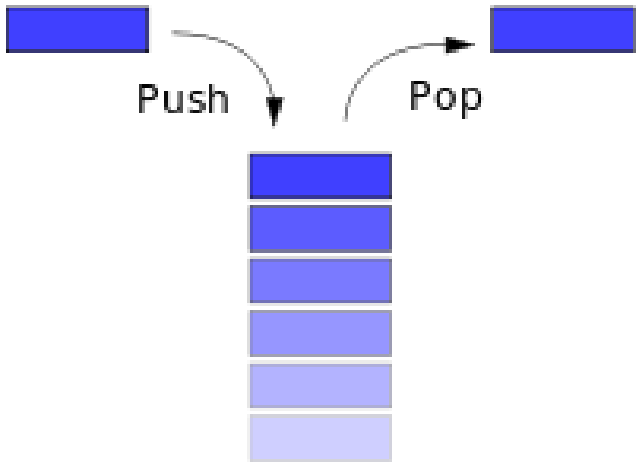
Chapter 12. Fundamental Data Structures

Byoung-Tak Zhang
TA: Hanock Kwak
Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University

http://bi.snu.ac.kr

Stacks

- Collection of elements, with two principal operations
  - push
    - adds an element to the collection
  - pop
    - removes the last element that was added.
- First-In-Last-Out (FILO)



Stacks

- Stack with integer elements
  - `int stack[MAX];`
  - `int top = -1; // -1 means that the stack is empty.`

top = -1

i	stack[i]
6	don't care
5	don't care
4	don't care
3	don't care
2	don't care
1	don't care
0	don't care

Stacks

- push(5);

top = 0

i	stack[i]
6	don't care
5	don't care
4	don't care
3	don't care
2	don't care
1	don't care
0	5

Stacks

- push(7);

top = 1

i	stack[i]
6	don't care
5	don't care
4	don't care
3	don't care
2	don't care
1	7
0	5

Stacks

- push(1);
- push(3);

top = 3

i	stack[i]
6	don't care
5	don't care
4	don't care
3	3
2	1
1	7
0	5

Stacks

- pop();

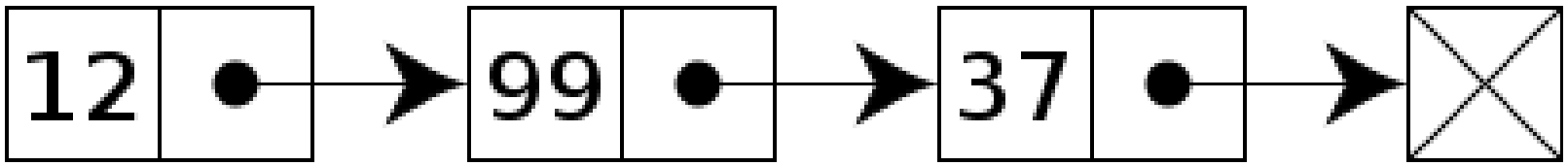
top = 2

i	stack[i]
6	don't care
5	don't care
4	don't care
3	don't care
2	1
1	7
0	5

Linked List

- The linked list is a data structure consisting of a group of nodes which together represent a sequence.

- Each node is composed of data and a reference (pointer).



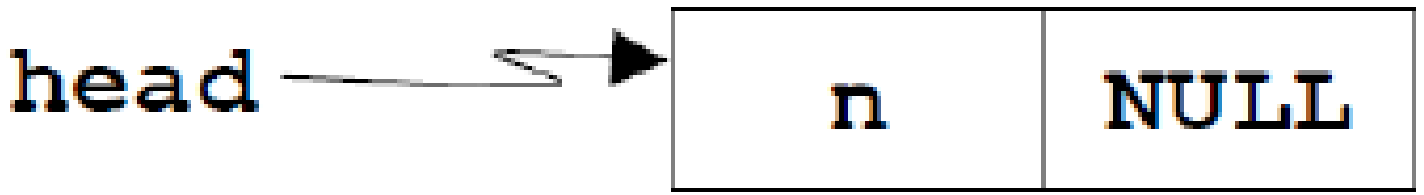
Linked Lists

[list.h]

```
typedef char DATA;
struct linked_list {
   DATA d;
   struct linked_list *next;
};
typedef struct linked_list ELEMENT;
typedef ELEMENT *LINK;
```

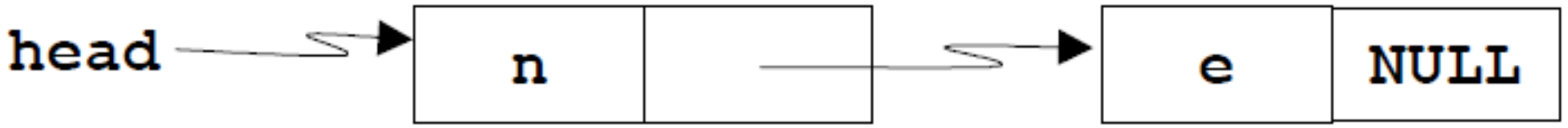
Storing three characters n, e, w

```
LINK head; /* ELEMENT *head */
head = malloc(sizeof(ELEMENT));
head->d = 'n';
head->next = NULL;
```



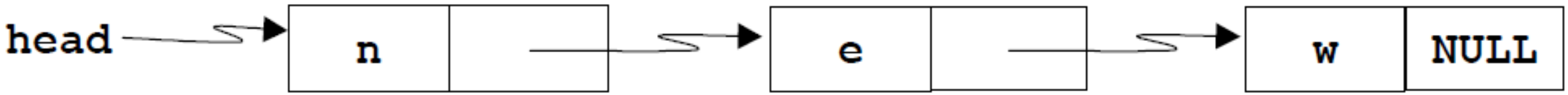
Storing three characters n, e, w

```
head->next = malloc(sizeof(ELEMENT));
head->next->d = 'e';
head->next->next = NULL;
```



Storing three characters n, e, w

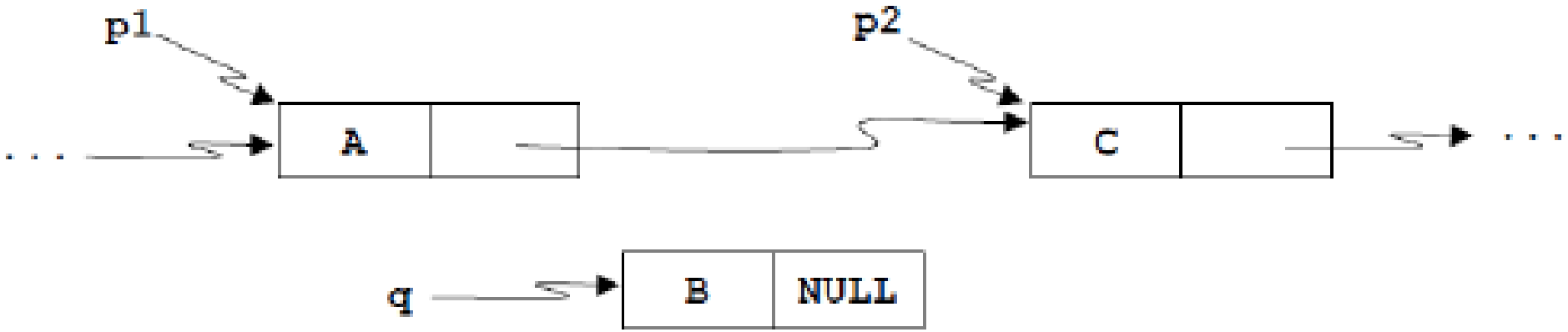
```
head->next->next =
malloc(sizeof(ELEMENT));
head->next->next->d = 'w';
head->next->next->next = NULL;
```

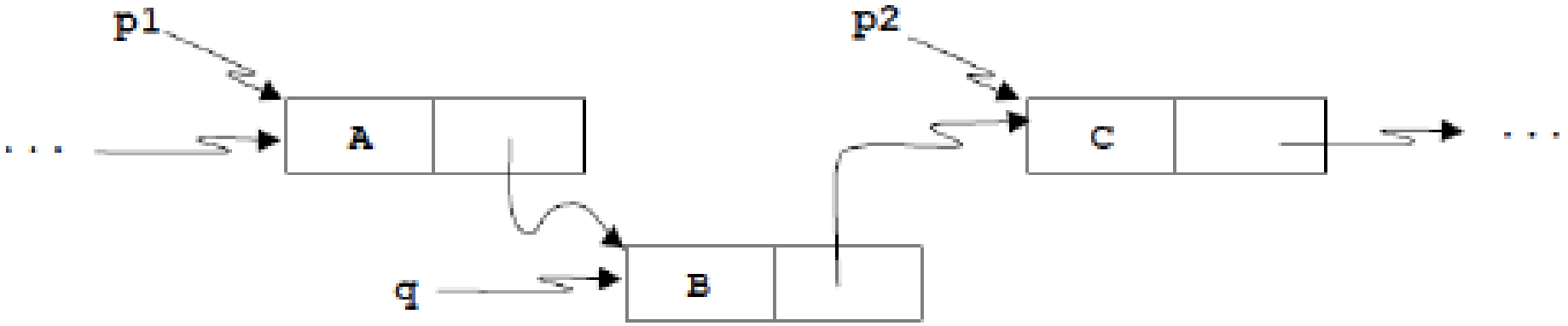


List Operations

- Inserting an element
- Deleting an element
- Looking up an element
- Counting the elements

Inserting an element

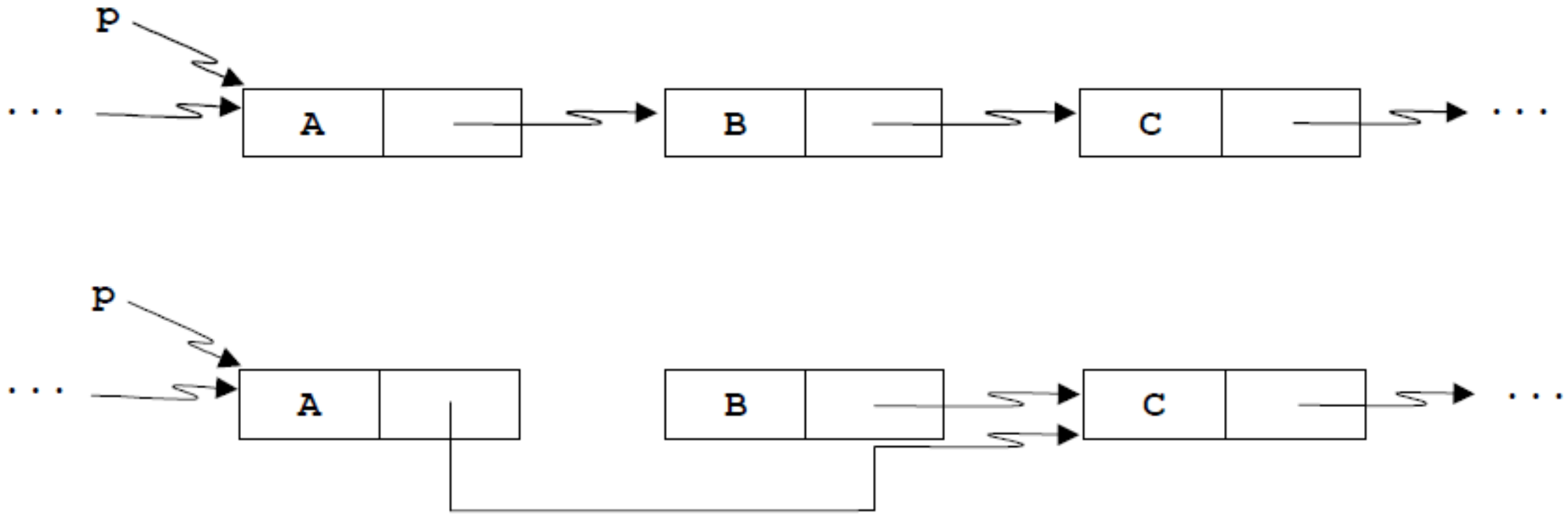




Inserting an element

```
void insert(LINK p1, LINK p2, LINK q)
{
   assert(p1 -> next == p2);
   p1 -> next = q; /* insert */
   q -> next = p2;
}
```

Deleting an element



Deleting an element

```
void delete(LINK p)
{
   p->next=p->next->next;
   /*element containing B becomes a garbage*/
}
```

Looking up an element

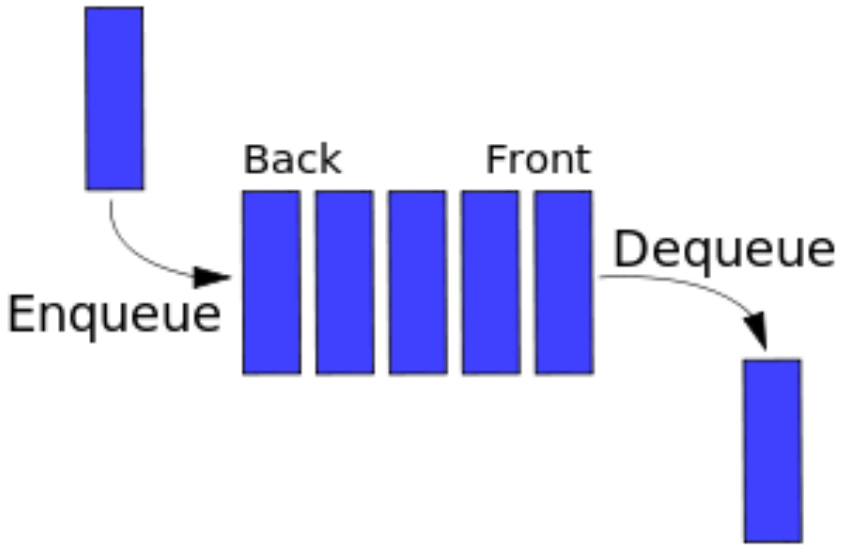
```
/* Looking up an element recursively. */
LINK find(LINK head, DATA d)
{
    if (head == NULL) {
        return 0; // not found
    }
    else if (head->d == d) {
        return head; // found !
    }
    else {
        return find(head -> next); // find recursively
    }
}
```

Counting the elements

```
/* Count a list recursively. */
int count(LINK head)
{
   if (head == NULL) {
      return 0;
   }
   else {
      return (1 + count(head -> next));
   }
}
```

Queues

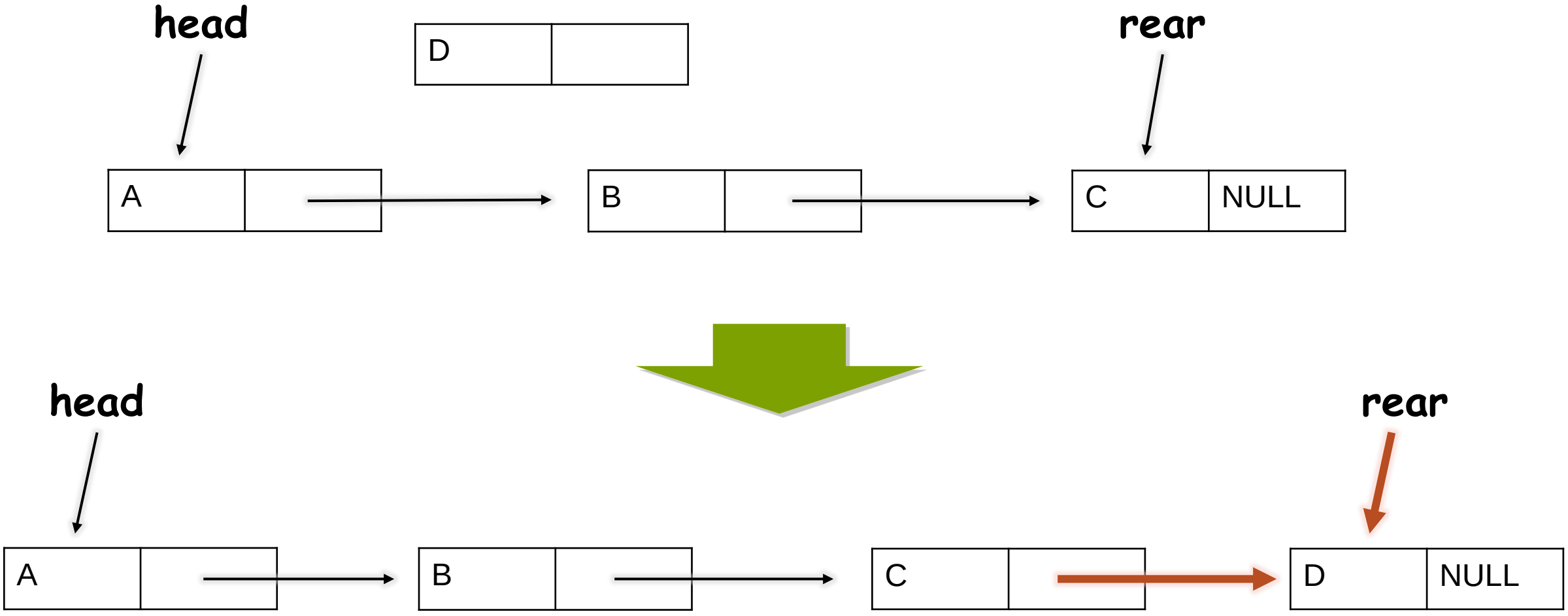
- Collection of elements, with two principal operations
  - enqueue
    - adds an element to the rear-end of the collection
  - dequeue
    - removes the front-end element in the collection.
- First-In-First-Out (FIFO)



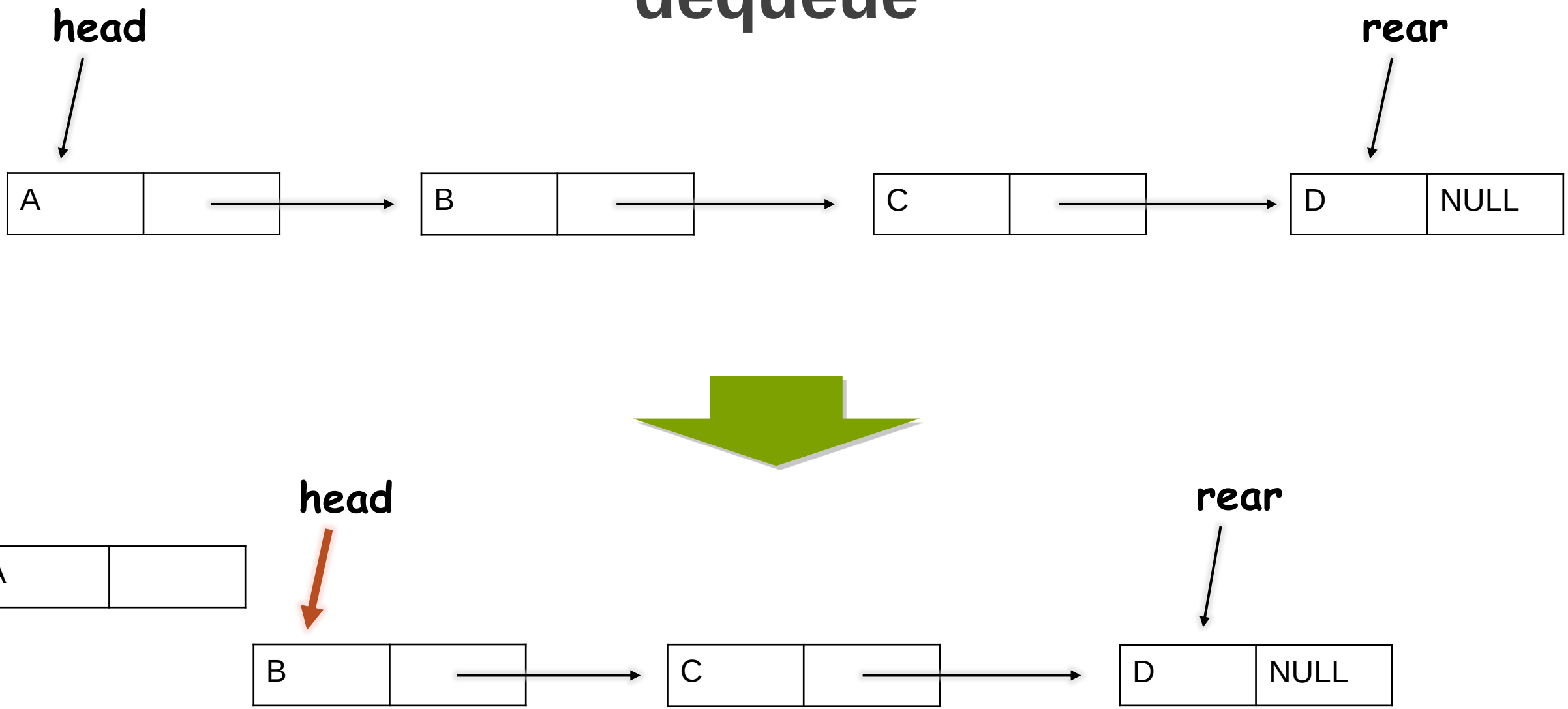
Implementing a Queue with a Linked List

```
LINK head;  // pt. to the front-end element
LINK rear;  // pt. to the rear-end element
```

enqueue



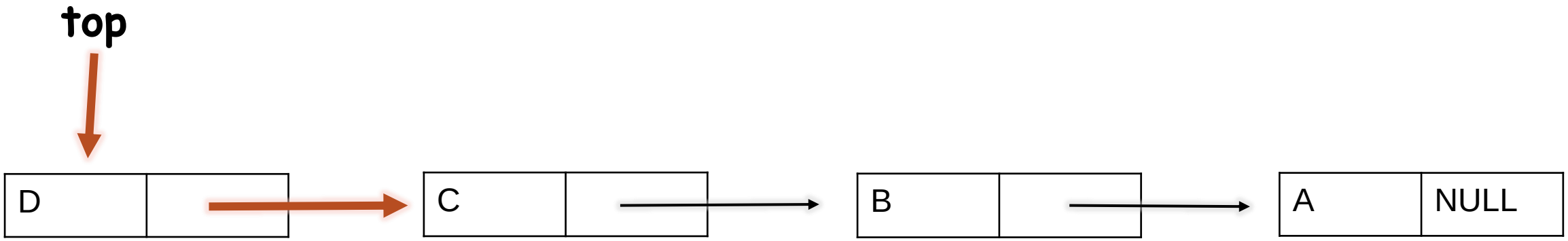
dequeue



Implementing a Stack with a Linked List

```
LINK top;  // pt. to the top element
```

push('D')



pop