Linked Lists

Raghu H V,
C-DAC Bengaluru

Linked Lists

- A ***linked list*** is a linear collection of data elements, called ***nodes***, where the linear order is given by means of ***pointers***.

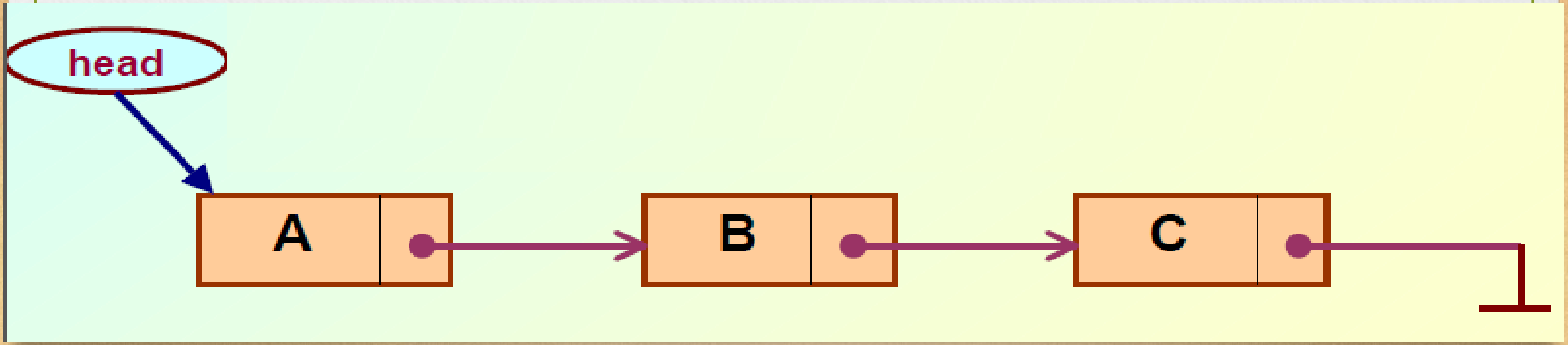
- Each **node** is divided into two parts:
  - The first part contains the ***information*** of the element and
  - The second part contains the address of the next node (***link /next pointer field***) in the list.

- Linked List Operations
  - Insertion
  - Deletion
  - Traversal

Introduction:

A linked list is a data structure which can change during execution.

- –Successive elements are connected by pointers.
- –Last element points to **NULL**.
- –It can grow or shrink in size during execution of a program.
- –It can be made just as long as required.
- –It does not waste memory space.



Introduction:

- **Keeping track of a linked list:**
  - Must know the pointer to the first element of the list (called ***start, head, list*** *etc.*).
- **Linked lists provide flexibility in allowing the items to be rearranged efficiently.**
  - Insert an element.
  - Delete an element.

Arrays Vs Linked Lists

- **Arrays are suitable for:**
  - Inserting/deleting an element at the end.
  - Randomly accessing any element.
  - Searching the list for a particular value.
- **Linked lists are suitable for:**
  - Inserting an element.
  - Deleting an element.
  - In situations where the number of elements cannot be predicted before hand.

Basic Operations on a List

- Creating a list
- Traversing the list
- Inserting an item in the list
- Deleting an item from the list
- Concatenating two lists into one

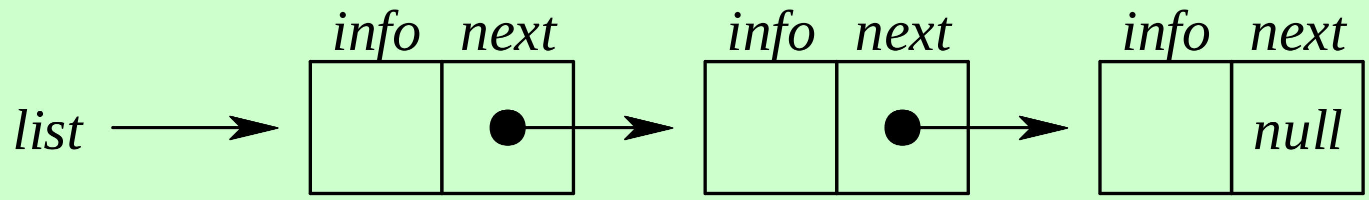
Example: Working with linked list

```
/* structure containing a data part and link part */
struct node
{
        int data ;
        struct node * link ;
} node;
```

- **To start with, we have to create a node (the first node), and make head point to it.**

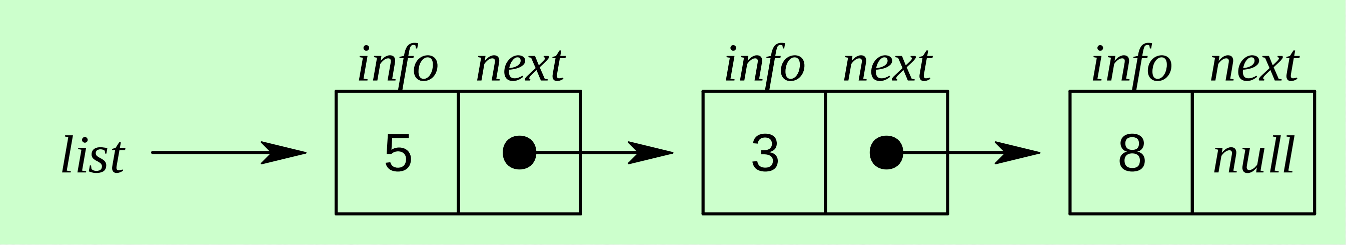
```
head = (node *) malloc(sizeof(node));
```

Linked Lists



Linear linked list

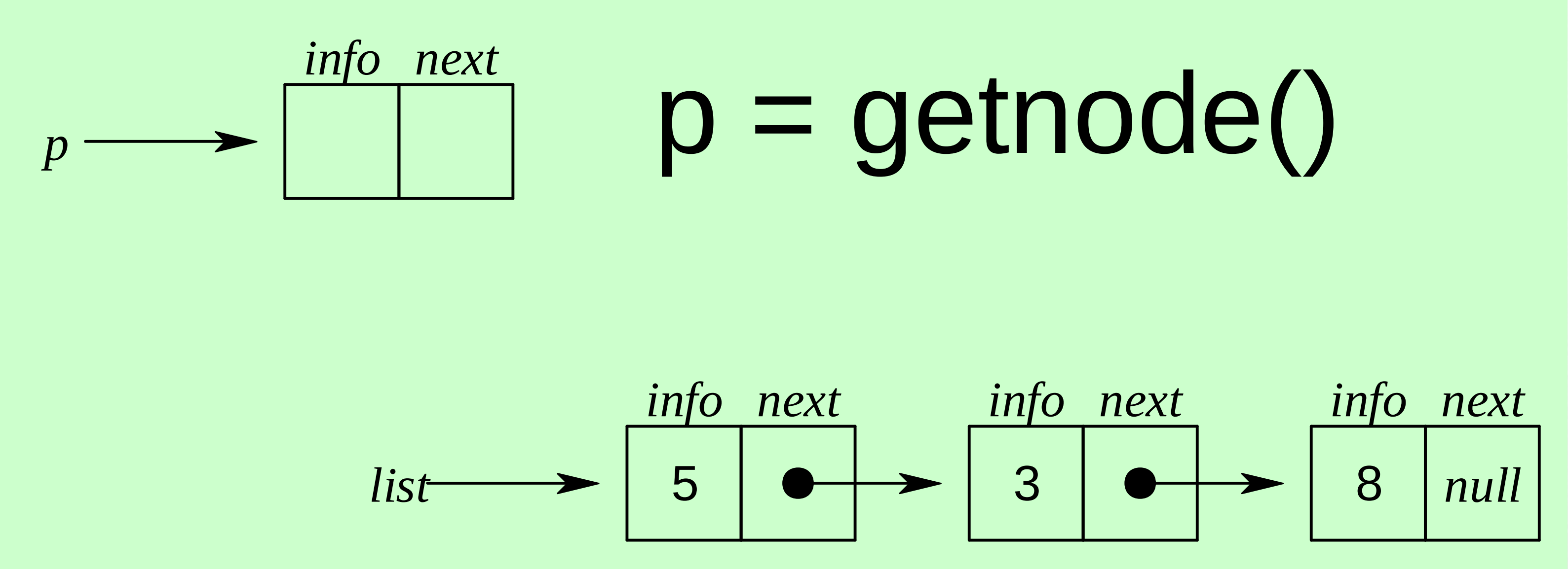
Adding an Element to the front of a Linked List



Some Notations for use in algorithm

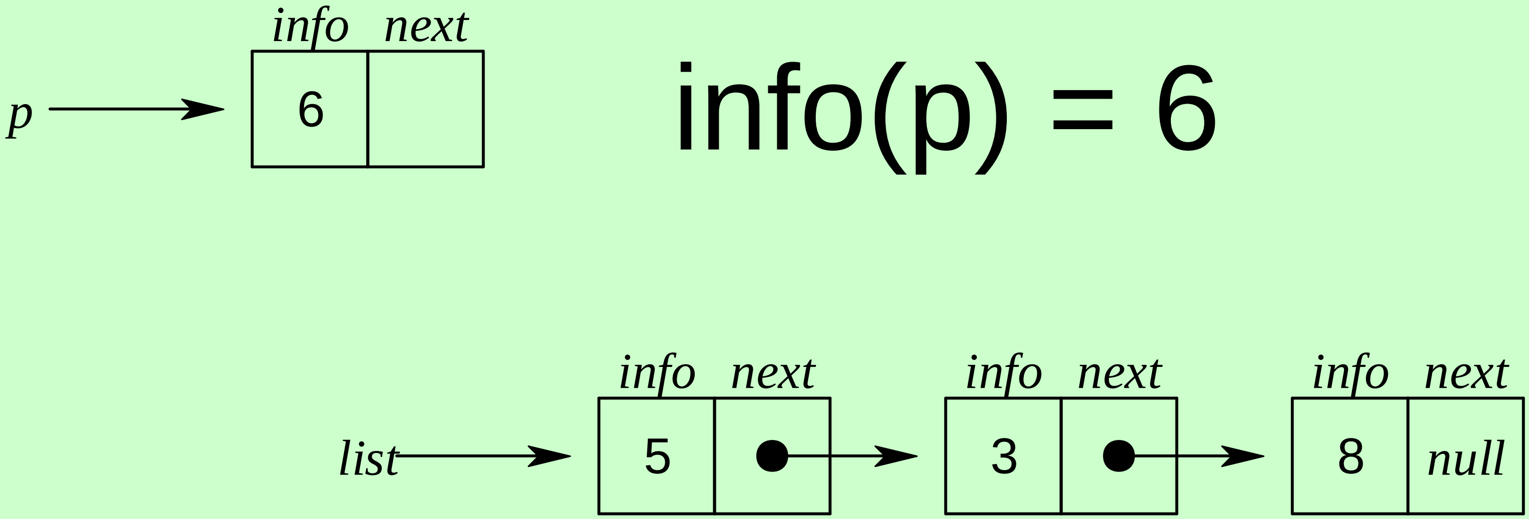
- *p*: is a pointer
- *node(p)*: the node pointed to by *p*
- *info(p)*: the information portion of the node
- *next(p)*: the next address portion of the node
- *getnode()*: obtains an empty node
- *freenode(p)*: makes *node(p)* available for reuse

Adding an Element to the front of a Linked List

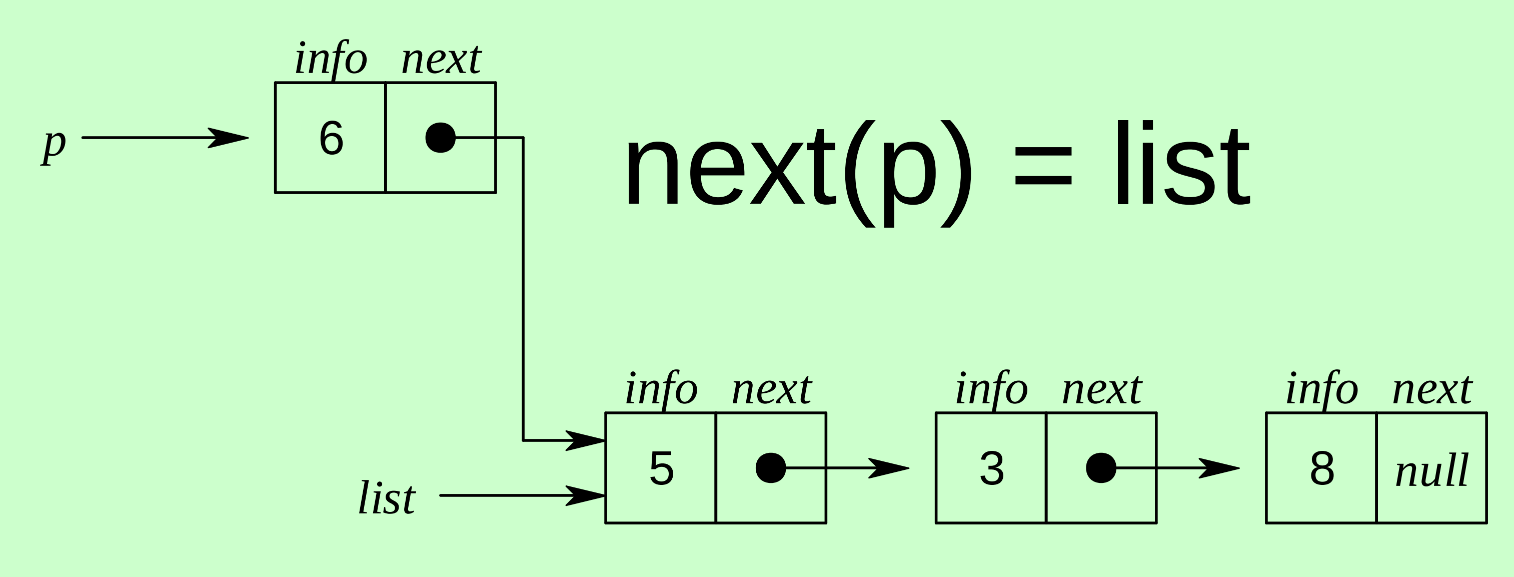


```
p = getnode()
```

Adding an Element to the front of a Linked List

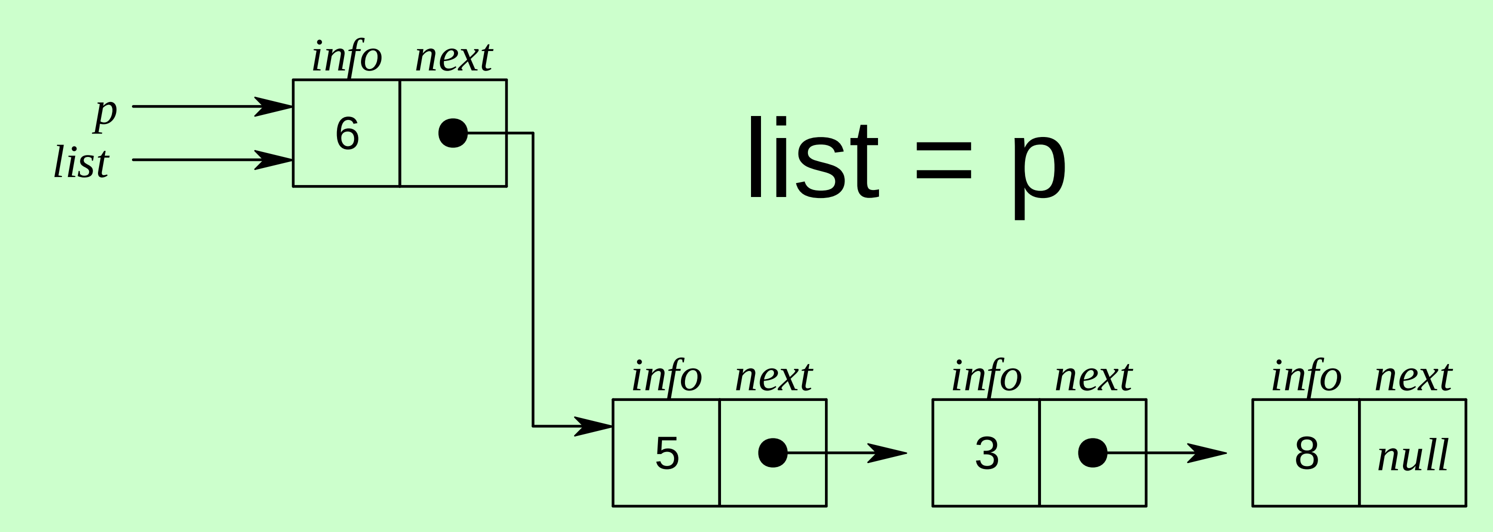


Adding an Element to the front of a Linked List



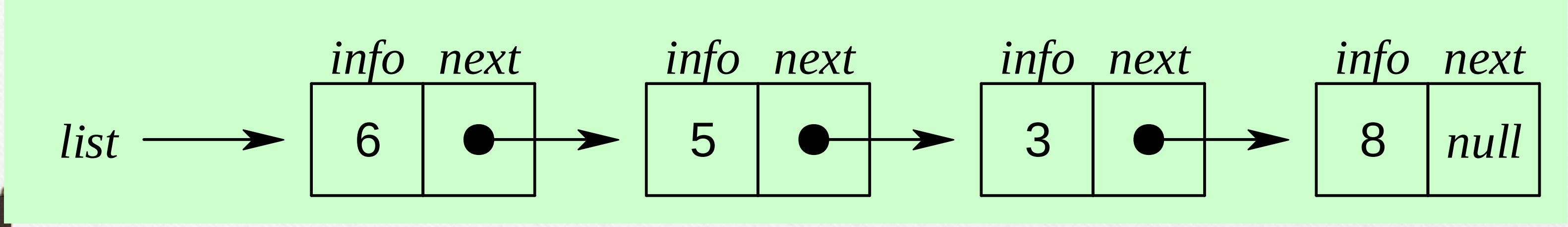
next(p) = list

Adding an Element to the front of a Linked List

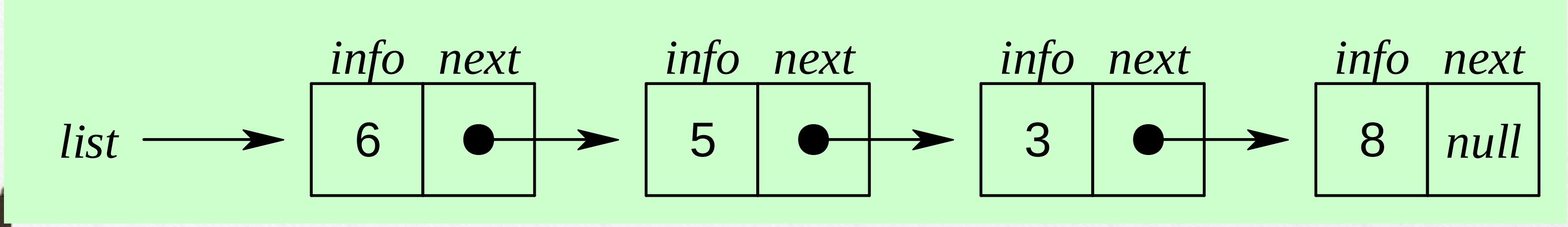


list = p

Adding an Element to the front of a Linked List

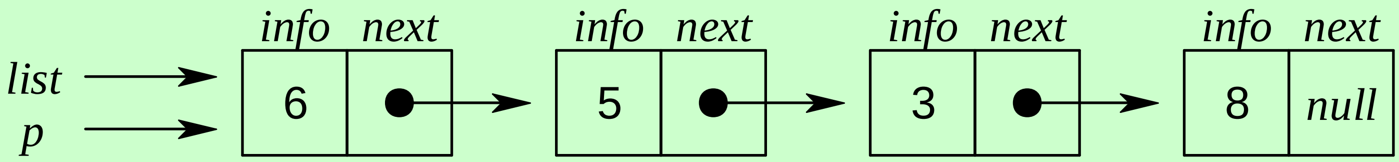


Removing an Element from the front of a Linked List

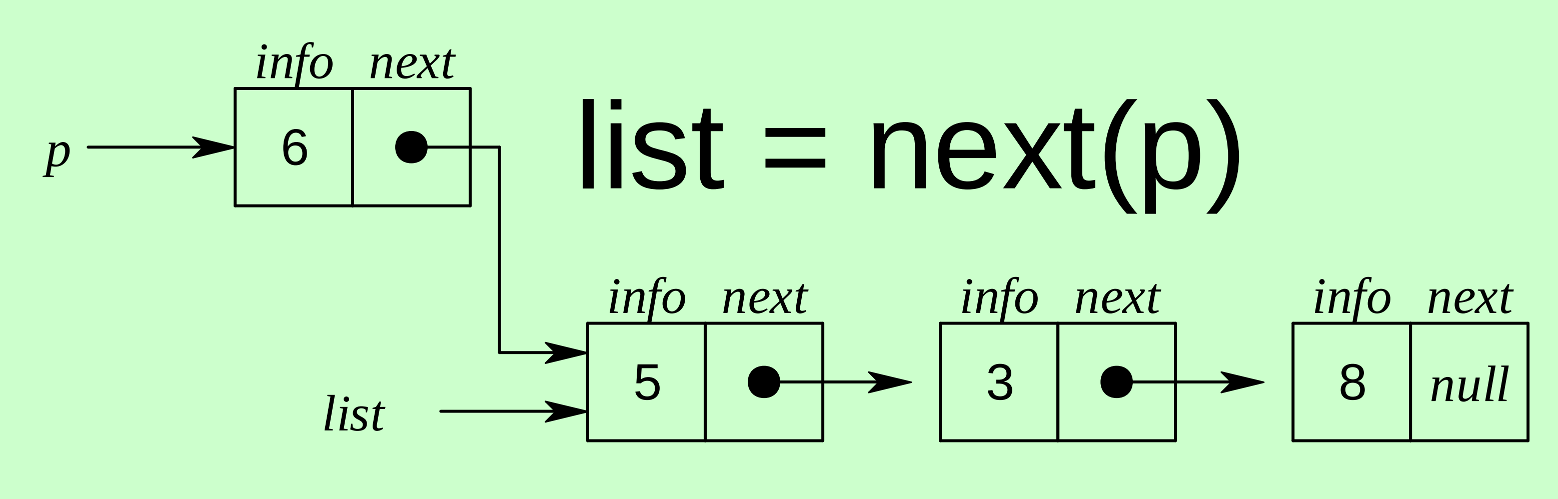


Removing an Element from the front of a Linked List

p = list

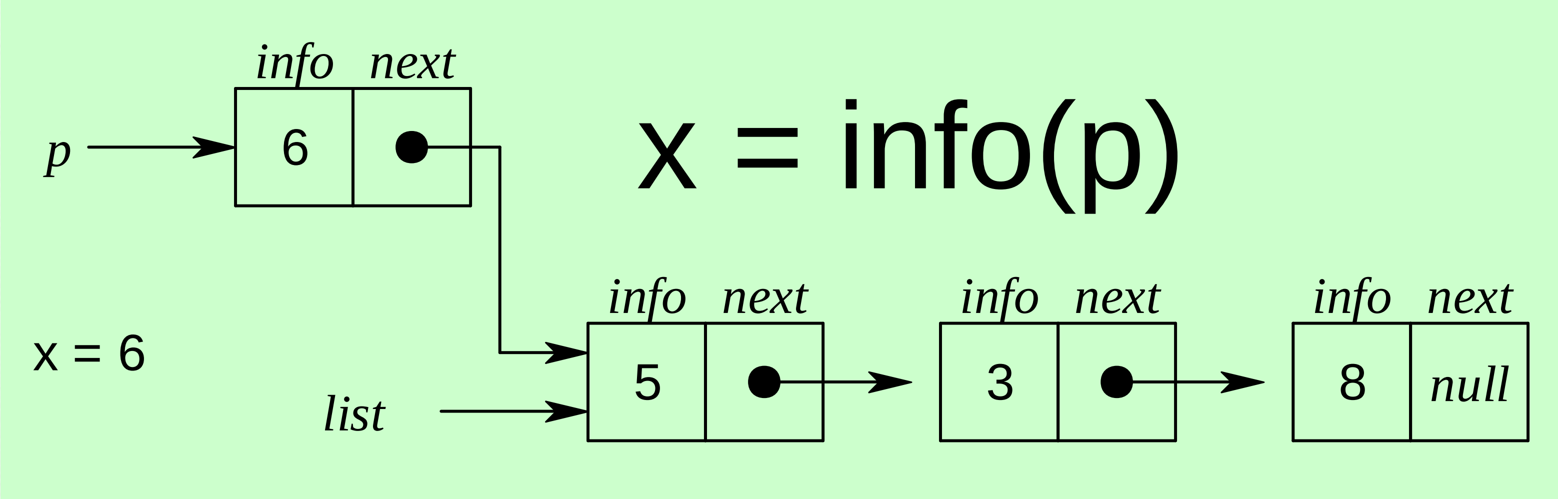


Removing an Element from the front of a Linked List



list = next(p)

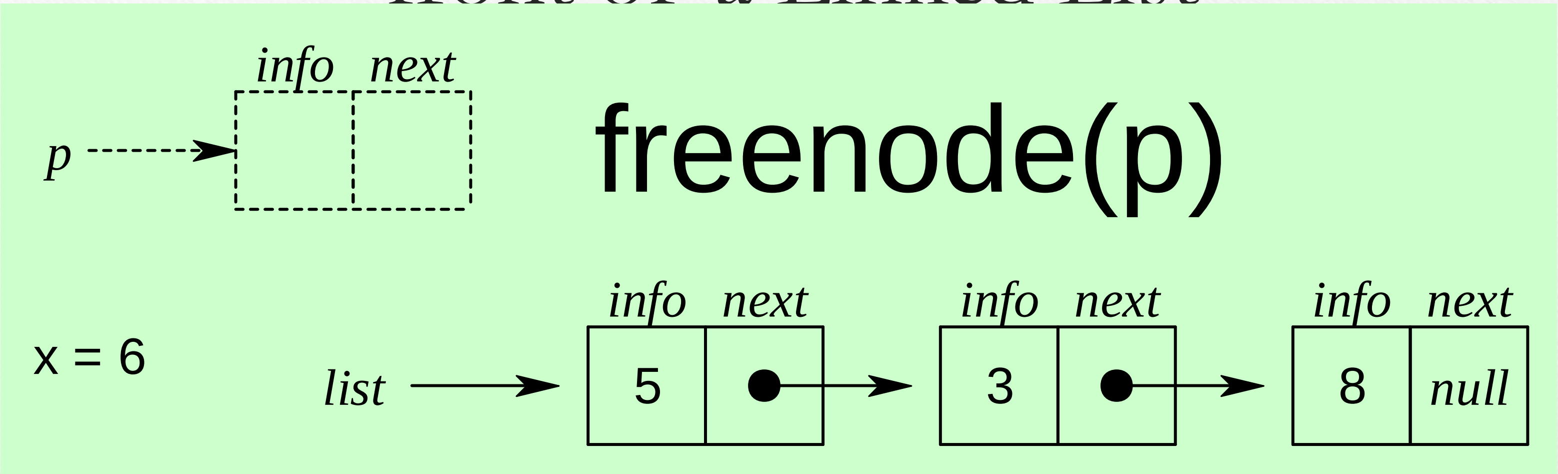
Removing an Element from the front of a Linked List



x = info(p)

x = 6

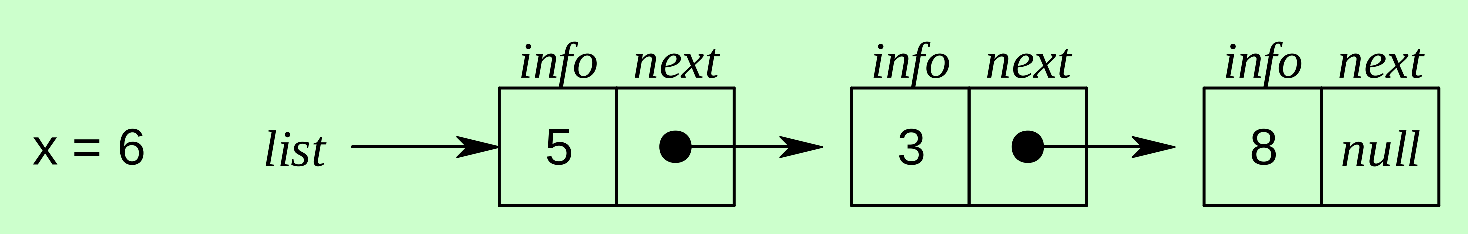
Removing an Element from the front of a Linked List



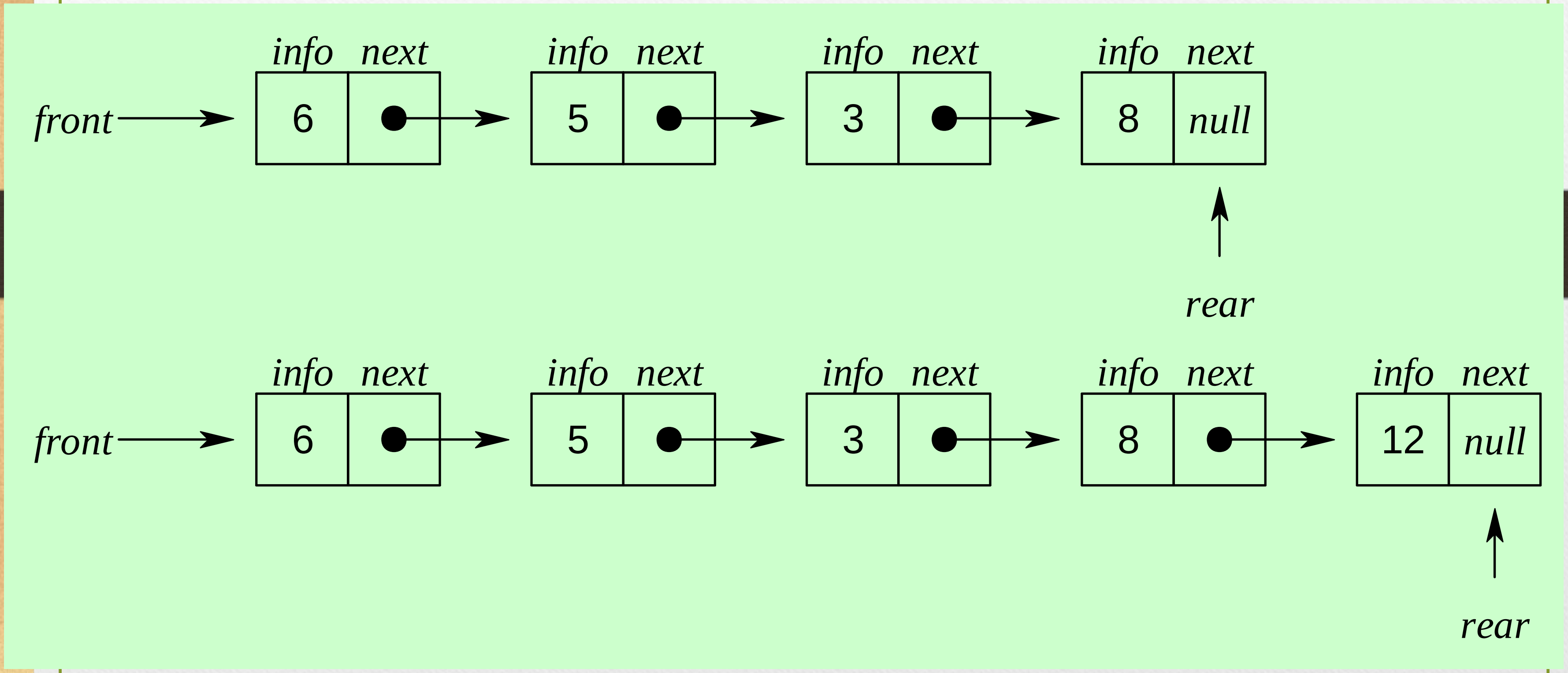
freenode(p)

x = 6

Removing an Element from the front of a Linked List



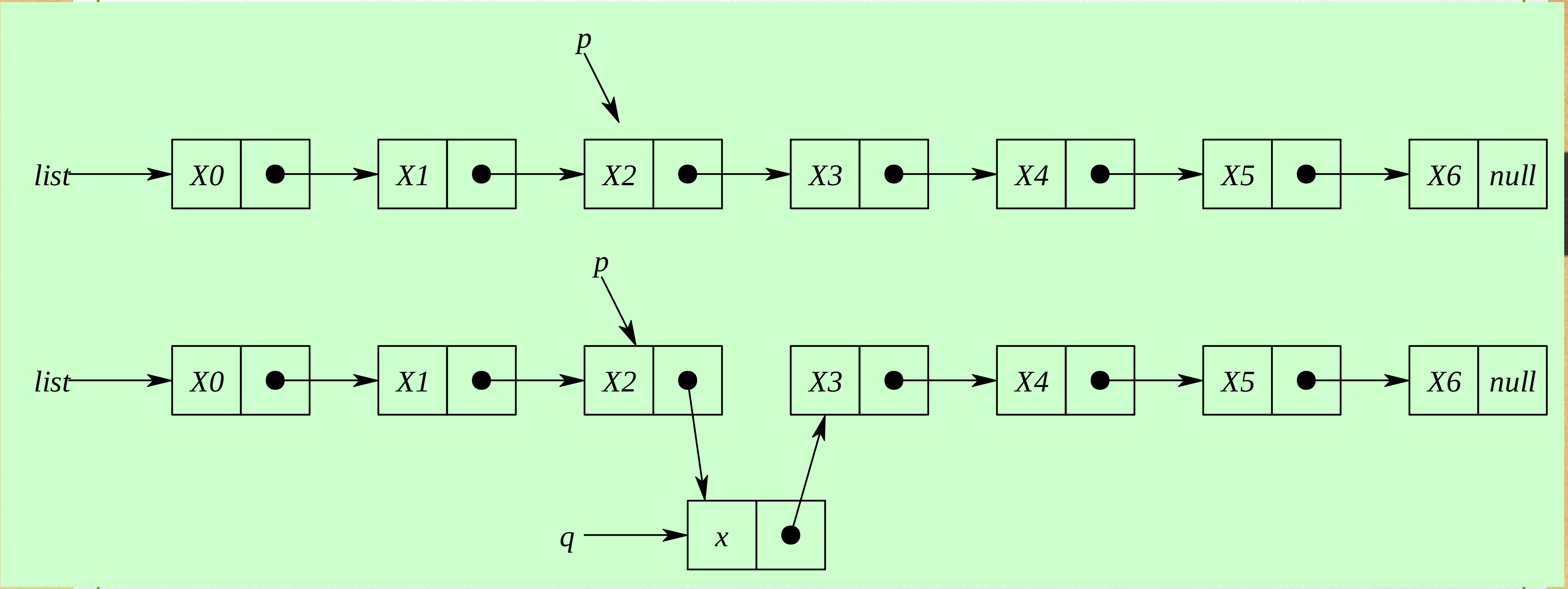
Linked List Implemantation of QUEUES



Linked List as a Data Structure

- An item is accessed in a linked list by traversing the list from its beginning.
- An array implementation allows acccess to the *n*th item in a group using single operation, whereas a list implementation requires *n* operations.
- The advantage of a list over an array occurs when it is necessary to insert or delete an element in the middle of a group of other elements.

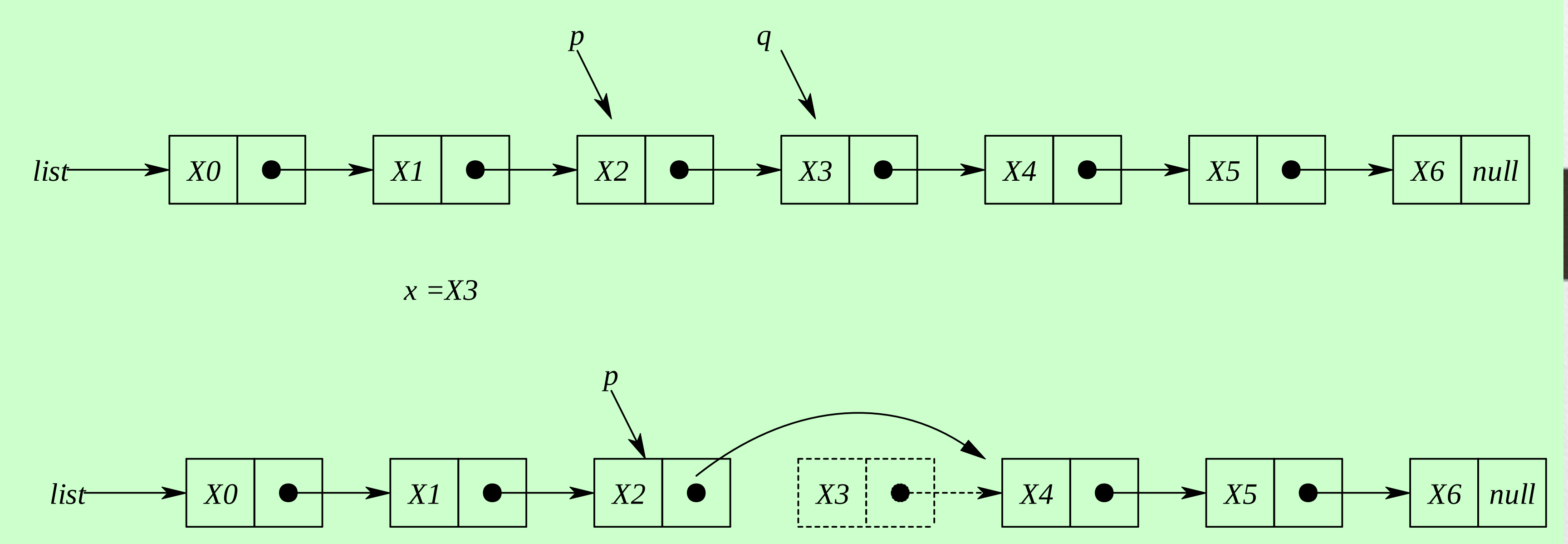
Inserting an item x into a list after a node pointed to by p



Inserting an item *x* into a list after a node pointed to by *p*

```
q=getnode();
info(q)=x;
next(q)=next(p);
next(p)=q;
```

Deleting an item *x* from a list after a node pointed to by *p*



Deleting an item *x* from a list after a node pointed to by *p*

```
q=next(p);
x=info(q);
next(p)=next(q);
freenode(q);
```

LINKED LISTS STRUCTURES AND BASIC FUNCTIONS

```
struct node{
    int info;
    struct node *next;
};
typedef  struct node *NODEPTR;
```

LINKED LISTS STRUCTURES AND BASIC FUNCTIONS

- When a new node is no longer used (e.g. to be deleted from the list) the following function, ***freenode***, can be used to release the node back to the memory.

```
void freenode(NODEPTR p)
{
  free(p);
}
```

Assignments on Linked List

❑Write a linked list program with following operations

- Append node
- Add node at beginning
- Add node at particular location
- Delete specific(data) node
- Delete the first node
- Delete last node
- find the number of nodes in the linked list
- Display linked list elements

Assignments on Linked List

- Write a linked list program with following operations
  - Insert node at beginning
  - Delete node at beginning
  - find the number of nodes in the linked list

- Write a linked list program with following operations
  - Append node
  - Delete the first node
  - find the number of nodes in the linked list