struct ListBag *b, TYPE val) {

}
```



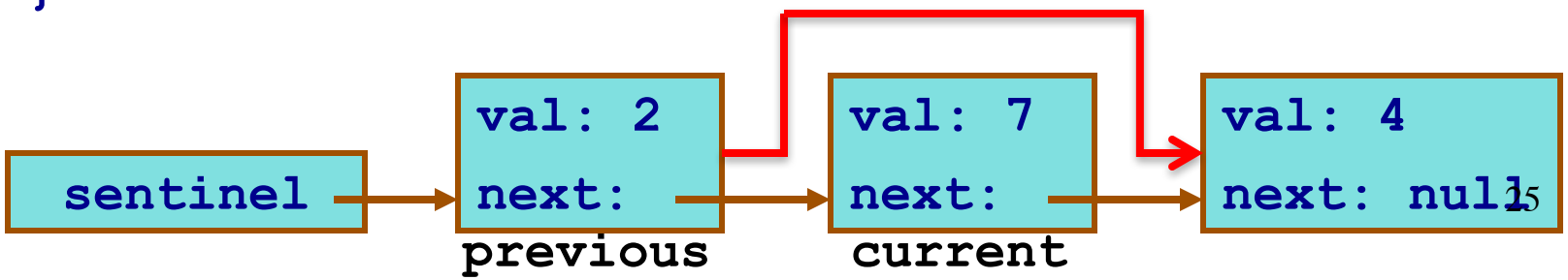
Remove

```
void removeListBag(struct ListBag *b, TYPE val) {
    struct Link *previous = b->sentinel;
    struct Link *current = b->sentinel->next;
    while (current != NULL){

    }
}
```



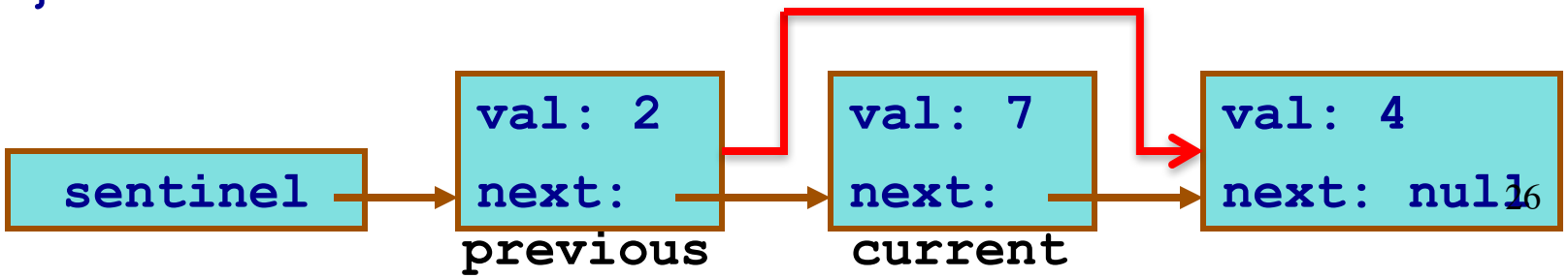
Remove

```
void removeListBag(struct ListBag *b, TYPE val) {
    struct Link *previous = b->sentinel;
    struct Link *current = b->sentinel->next;
    while (current != NULL){
      if (EQ(current->value,val)) {
        previous->next = current->next;

      }
    }
}
```



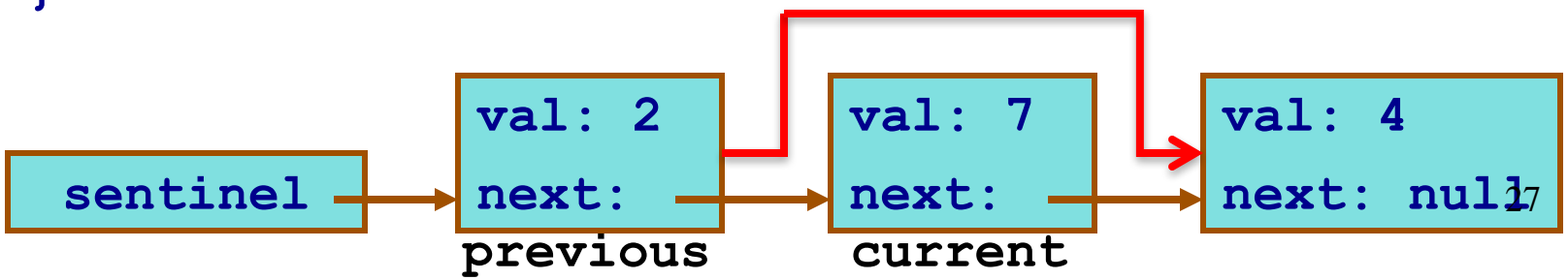
Remove – First Occurrence

```
void removeListBag(struct ListBag *b, TYPE val) {
    struct Link *previous = b->sentinel;
    struct Link *current = b->sentinel->next;
    while (current != NULL){
      if (EQ(current->value,val)) {
        previous->next = current->next;
        free(current);
        return; /*removes only the first occurrence*/
      }
      previous = current;
      current = current->next;
    }
}
```



Remove -- All Occurrences

```
void removeListBag(struct ListBag *b, TYPE val) {
    struct Link *previous = b->sentinel;
    struct Link *current = b->sentinel->next;
    while (current != NULL){
      if (EQ(current->value,val)) {
        previous->next = current->next;
        free(current);
        current = previous;
      }
      previous = current;
      current = current->next;
    }
}
```



When you find it

- When you find the element to be deleted, what does previous point to?

- What if the element to be deleted is at the front of the list? Does this matter?