CS 261: Data Structures

Sorted Linked Lists

Complexity – Lists and Arrays

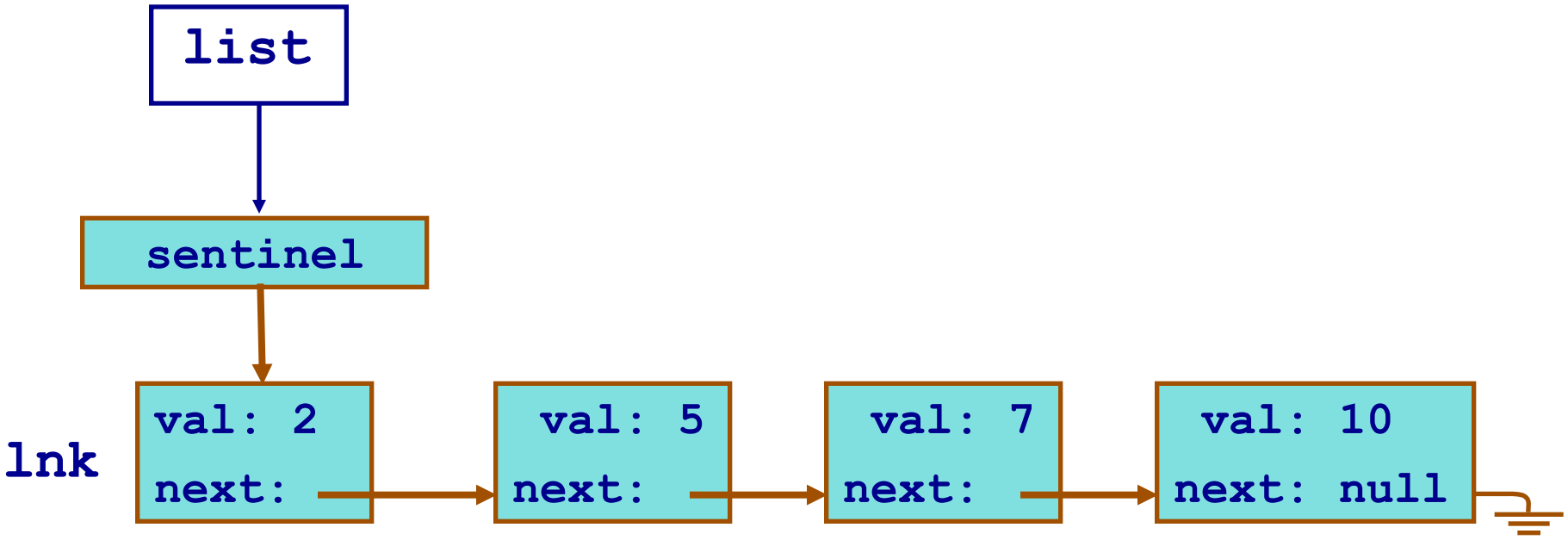
- Unordered linked have:
  - Fast Add operation: O(1)
  - Slow Search (Contains): O(n)
- Sorted Arrays have:
  - Slow Add operation: O(n)
  - Fast Search O(log n)

Complexity – Lists and Arrays

- Unordered linked have:
  - Fast Add operation: O(1)
  - Slow Search (Contains): O(n)
- Sorted Arrays have:
  - Slow Add operation: O(n)
  - Fast Search O(log n)

What about sorted lists?

Sorted Linked List



Operations for Sorted Linked Lists

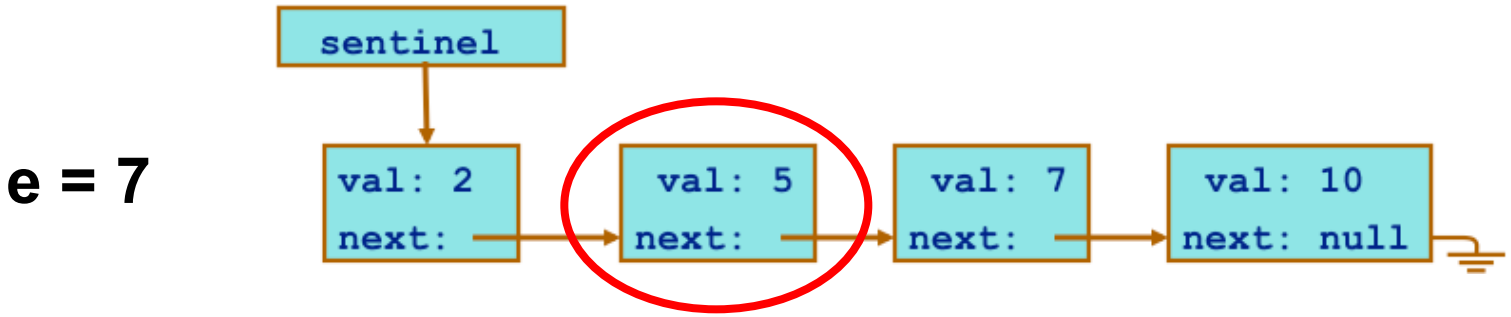
- Add:
  - find the correct location,
  - add the new element
- Contains:
  - find the correct location,
  - check if the element is in the list
- Remove:
  - find the correct location,
  - remove the element if found in the list

Sorted List Structure Definition

```
struct list {
    struct link * Sentinel;
    int size;
};
```

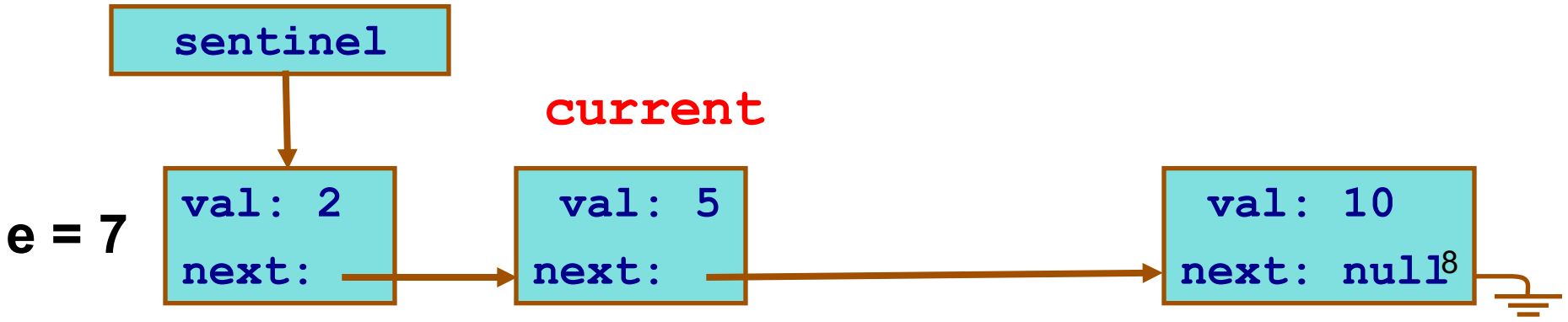
Find an Element in a Sorted List

```
struct link * slideRightSortedList
                        (struct link *current, TYPE e){
    assert(current);
    while ((current->next != 0)
            && LESS_THAN(current->next->value, e))
        current = current->next;
    return current; /* Returns the link RIGHT BEFORE */
}
```



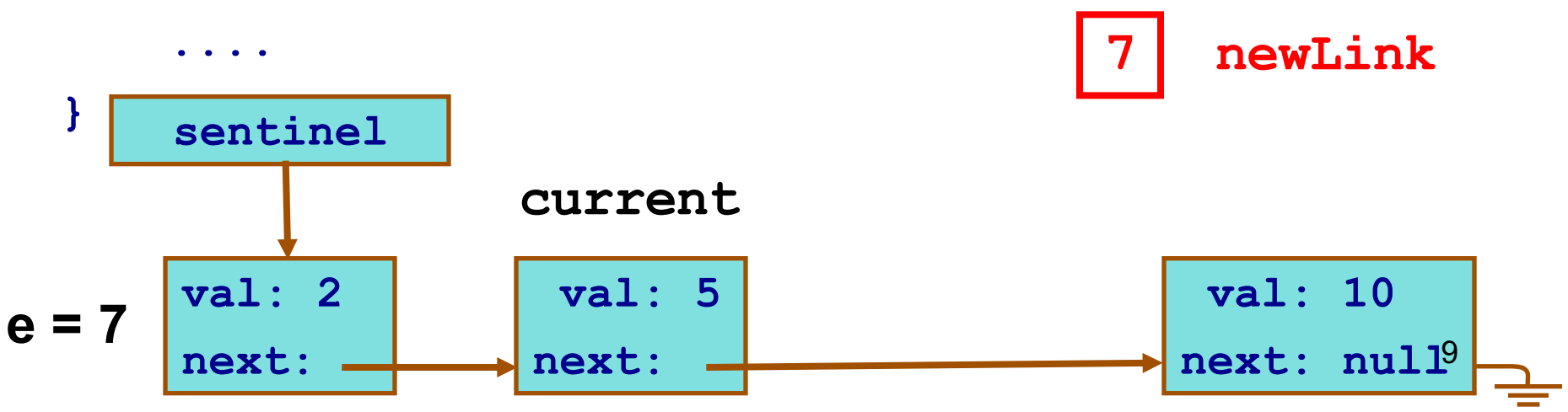
Add Sorted List

```
void addSortedList (struct list* lst, TYPE e) {
    assert(lst);
    struct link * current =
          slideRightSortedList(lst->Sentinel, e);

      ...

}
```



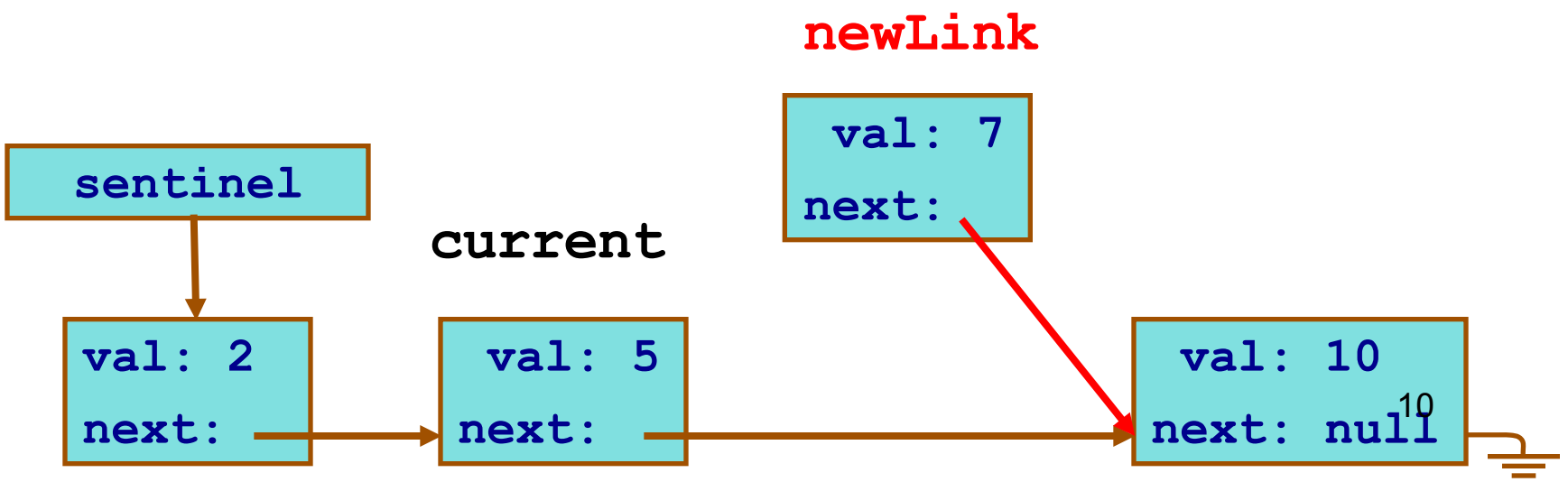
Add Sorted List

```
void addSortedList (struct list* lst, TYPE e) {
    assert(lst);
    struct link * current =
          slideRightSortedList(lst->Sentinel, e);
    struct link * newLink =
          (struct link *) malloc(sizeof(struct link));
    assert (newLink != 0);
    newLink->value = e;
     ....
}
```



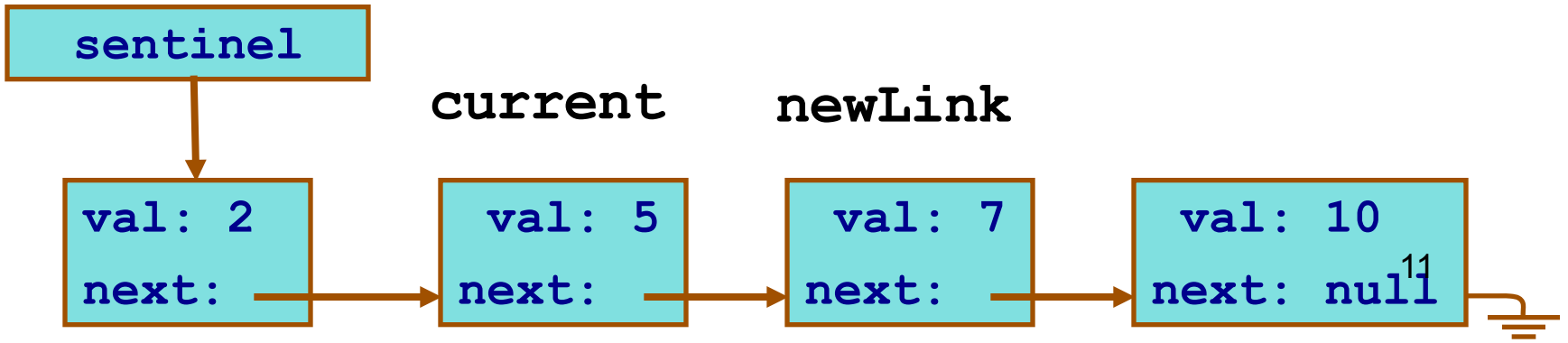
Add Sorted List

```
void addSortedList (struct list* lst, TYPE e) {
    ...
    newLink->next = current->next;
    /*  For doubly linked lists */
    /* newLink->previous = current; */
    ...

}
```



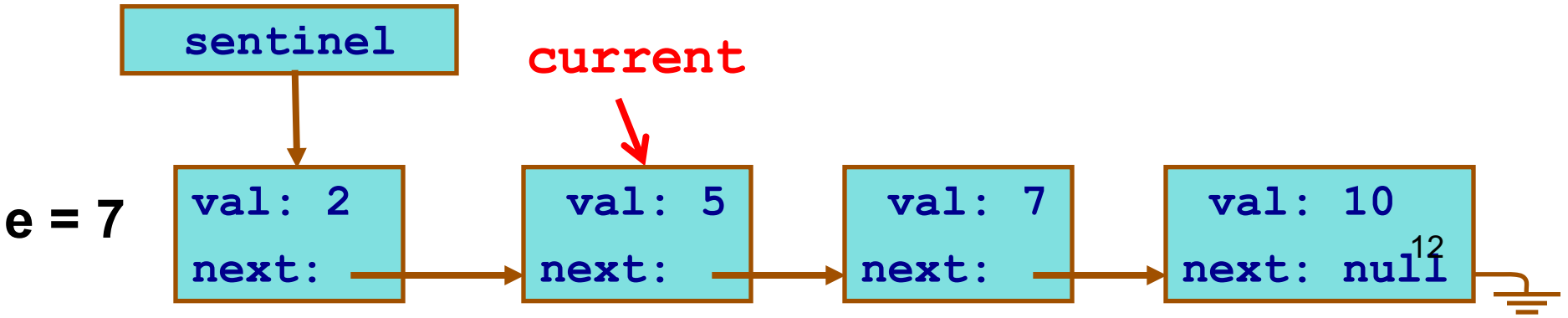
Add Sorted List

```
void addSortedList (struct list* lst, TYPE e) {
    ...
    newLink->next = current->next;
    /*  For doubly linked lists */
    /* newLink->previous = current; */
    /* current->next->previous = newLink; */
    current->next = newLink;
    lst->size++;
}
```



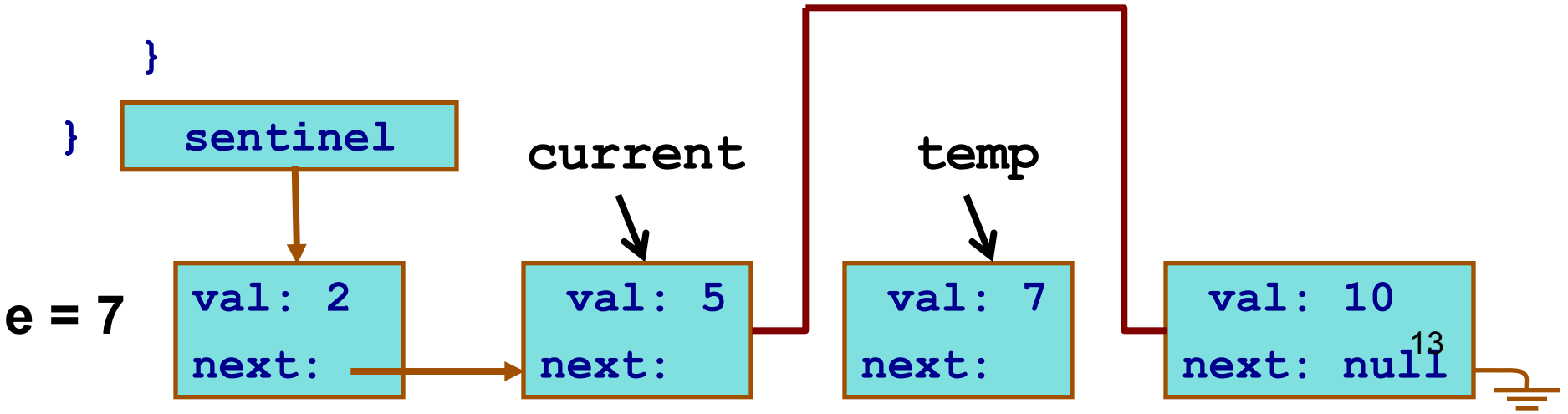
Remove Sorted List

```
void removeSortedList (struct list *lst, TYPE e) {
    struct link * temp; assert(lst);
    struct link * current =
        slideRightSortedList(lst->Sentinel, e);
    if ((current->next != 0) &&
                    (EQ(current->next->value, e))){
        ....
    }
}
```



Remove Sorted List

```
void removeSortedList (struct list *lst, TYPE e) {
    …if ( ... ){
        temp = current->next
        current->next = current->next->next;
        /* For doubly linked lists */
        /* current->next->previous = current; */
        free(temp);
        lst->size--;
    }
}
```



Sorted Linked List

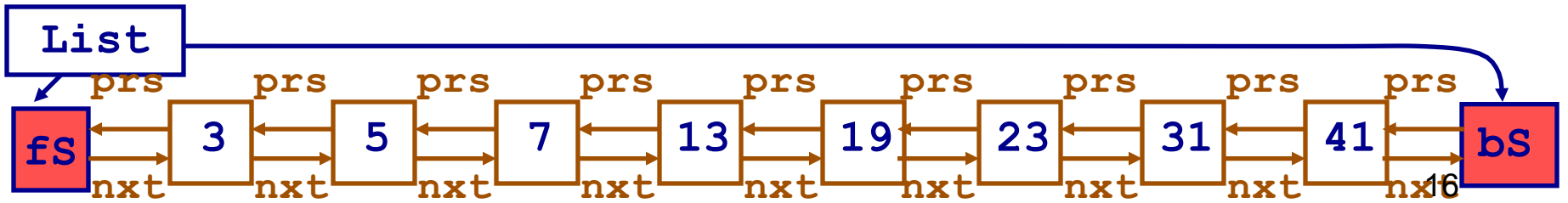
- What is complexity of:
  - search O(??)
  - insertion O(??)
  - removal O(??)
- Because we do not have a direct access to each link in the list

Problem with Sorted List

- What's the use?
- Add, contains, remove → O(n)
- No better than an unsorted list
- **Major problem: sequential access**

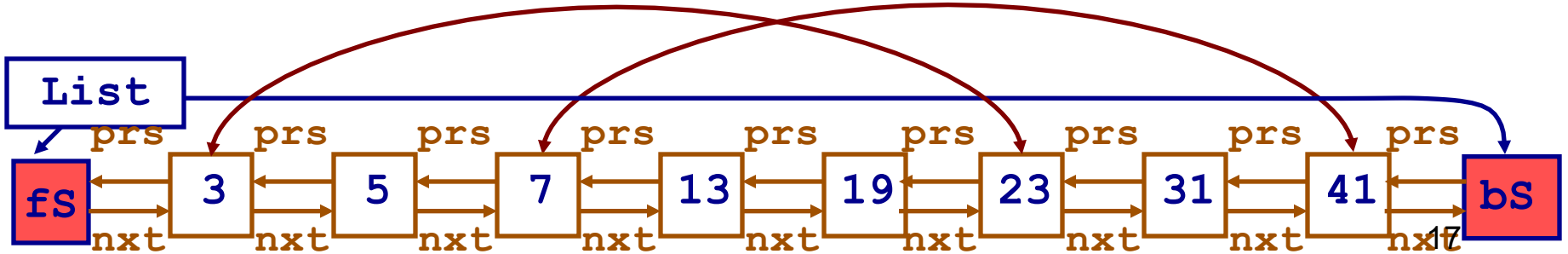
Sorted Linked List

- How to make a sorted linked list have faster operations?



Sorted Linked List

- Should I add more pointers?
  - –E.g., add *log n* pointers



Adding more pointers…

- In theory this would work
- Would give us O(log n) search
- Hard to maintain insertion and removal