# Listas enlazadas

# Introducción

- En este capítulo se comienza el estudio de las estructuras de datos dinámicas.
- Las listas dinámicas son un conjunto de elementos que crecen y decrecen dinámicamente en función de si se necesitan más elementos o si no se necesitan.

# Introducción

- Es una estructura de datos donde se cumple:
  - Todos sus componentes son del mismo tipo.
  - Cada elemento o nodo va seguido de otro del mismo tipo o de ninguno.
  - Sus componentes se almacenan según cierto orden.

# Lista enlazada

- Es un conjunto de elementos enlazados a través de un campo en el que se guarda o se hace referencia a la posición del siguiente elemento.
- Cada elemento contiene información necesaria para llegar al siguiente elemento.
- La posición del siguiente elemento de la estructura, la determina el elemento actual.
- Es necesario almacenar la posición del primer elemento, además es dinámica, es decir, su tamaño cambia durante la ejecución del programa.

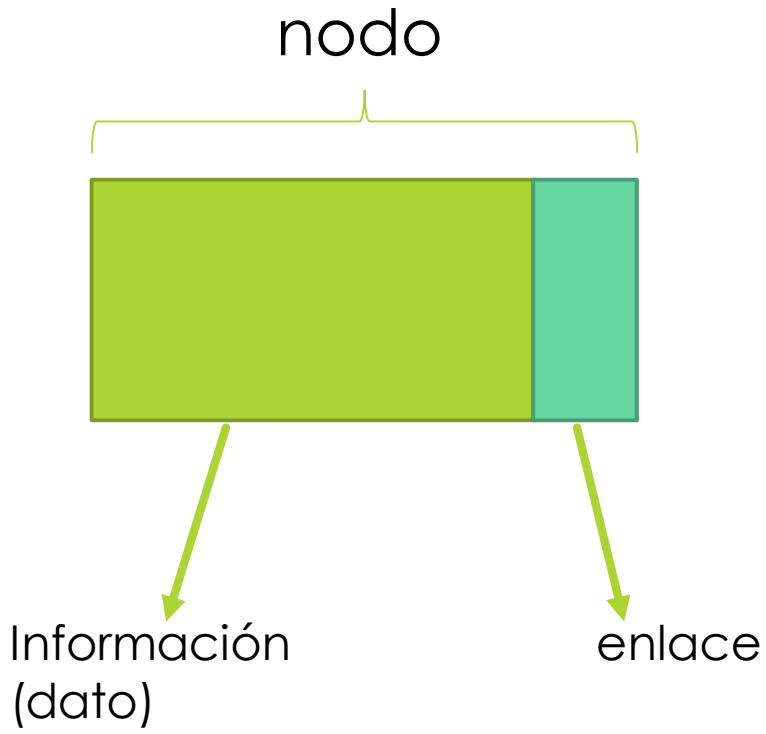
# Tipos de listas enlazadas

- Listas simplemente enlazadas
- Listas doblemente enlazadas
- Listas simplemente enlazadas circulares
- Listas doblemente enlazadas circulares
- Listas con nodo cabeza.

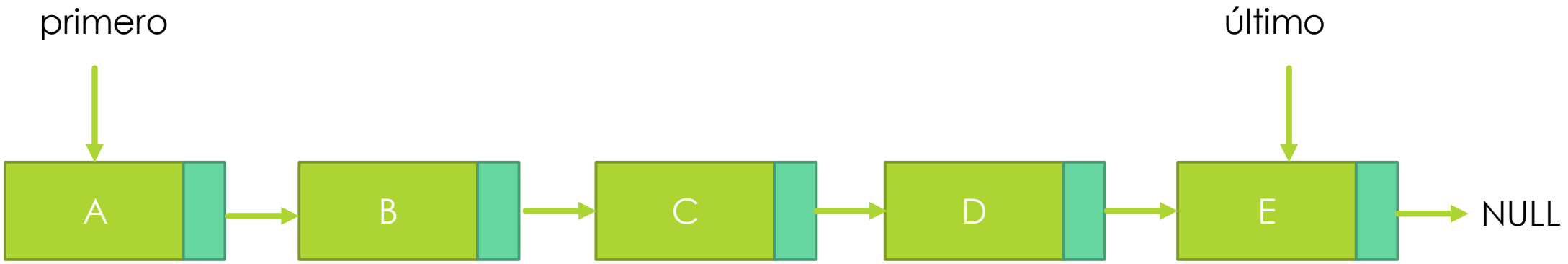
# Listas simplemente enlazadas

- En este tipo de listas cada elemento conoce que elemento es el que le sucede en el orden, pero no cual le precede.
- Se caracterizan porque solo tienen un enlace.
- Las listas simplemente enlazadas son tipos de datos dinámicos que se construyen con nodos.
- Un nodo es un registro que tiene diferentes campos para el registro de datos y el enlace.
- Ejemplos: Lista de estudiantes, lista de materias, etc.

# Representación de un nodo



# Representación de listas simplemente enlazadas



# Uso de las listas simplemente enlazadas

- En la resolución de problemas que requieran el manejo de un conjunto de elementos y que estos elementos se encuentren uno a continuación de otro.
- Ejemplos:
- Lista de estudiantes de una materia.
- Atención de clientes en un banco.
- Atención de pacientes en un hospital.

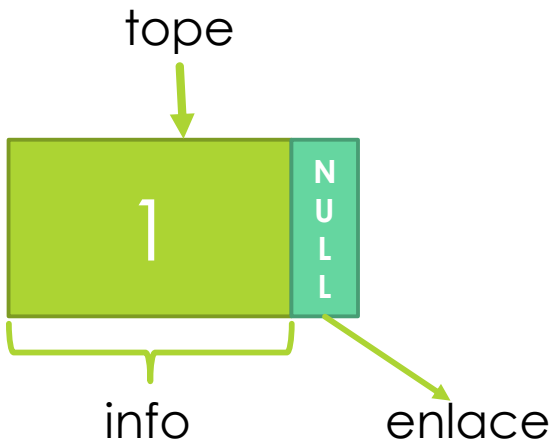
# Implementación de pilas con LSE - Insertar

1. Crear la clase Nodo.

2. Crear la clase ListasSimplementeEnlazadas.

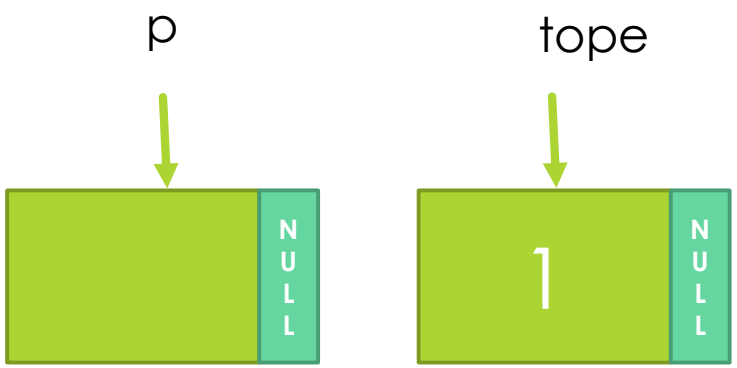
3. Crear los objetos de la clase ListasSimplementeEnlazadas llamados tope y p que inicialmente son igual a null.

4. Si el tope = null -> tope = new Nodo(); Asignamos los valores para tope.

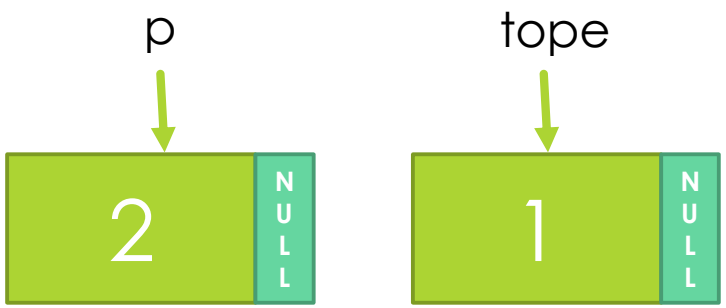


# Implementación de pilas con LSE - Insertar

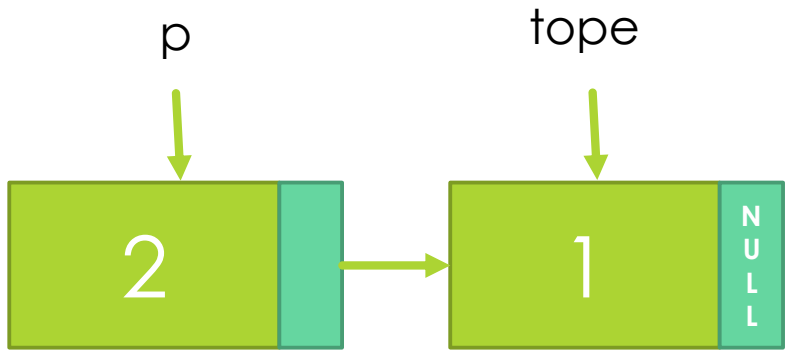
a)



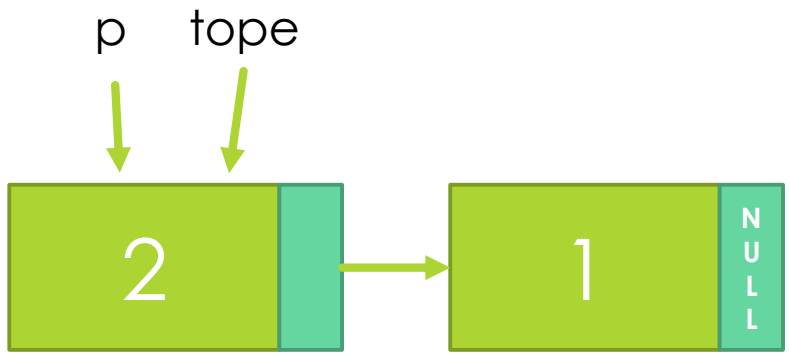

b)



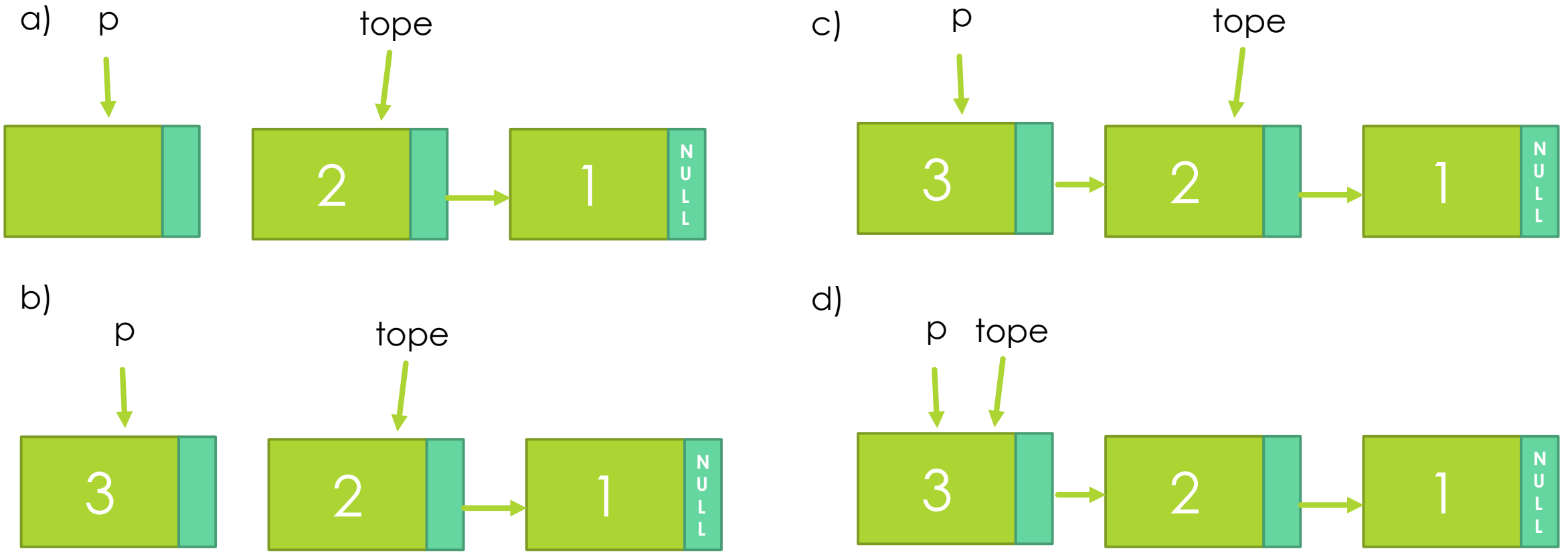

c)



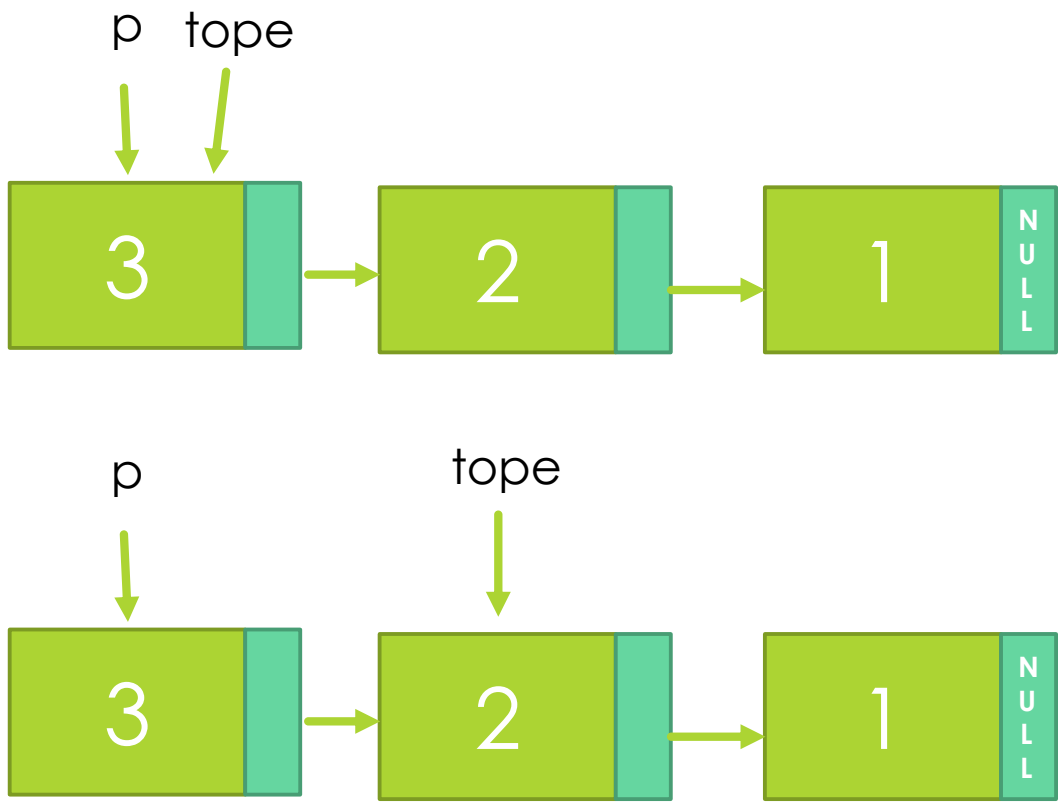

d)

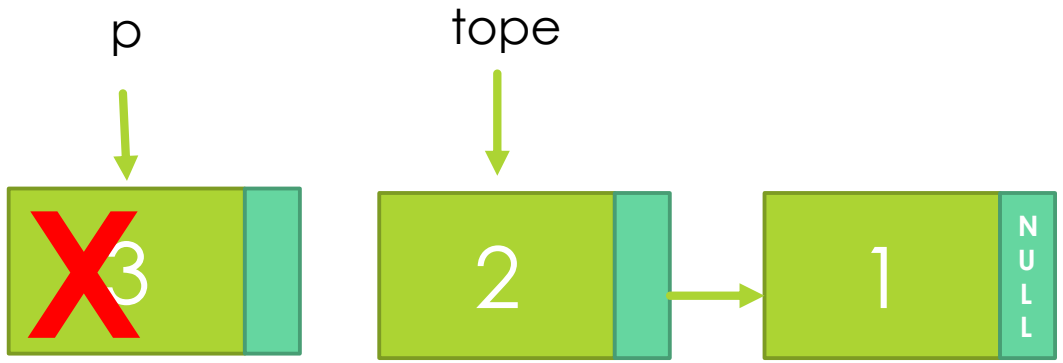


# Implementación de pilas con LSE - Insertar

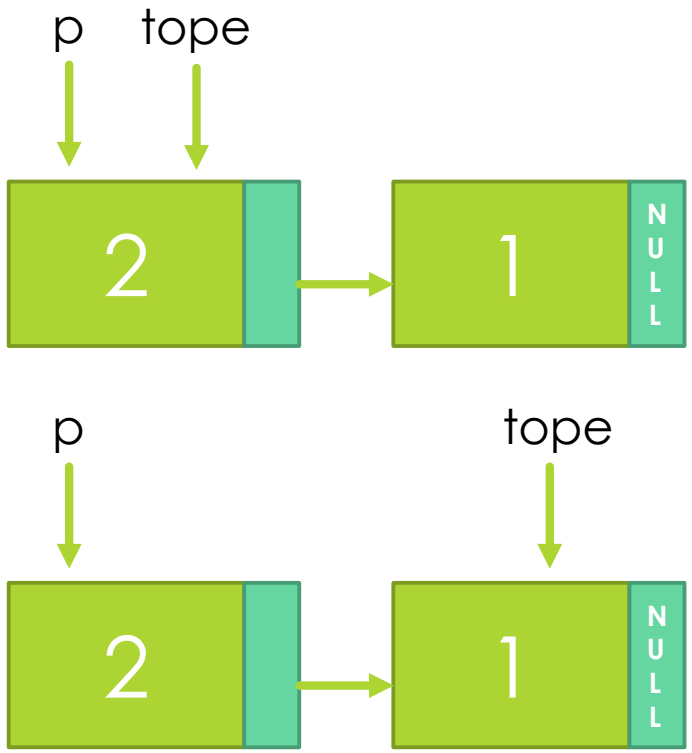


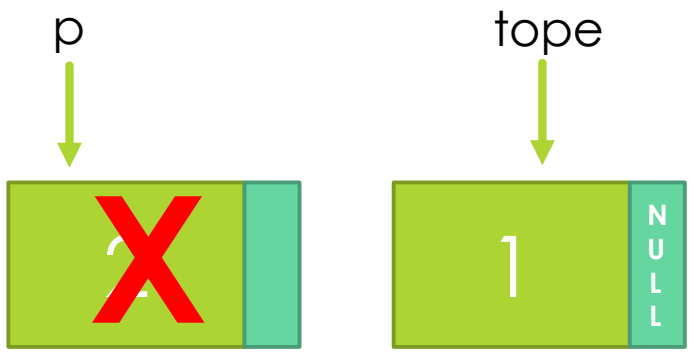
# Implementación de pilas con LSE - Eliminar



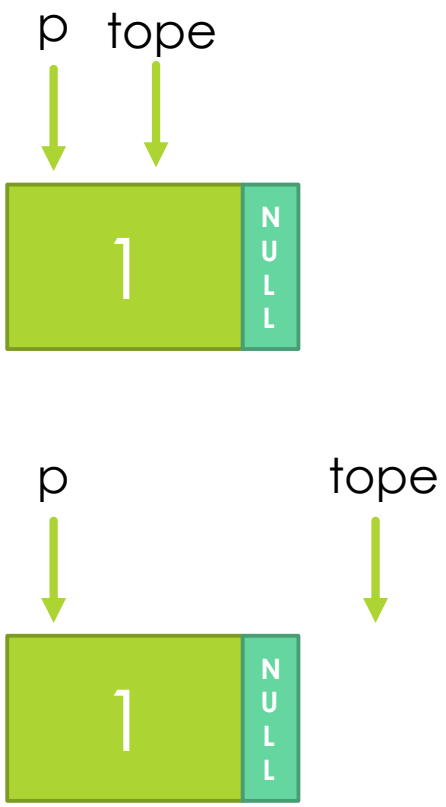



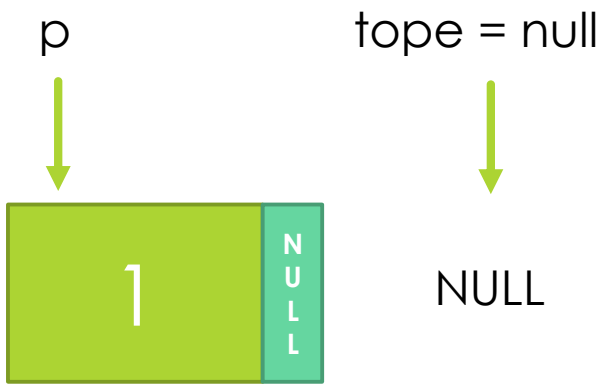
# Implementación de pilas con LSE - Eliminar



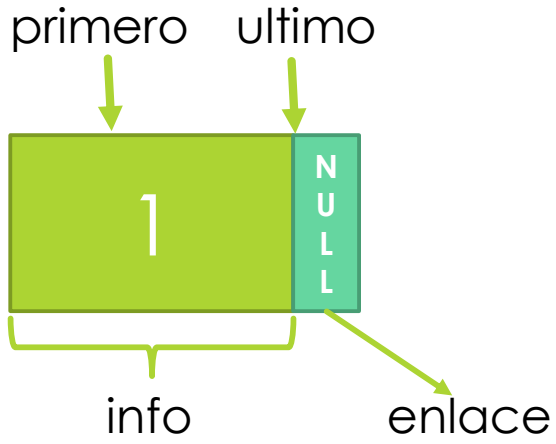
# Implementación de pilas con LSE - Eliminar



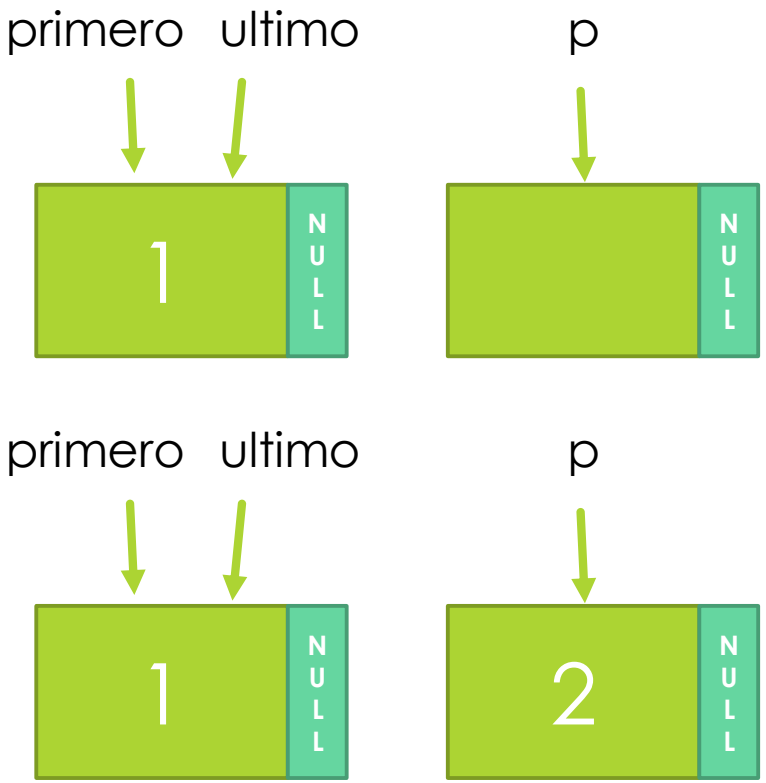



# Implementación de colas con LSE - Insertar

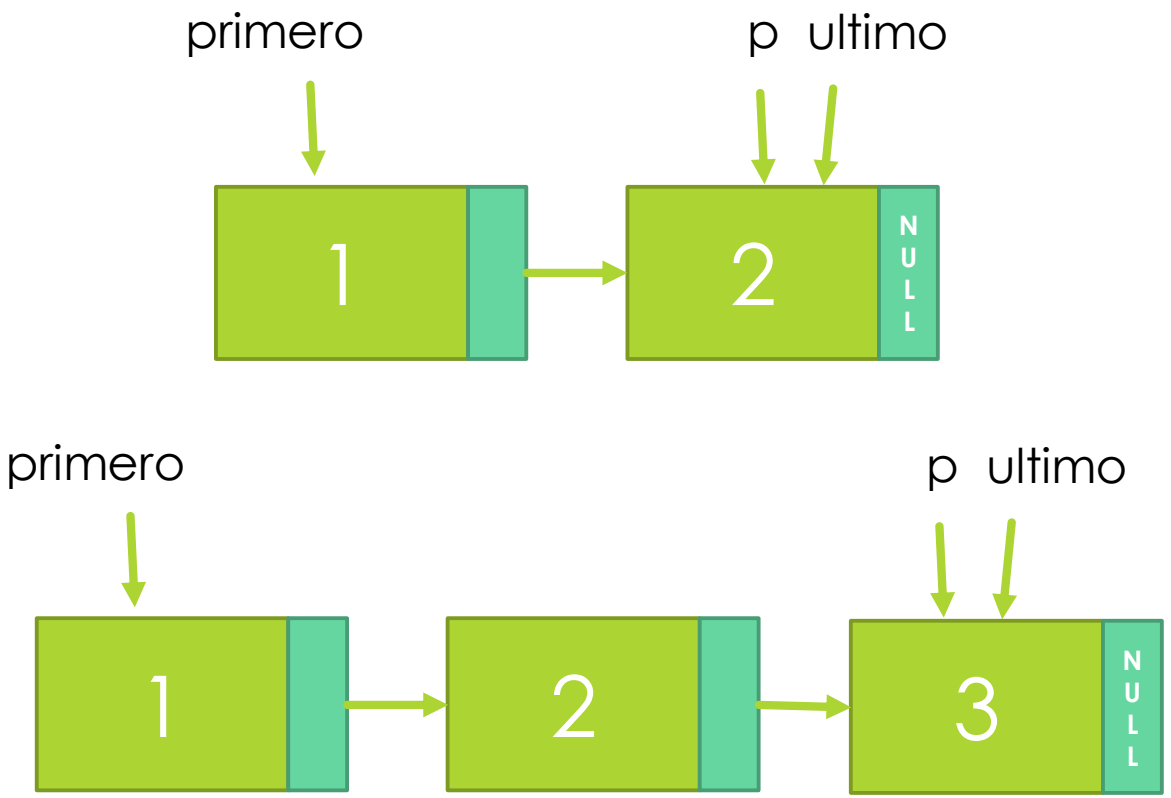
Si ultimo = null ; creamos una nueva instancia del nodo en último y le asignamos valores.



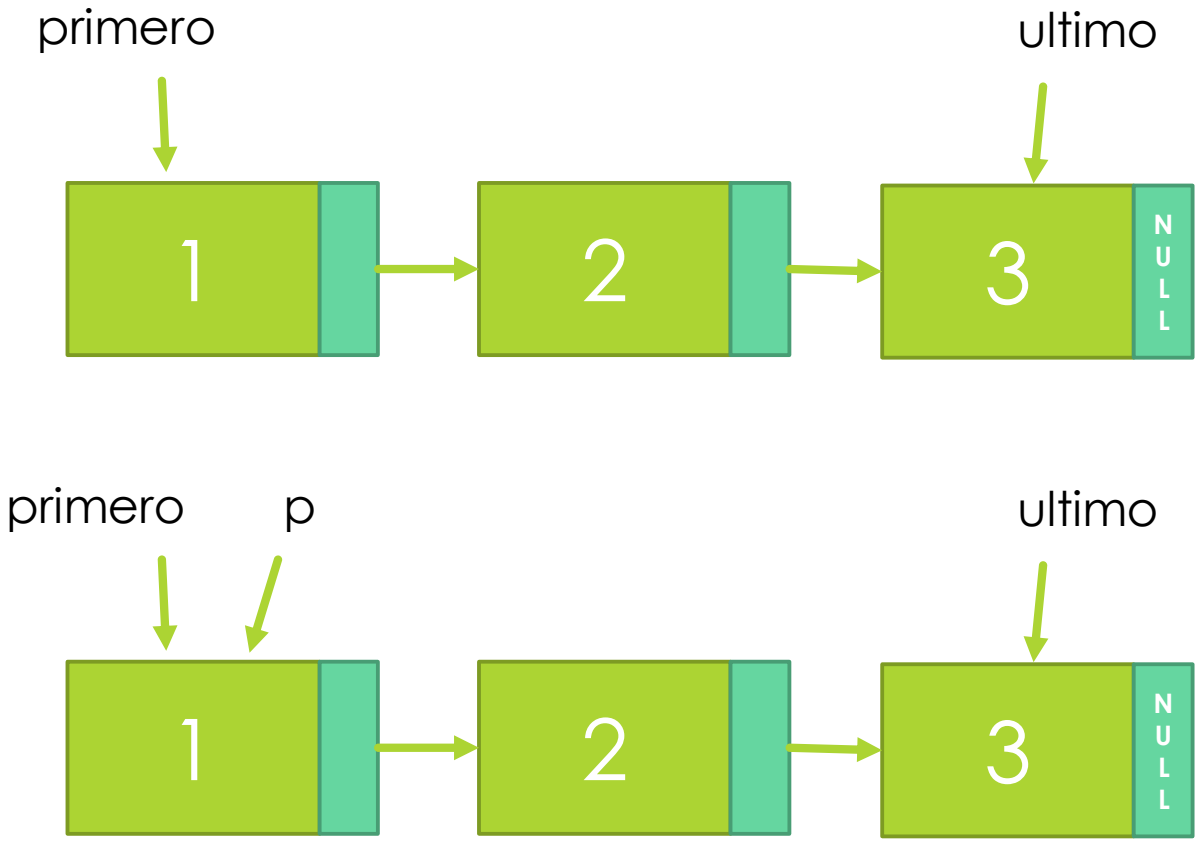
# Implementación de colas con LSE - Insertar



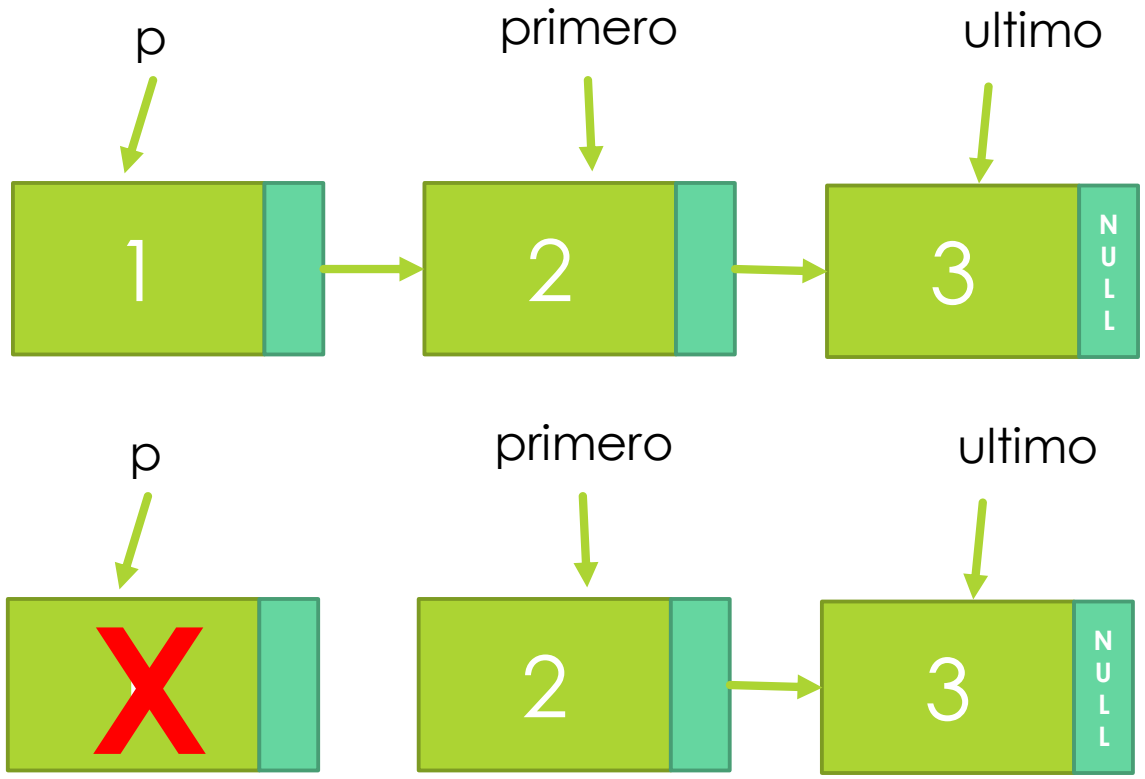
# Implementación de colas con LSE - Insertar



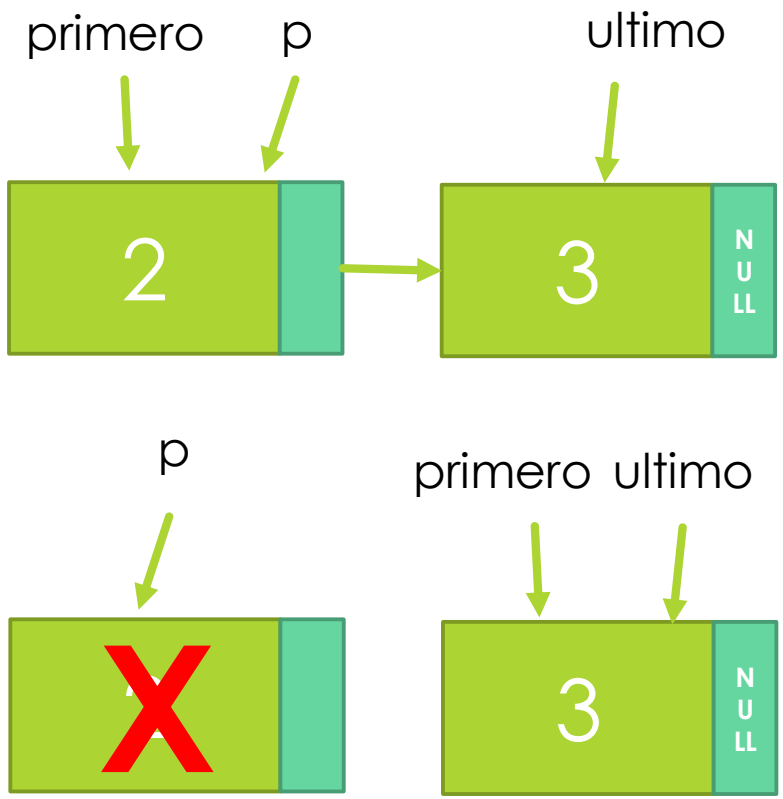
# Implementación de colas con LSE - Eliminar



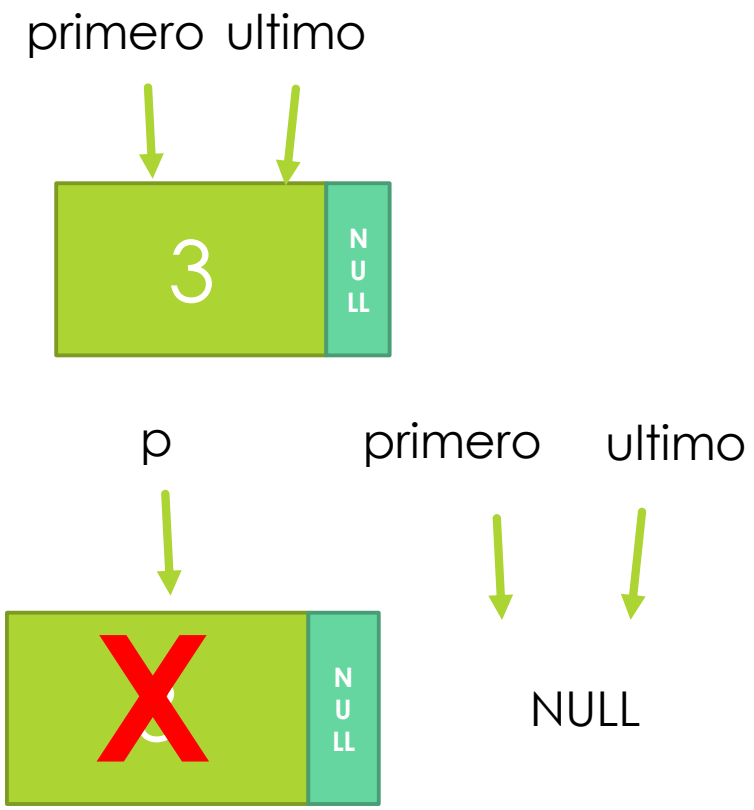
# Implementación de colas con LSE - Eliminar



# Implementación de colas con LSE - Eliminar



# Implementación de colas con LSE - Eliminar



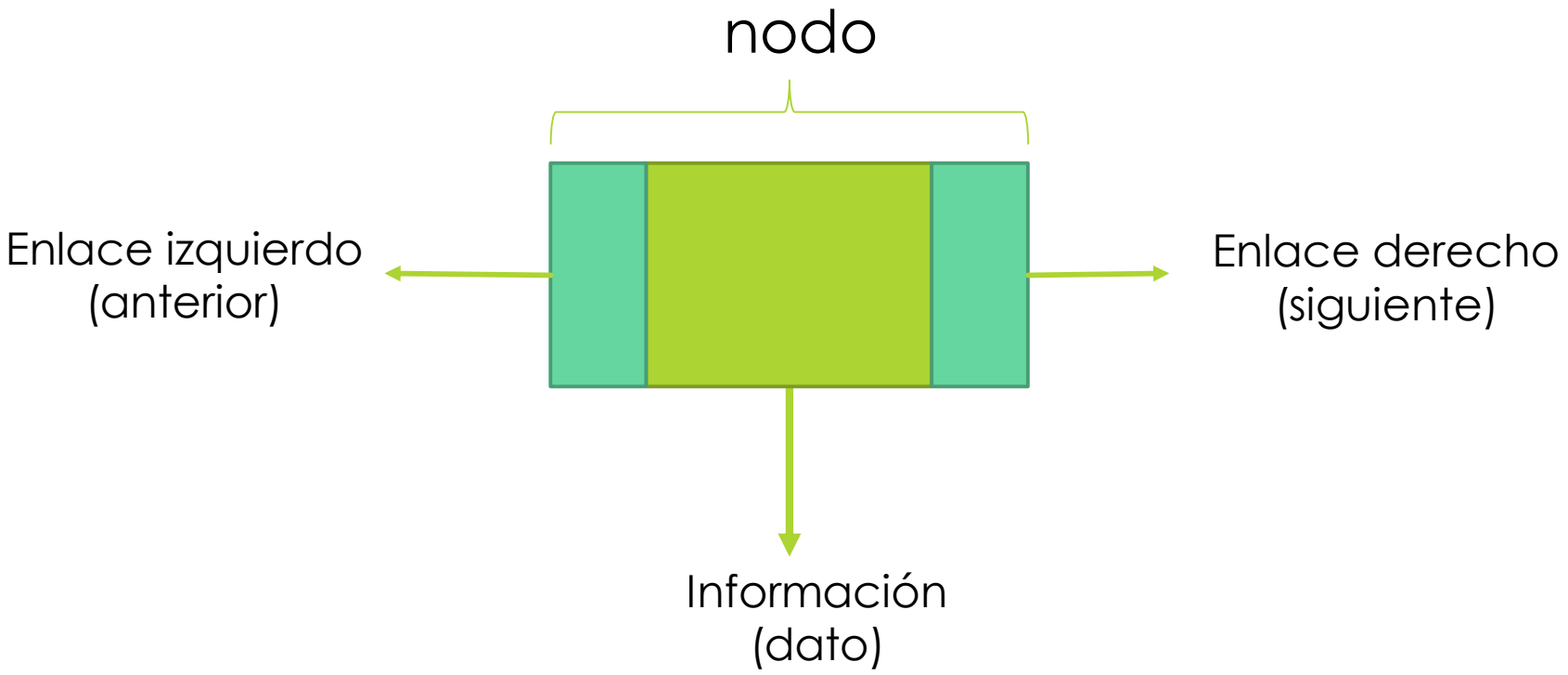
# Trabajo práctico

- Implementar el método insertar elementos en una lista simplemente enlazada de forma ordenada.
- Implementar un método que permita contar la cantidad de números pares e impares que contiene la lista simplemente enlazada.

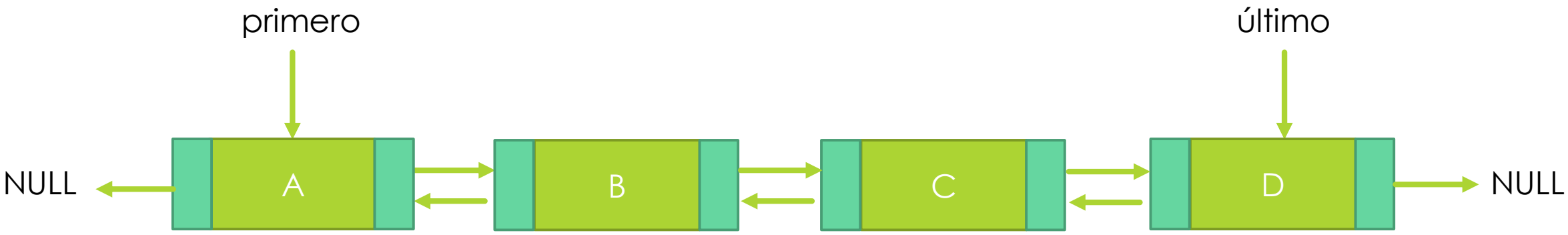
# Listas doblemente enlazadas

- Es una lista que está enlazada en dos sentidos.
- Cada nodo contiene la dirección del nodo anterior y del nodo siguiente.
- Cada elemento conoce que elemento le precede y cual le sucede en el orden.
- En forma general una lista doblemente enlazada tiene los campos: “enlace izquierdo”, “información”, “enlace derecho”.

# Representación de un nodo para LDE



# Representación gráfica de una lista doblemente enlazada



# Operaciones básicas de las LDE

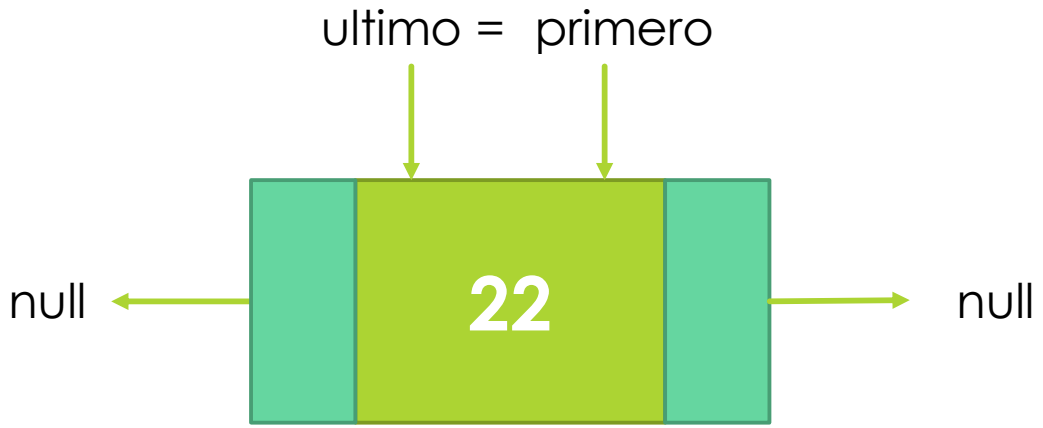
- Vacía: si primero = null
- Insertar
- Eliminar

- Otro ejemplo:
- Insertar de forma ordenada

- Analicemos la forma de resolución del problema e implementemos en c#

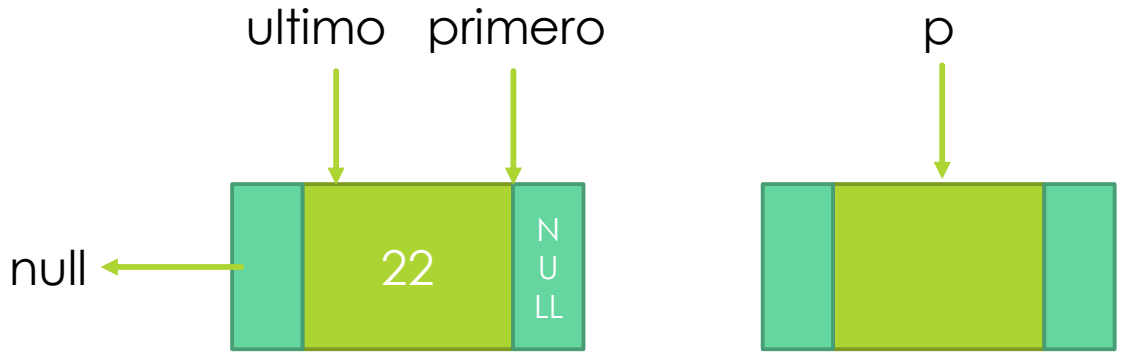
# Operación básica: Insertar

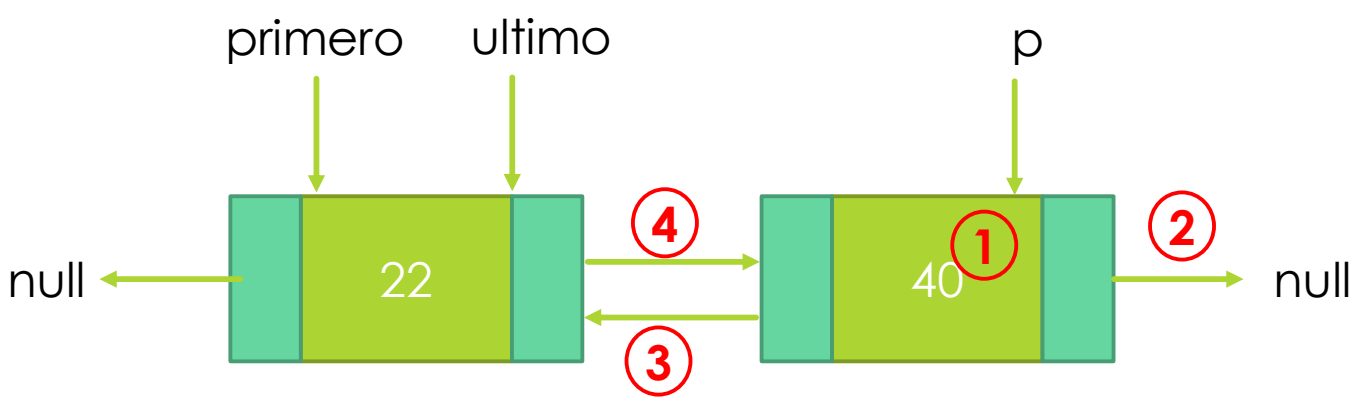
- **Si vacia -> nuevo nodo primero**

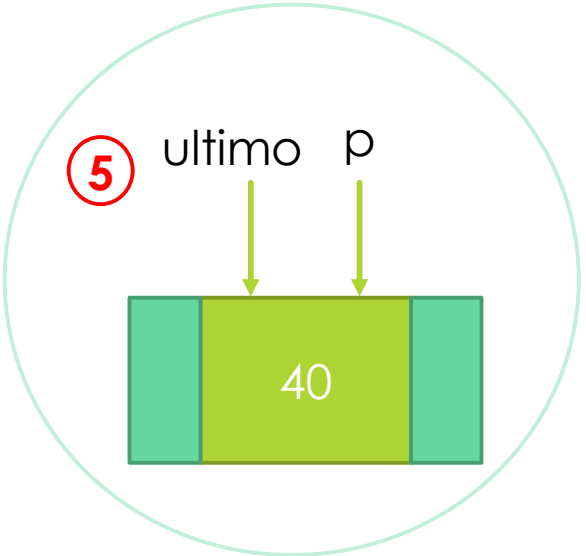


# Operación básica: Insertar

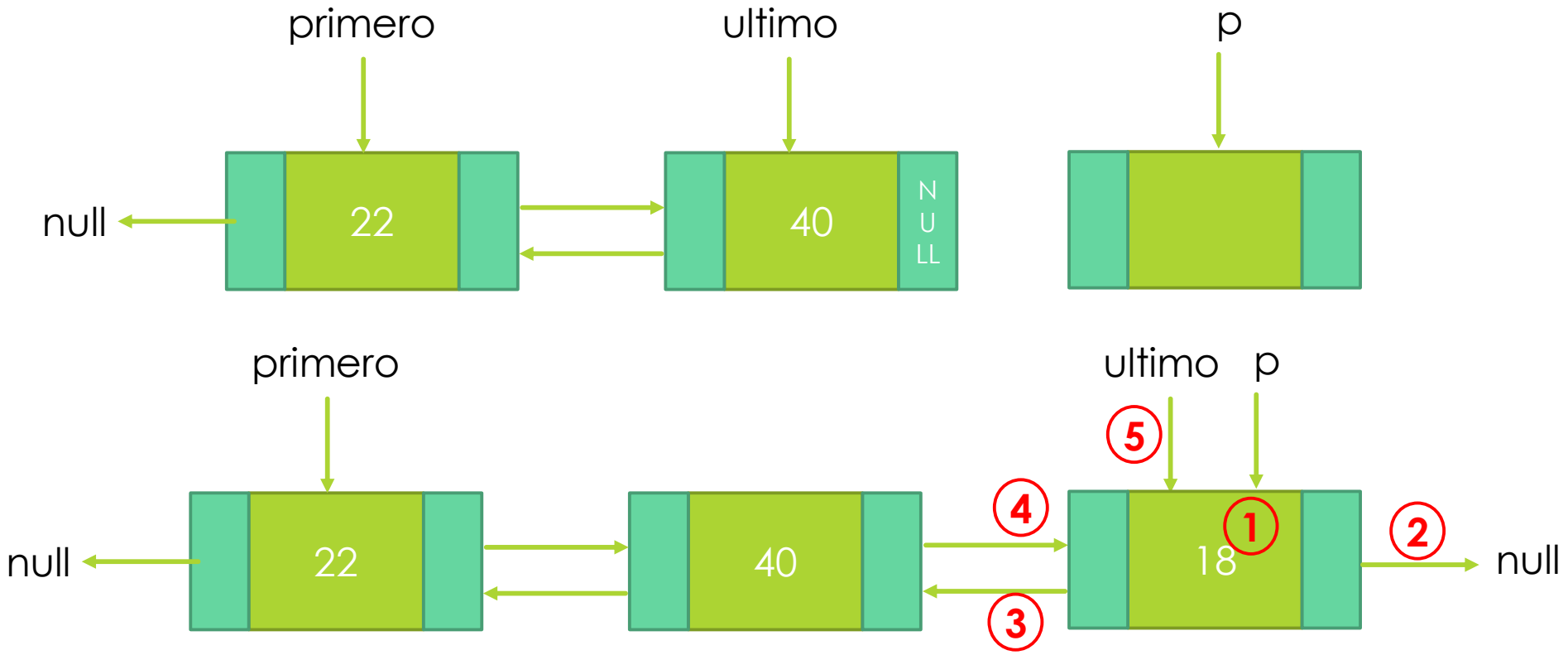
- **si no**



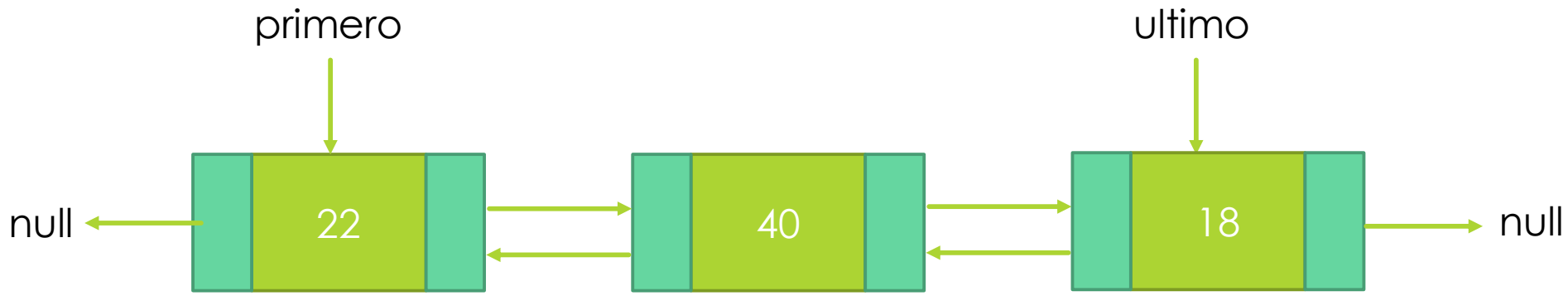



# Operación básica: Insertar



# Operación básica: Eliminar dado elemento x

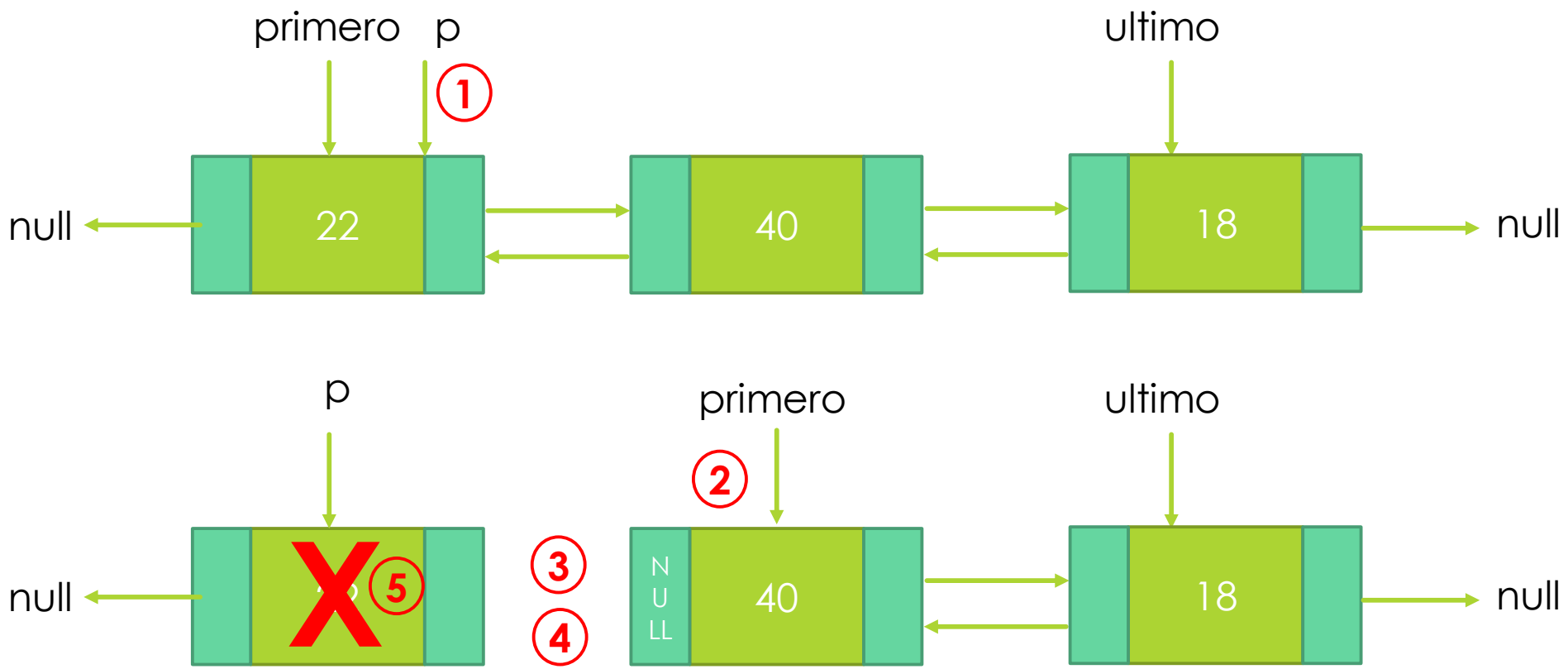
- **Si lista no vacia**




Tenemos 3 opciones:

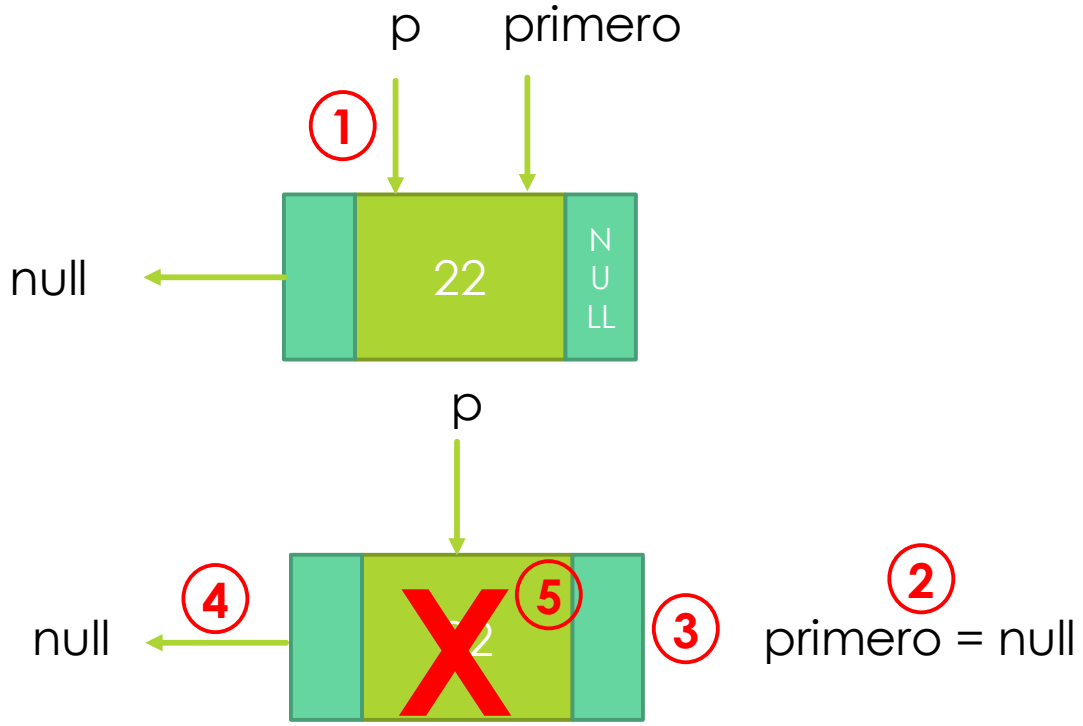
1. El elemento que queremos eliminar se encuentra al principio de la lista.
2. El elemento que queremos eliminar se encuentra en una posición intermedia de la lista.
3. El elemento que queremos eliminar se encuentra al final de la lista.

# 1. El elemento que queremos eliminar se encuentra al principio de la lista.

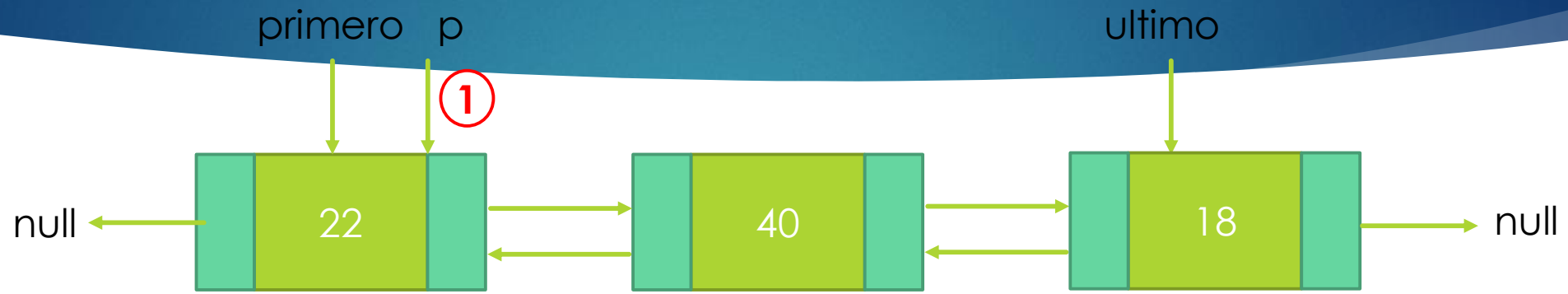


# 1. El elemento que queremos eliminar se encuentra al principio de la lista.

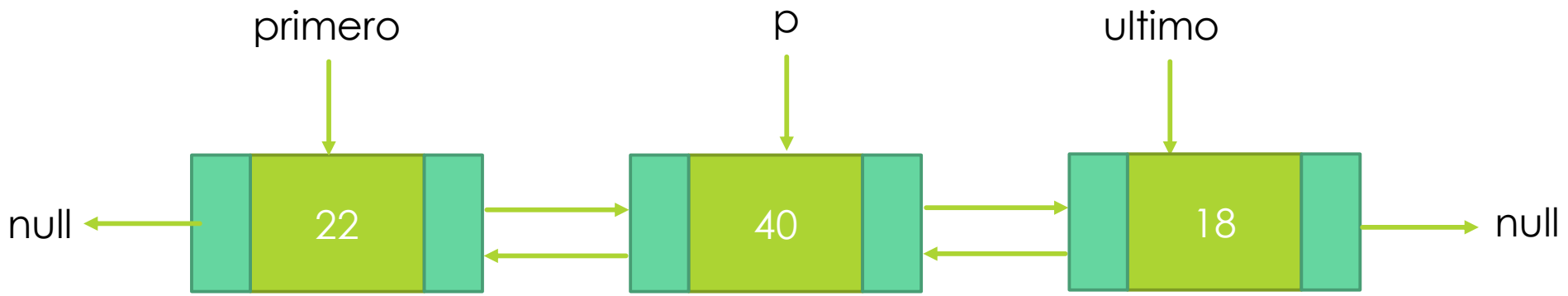
- En el caso de que primero sea el único elemento de la lista



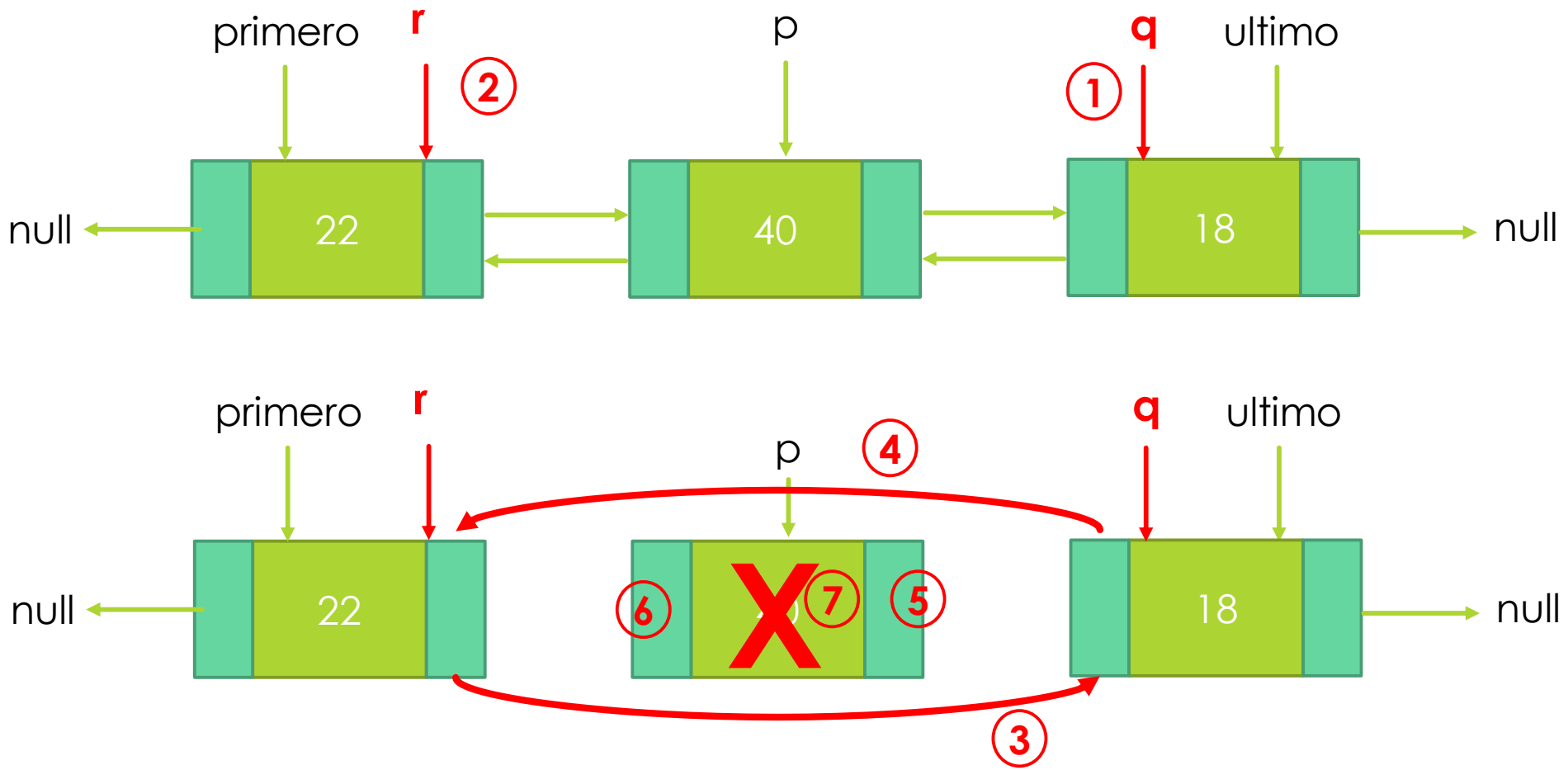
# 2. El elemento que queremos eliminar se encuentra en una posición intermedia de la lista.



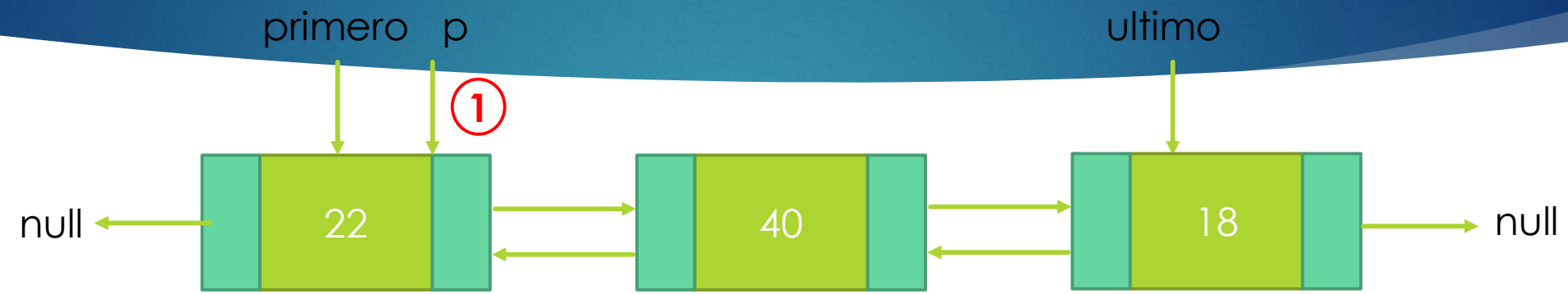

Recorremos la lista a la derecha mientras p <> null && **x** no coincida con un elemento de la lista.



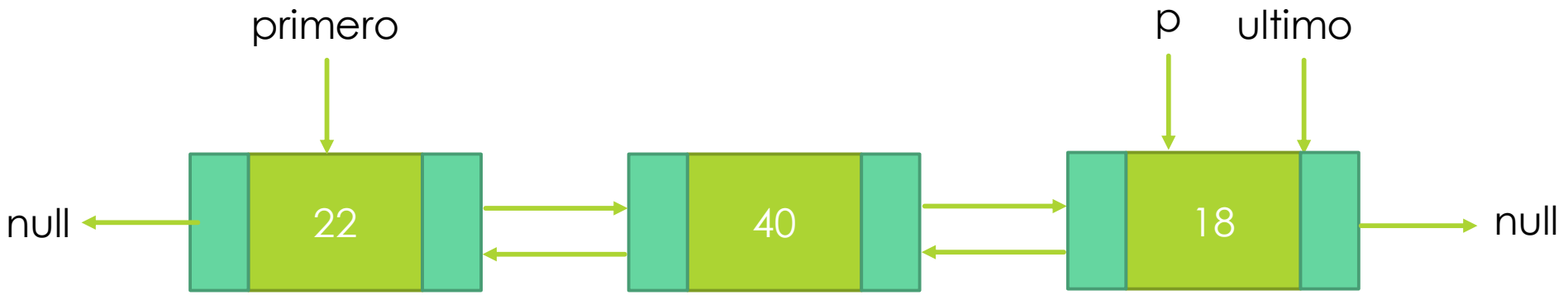
# 2. El elemento que queremos eliminar se encuentra en una posición intermedia de la lista.



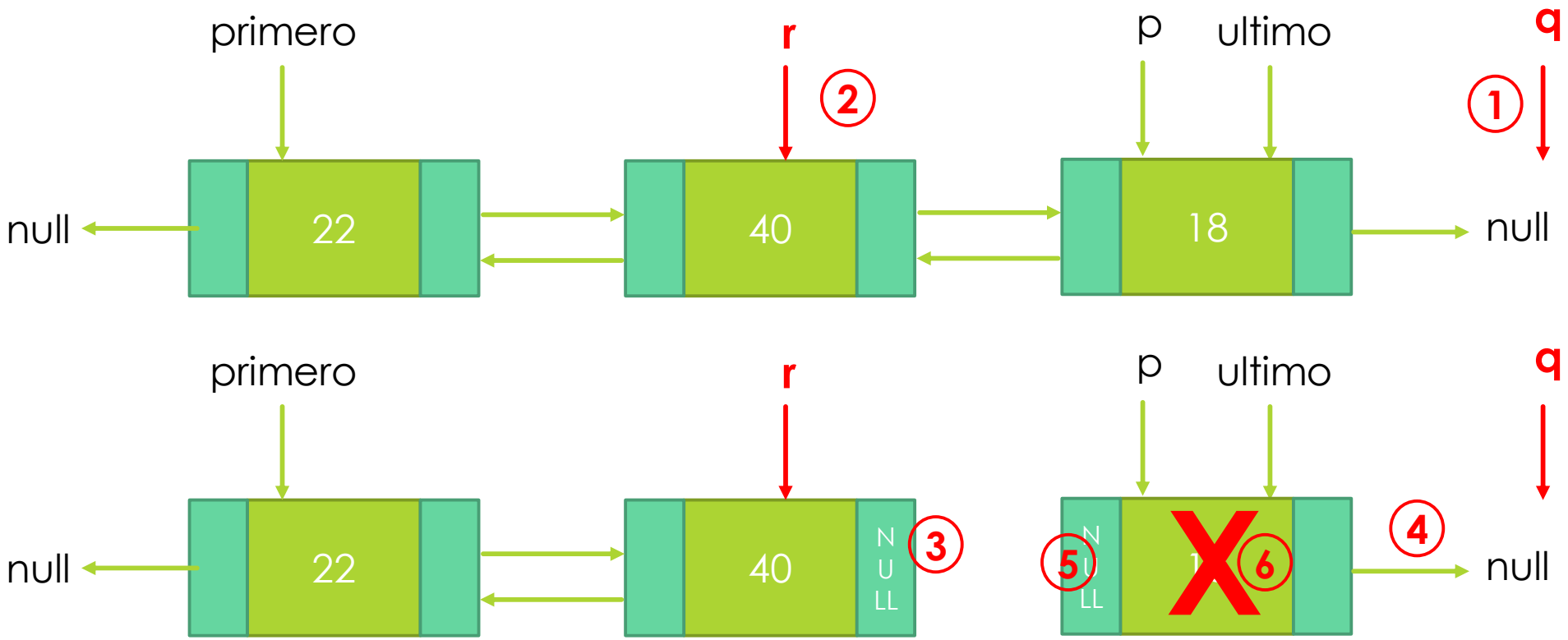
# 3. El elemento que queremos eliminar se encuentra al final de la lista.




Recorremos la lista a la derecha mientras p <> null && **x** no coincida con un elemento de la lista.

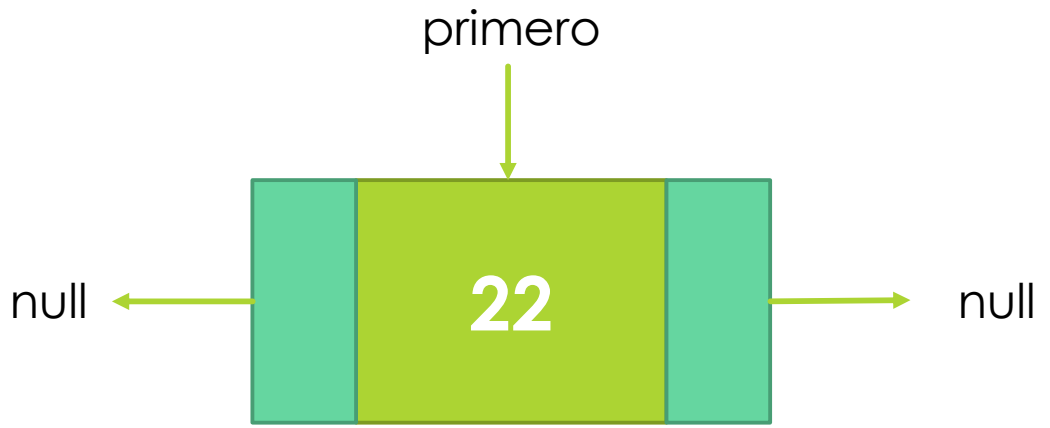


# 3. El elemento que queremos eliminar se encuentra al final de la lista.



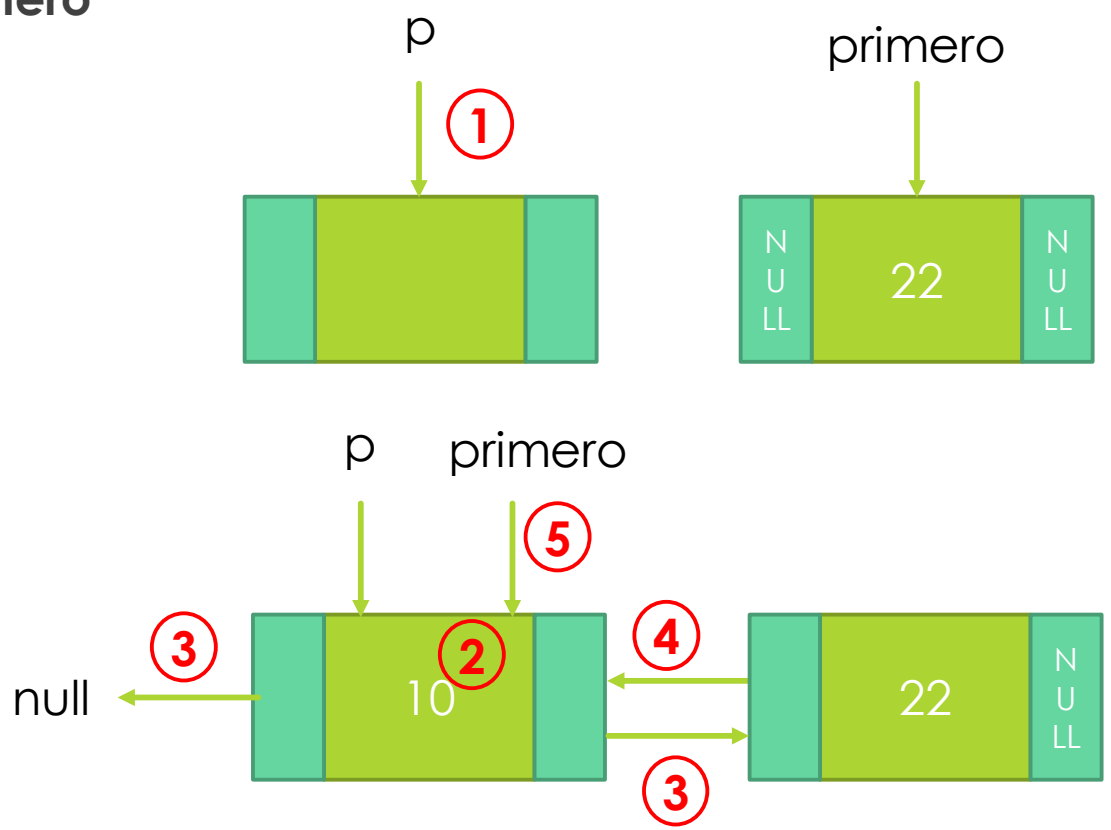
# Insertar de forma ordenada

- **Si vacia -> nuevo nodo primero**



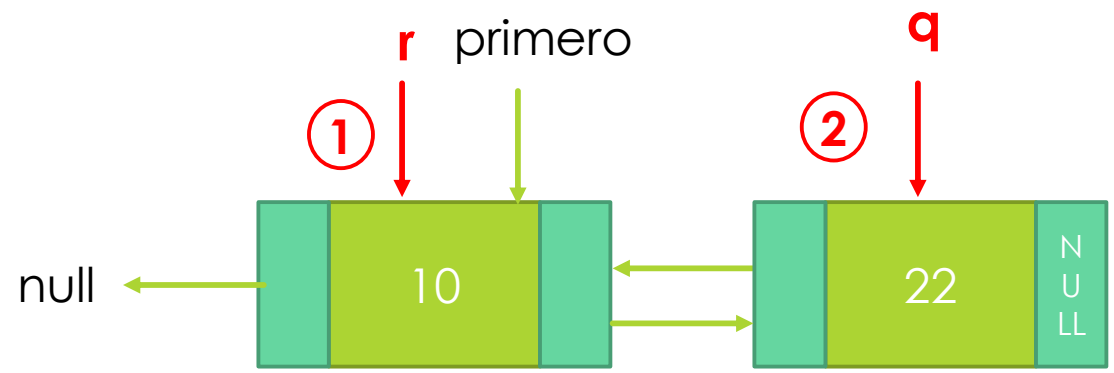
# Insertar de forma ordenada

- **Si x < primero**

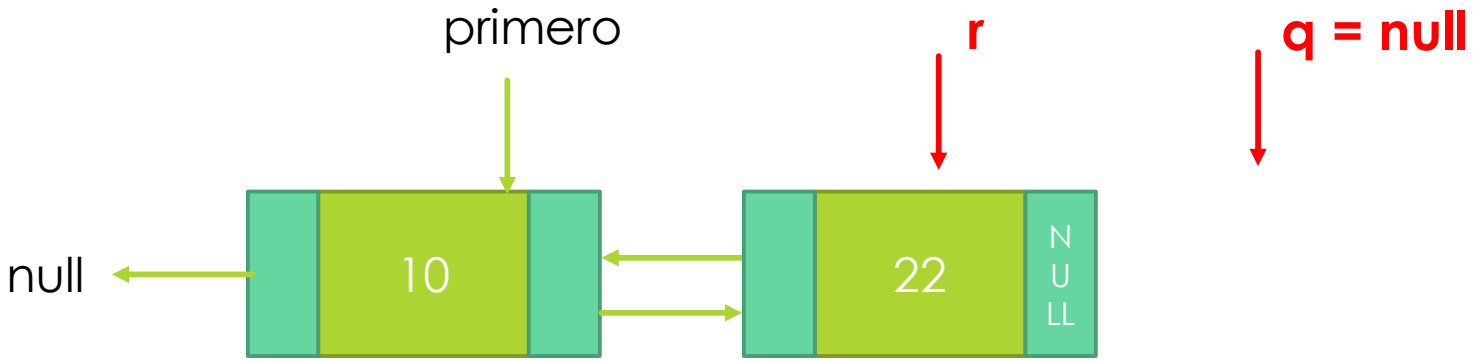


# Insertar de forma ordenada

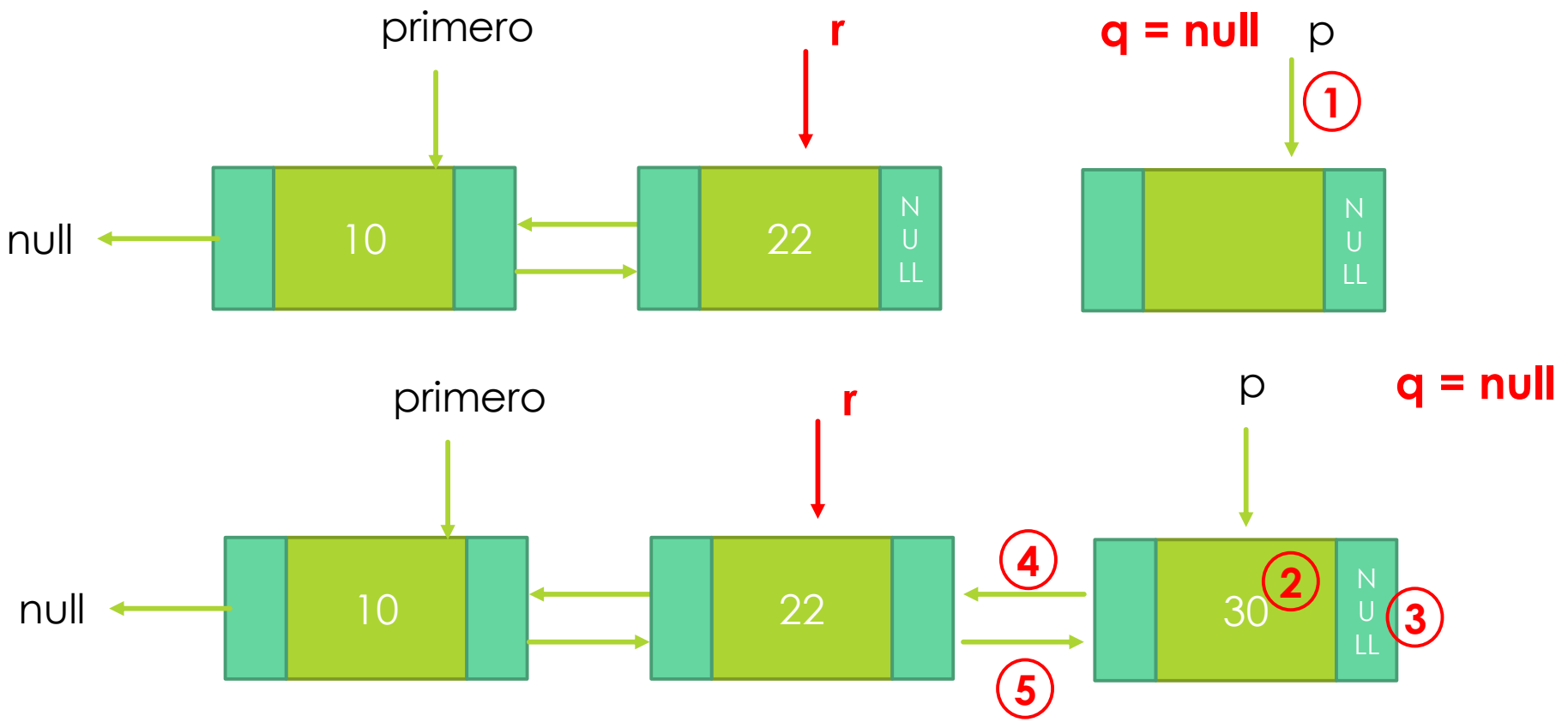
- **Si NO**




Recorremos la lista a la derecha mientras q <> null  && x > q.info.

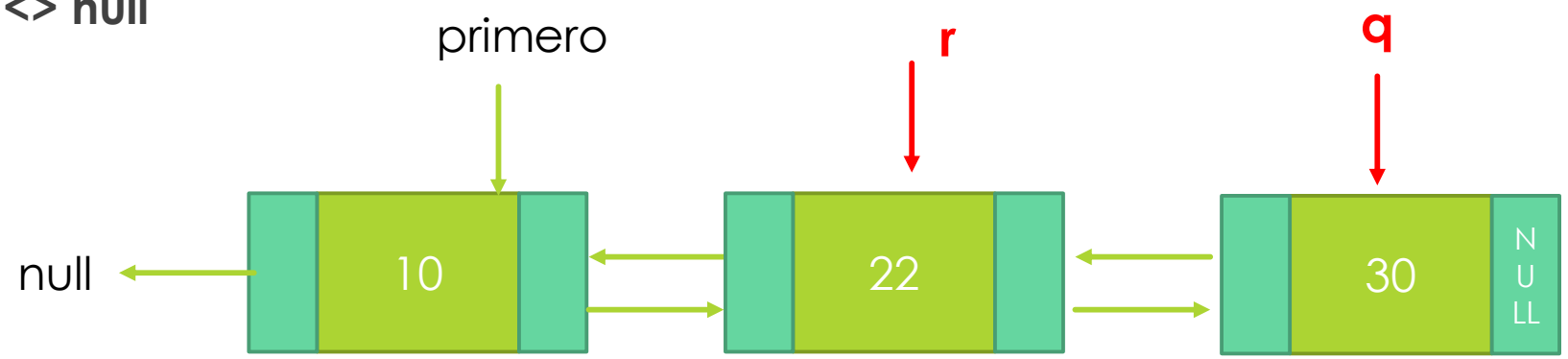


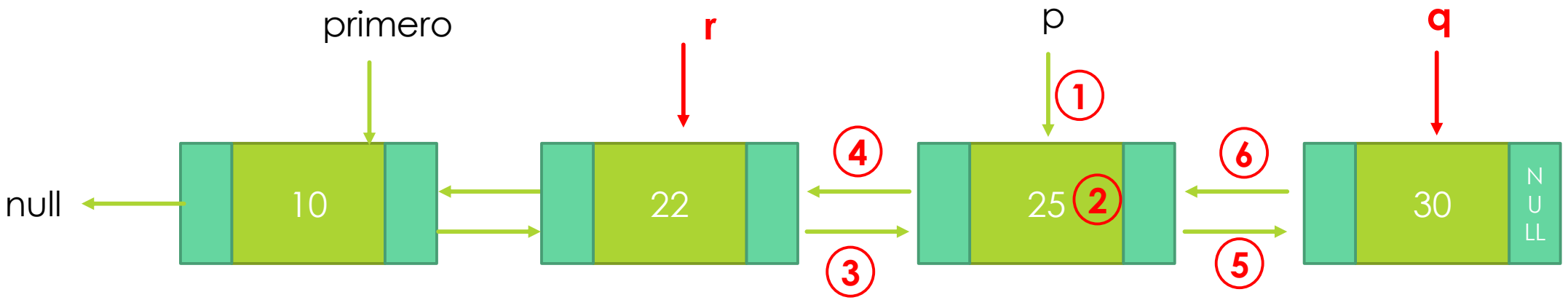
# Insertar de forma ordenada



# Insertar de forma ordenada

- **Si q <> null**



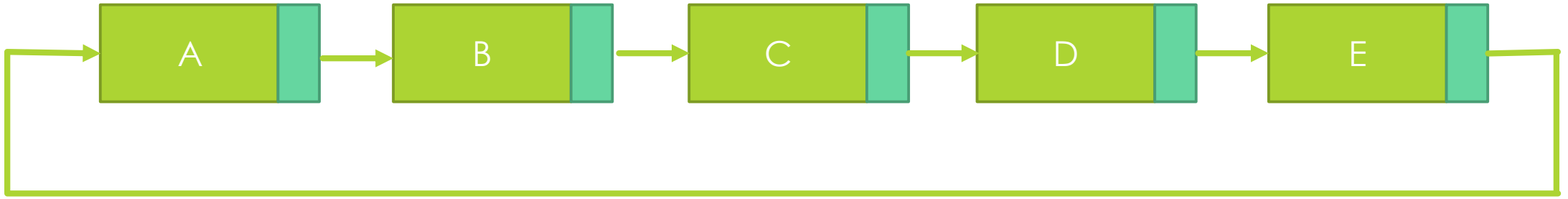



# Listas circulares

- Una lista circular es una lista lineal en la que el último nodo apunta al primero.
- Cada nodo siempre tiene uno anterior y uno siguiente.
- Los enlaces de cada nodo nunca son null.

- Podemos tener 2 tipos de listas circulares:
  - 1. Listas simplemente enlazadas circular
  - 2. Listas doblemente enlazadas circular.

# Listas simplemente enlazadas circular



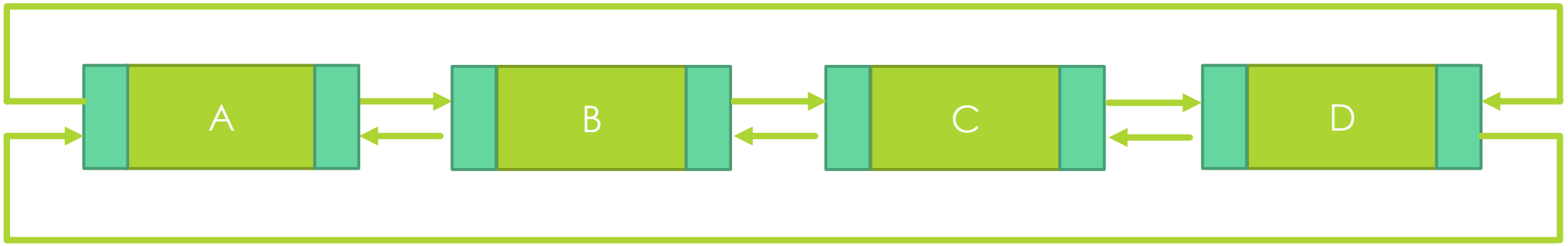

Su estructura es similar a la de las listas simplemente enlazadas.

```
class Nodo
{
    private int info;
    private Nodo enlace;
    8 referencias
    public int Info
    {
        get { return info; }
        set { info = value; }
    }

    11 referencias
    public Nodo Enlace
    {
        get { return enlace; }
        set { enlace = value; }
    }


}
```

# Listas doblemente enlazadas circular



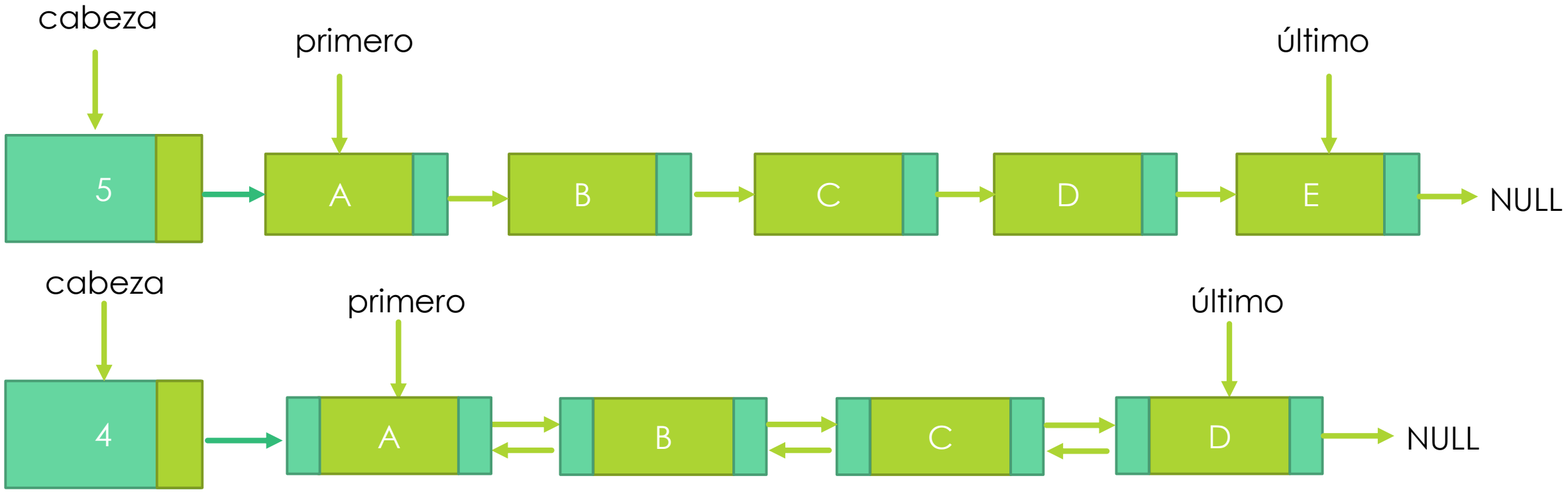

Su estructura es similar a la de las listas doblemente enlazadas.

```
class Nodo
{
    private int info;
    private Nodo enlaceIzquierdo;
    private Nodo enlaceDerecho;

    12 referencias
    public int Info
    {
        get { return info; }
        set { info = value; }
    }
    13 referencias
    public Nodo EnlaceIzquierdo
    {
        get { return enlaceIzquierdo; }
        set { enlaceIzquierdo = value; }
    }
    20 referencias
    public Nodo EnlaceDerecho
    {
        get { return enlaceDerecho; }
        set { enlaceDerecho = value; }
    }
}
```

# Listas con nodo Cabeza

- El nodo cabeza se utiliza para tener información general de la lista o para comenzar una lista.



# Implementación de listas circulares y con nodo cabeza

- Veamos 2 formas de insertar listas simplemente enlazadas circulares.
- Insertar lista doblemente enlazadas con nodo cabeza.

# Trabajo práctico y laboratorio #4

- Realizar los siguientes métodos en C#:
  - Insertar elementos en una lista doblemente enlazada circular.
  - Insertar elementos en una lista simplemente enlazada con nodo cabeza.
  - Eliminar un elemento “x” de una lista simplemente enlazada circular.
  - Eliminar un elemento “x” de una lista doblemente enlazada circular.
  - Eliminar un elemento “x” de una lista doblemente enlazada con nodo cabeza.
  - Eliminar un elemento “x” de una lista simplemente enlazada con nodo cabeza.
  - Crear el formulario con los componentes necesarios, para utilizar los métodos creados en los puntos anteriores y poder ejecutar la aplicación.