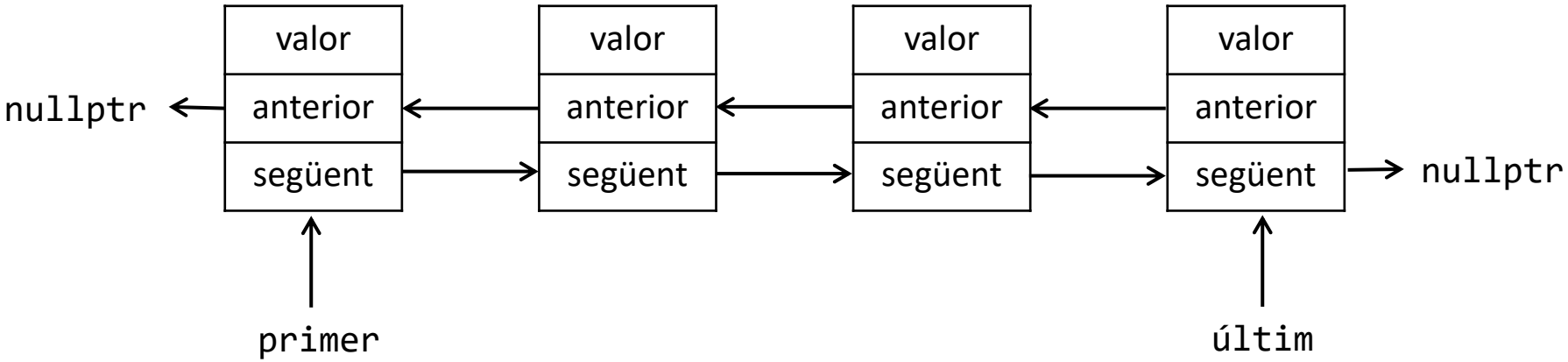
Tema 2 – Exercici Opcional 3

Llistes doblement enllaçades

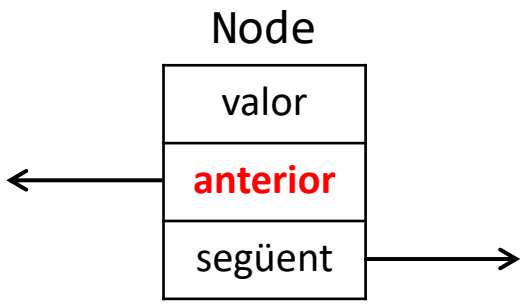
- Permeten recórrer la llista en dues direccions
- Faciliten inserir i eliminar en qualsevol posició
- Cada element té enllaços amb el següent i amb l'anterior element
  - Apuntadors al següent i a l'anterior element
- Hem de guardar on està el primer i l'últim element de la llista:
  - Apuntadors al primer i a l'últim element de la llista
- El següent de l'últim element de la llista i l'anterior del primer element apuntaran a `null`.



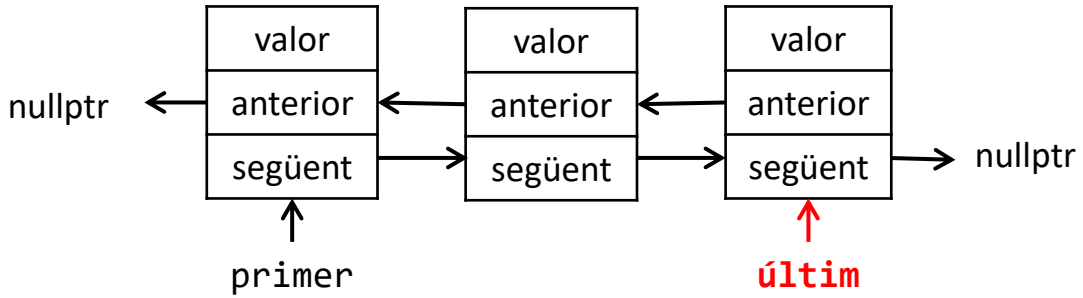
Tema 2 – Exercici Opcional 3

```
class Node
{
public:
   <tipus_llista> getValor();
   Node* getNext();
   Node* getPrev();
   void setValor(<tipus_llista> valor);
   void setNext(Node* next);
   void setPrev(Node* prev);
private:
   <tipus_llista> m_valor;
   Node* m_next;
   Node* m_prev;
};
```

A la classe Node afegim un atribut per guardar l'apuntador a l'element anterior:




```
class Llista
{
public:
   ...
   Node* getInici();
   Node* getFinal();
private:
   Node* m_primer;
   Node* m_ultim;
};
```



A la classe `Llista` afegim un atribut per guardar l'apuntador a l'últim element:

- Podem recuperar directament l'últim element amb `getFinal()`
- Podem recórrer la llista cap enrere amb el mètode `getPrev()` de la classe Node

Tema 2 – Exercici Opcional 3

- En els nostres programes haurem de gestionar diferents estructures dinàmiques enllaçades, cadascuna amb un tipus diferent, però totes amb necessitats similars a nivell de funcionalitat: inserir i eliminar elements, accedir als elements...
- En aquest exercici **implementarem la classe `LlistaDoble`** com un patró comú que permeti gestionar llistes enllaçades de qualsevol tipus amb mínims canvis.
- A la **llibreria estàndar**d de C++ hi ha una classe **`forward_list`** amb una funcionalitat similar:
  http://www.cplusplus.com/reference/forward_list/list/
- Per implementar la classe `LListaDoble` utilitzarem l'estructura de `Node` amb doble apuntador:

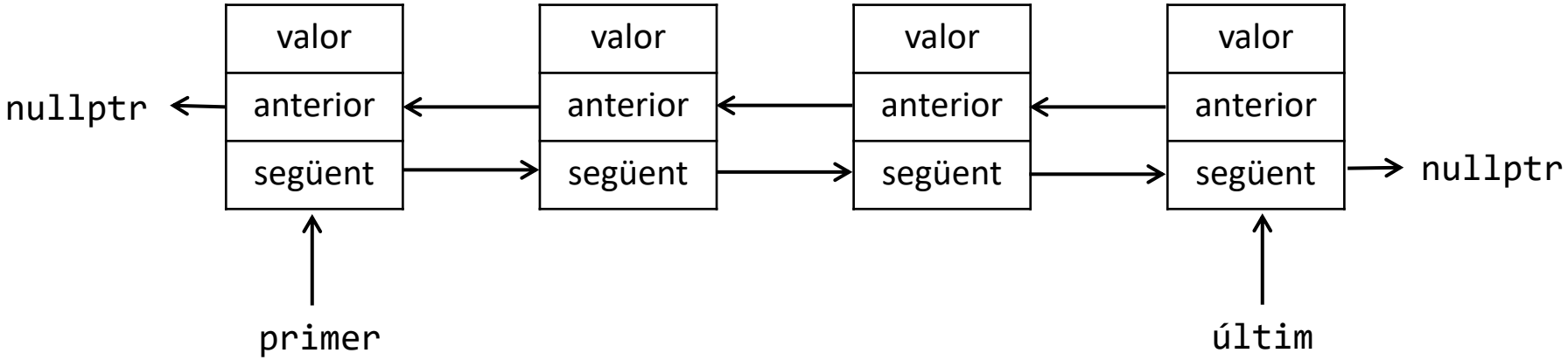
```
class Node
{
public:
   <tipus>& getValor();
   Node* getNext();
   void setValor(const <tipus>& valor);
   void setNext(Node* next);
 private:
   <tipus> m_valor;
   Node* m_next;
};
```

Canviant el tipus del valor podem implementar llistes que permetin guardar objectes de qualsevol tipus. En aquest exercici suposarem que la llista guarda objectes de **tipus `int`.**

Tema 2 – Exercici Opcional 3

```
class LlistaDoble
{
    public:
        LlistaDoble();
        ~LlistaDoble();

        bool empty() const;
        Node* begin() const;
        Node* rbegin() const;
        Node* insert(int valor, Node* posicio);
        Node* erase(Node* posicio);
        void unique();
        LlistaDoble& operator=(const LlistaDoble& llista);
    private:
        Node* m_primer;
        Node* m_ultim;
};
```



Tema 2 – Exercici Opcional 3

Implementació de la classe LlistaDoble: mètodes

- El mètodes **`begin`** i **`rbegin`** recuperen un apuntador al primer i a l'últim element de la llista, respectivament.
- El mètode **`empty`** ha de retornar un booleà indicant si la llista està buida o no.
- El mètode **`insert`** afegeix un element a la posició anterior del node passat com a paràmetre. Si l'apuntador que es passa com a paràmetre és `nullptr` s'ha d'afegir al final de tot de la llilsta. Retorna un apuntador a l'element que s'ha afegit.
- El mètode **`erase`** elimina l'element que ocupa la posició del node que es passa com a paràmetre. Si l'apuntador que es passa com a paràmetre és `nullptr` no s'ha de fer res. Retorna un apuntador a l'element següent de l'element eliminat.
- **L'operador d'assignació** ha de copiar (assignant nova memòria) tots els elements de la llista que es passa com a paràmetre a la llista actual, alliberant primer tots els elements que hi hagués originalment.
- El mètode **`unique`** elimina elements repetits consecutius, és a dir, de cada grup d'elements consecutius iguals, els elimina tots menys el primer. Per tant, eliminarà tots els elements que siguin iguals a l'element immediatament anterior.

http://www.cplusplus.com/reference/list/list/unique/

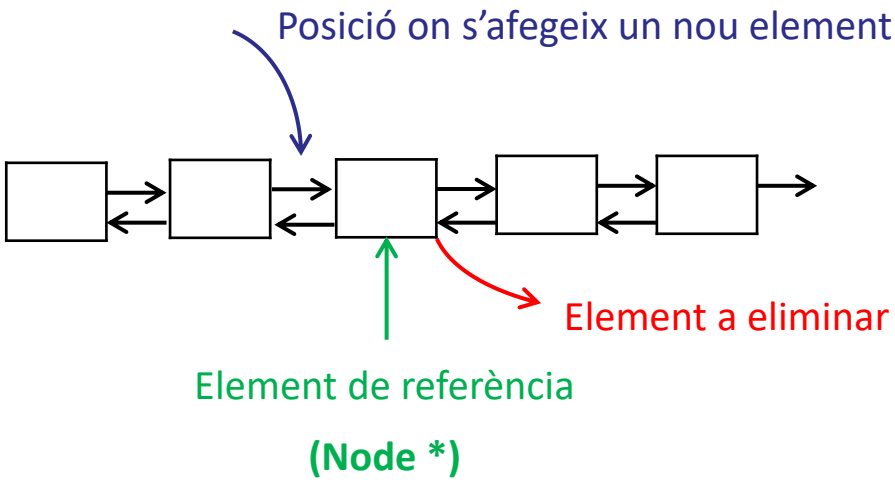
Tema 2 – Exercici Opcional 3

En una llista hem de poder:

- Afegir un element a una posició qualsevol de la llista.
  - L'element nou s'afegirà en la posició anterior a la que ocupa un element de referència donat.
- Eliminar un element a una posició qualsevol de la llista.
  - L'element que s'eliminarà és el que ocupa la posició d'un element de referència donat.

⇩

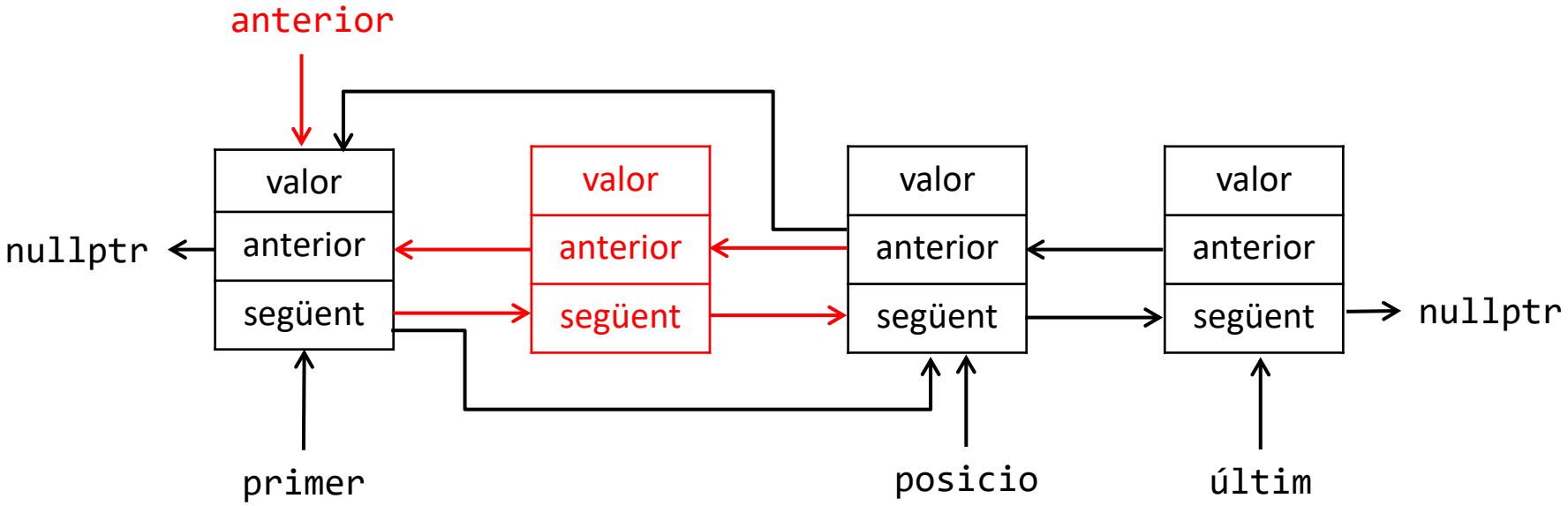
Necessitem algun mètode per identificar una posició qualsevol de la llista



Tema 2 – Exercici Opcional 3

```
Node* insert(int valor, Node* posicio);
```

Inserir element al mig de la llista (`posicio != nullptr, anterior != nullptr`)

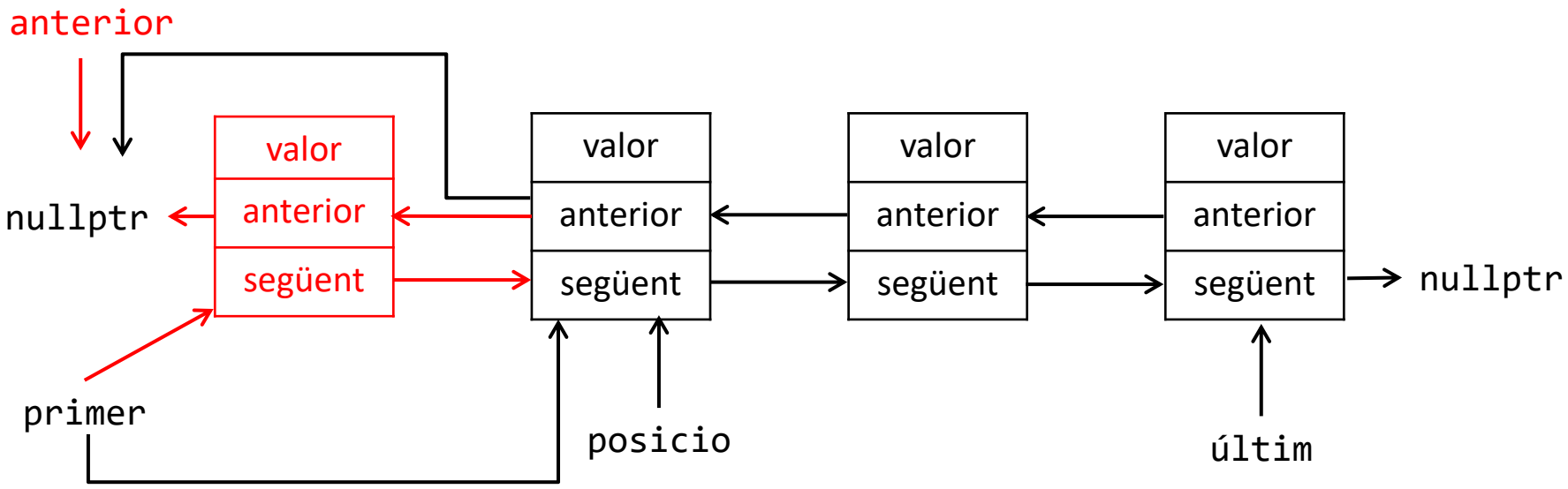


1. Recuperar apuntador a l'element anterior
2. Crear i inicialitzar el nou node
3. Enllaçar el nou node amb l'element anterior i l'element de referencia
4. Modificar l'apuntador següent del node anterior i l'apuntador anterior del node de referència

Tema 2 – Exercici Opcional 3

```
Node* insert(int valor, Node* posicio);
```

Inserir element al principi (`posicio != nullptr, anterior == nullptr`)

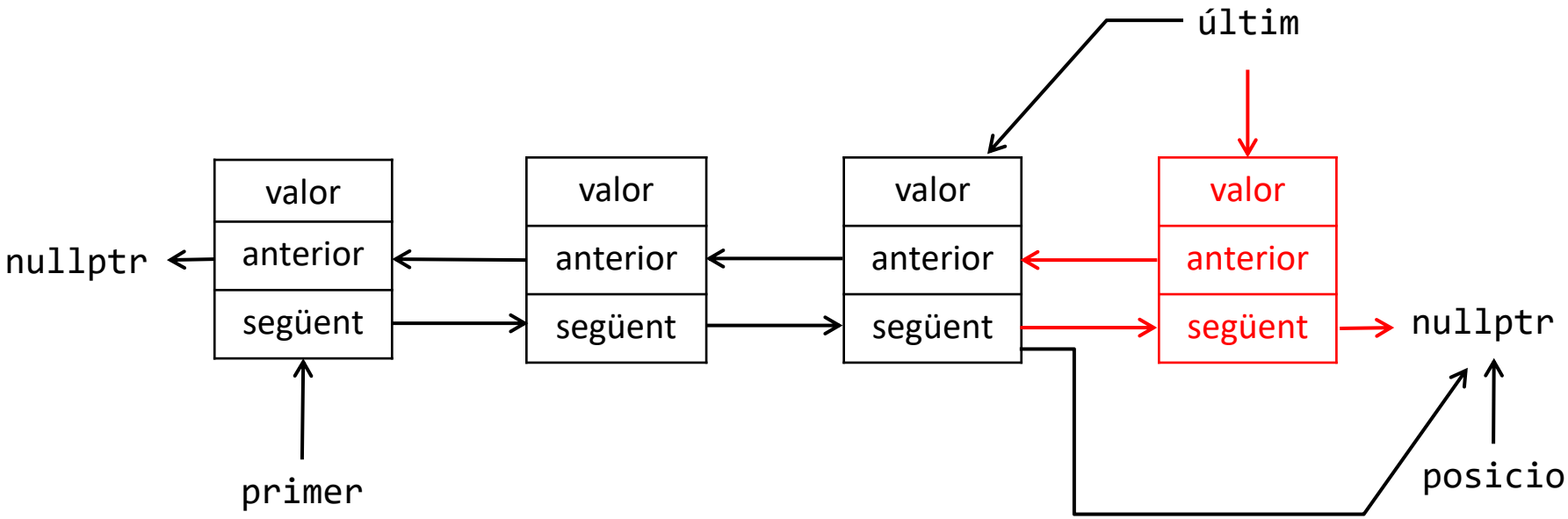


1. Recuperar apuntador a l'element anterior
2. Crear i inicialitzar el nou node
3. Enllaçar el nou node amb l'element anterior i l'element de referencia
4. Modificar l'apuntador anterior del node de referència
5. Modificar l'apuntador al primer fent que passi a apuntar al nou node

Tema 2 – Exercici Opcional 3

```
Node* insert(int valor, Node* posicio);
```

Inserir element al final de la llista (`posicio == nullptr`)

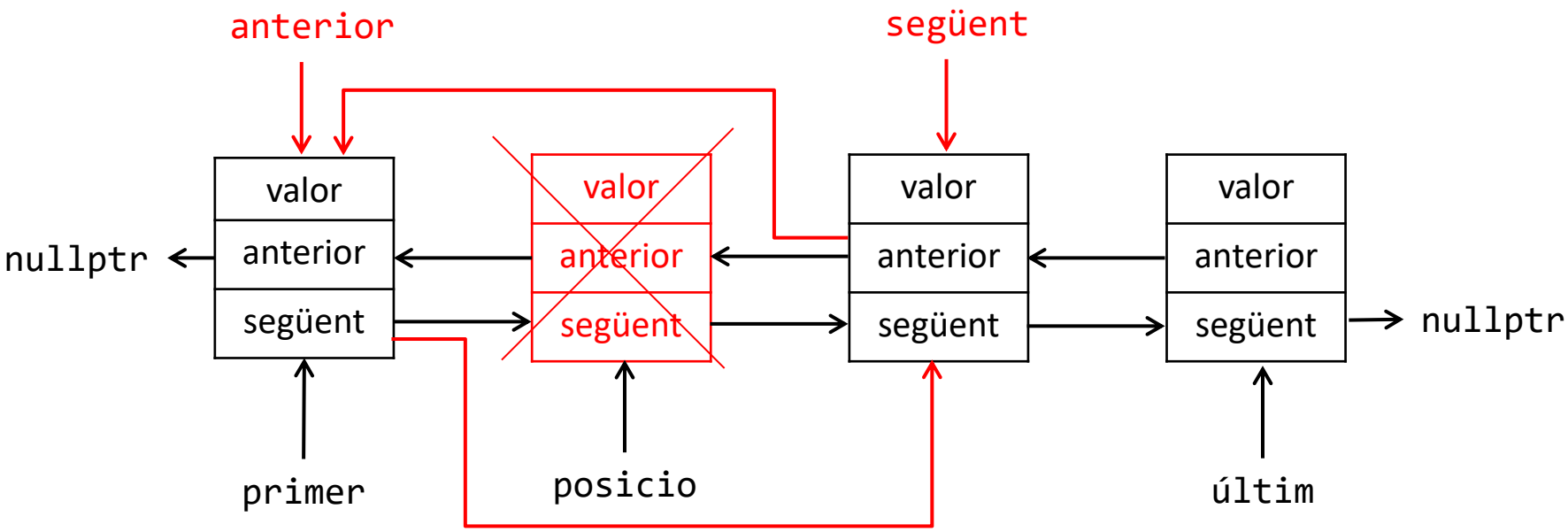


1. Crear i inicialitzar el nou node
2. Enllaçar el nou node amb l'últim element (anterior) i nullptr (següent)
3. Modificar l'apuntador següent de l'últim node
4. Modificar l'apuntador al últim node fent que passi a apuntar al nou node

Tema 2 – Exercici Opcional 3

```
Node* erase(Node* posicio);
```

Eliminar element del mig de la llista

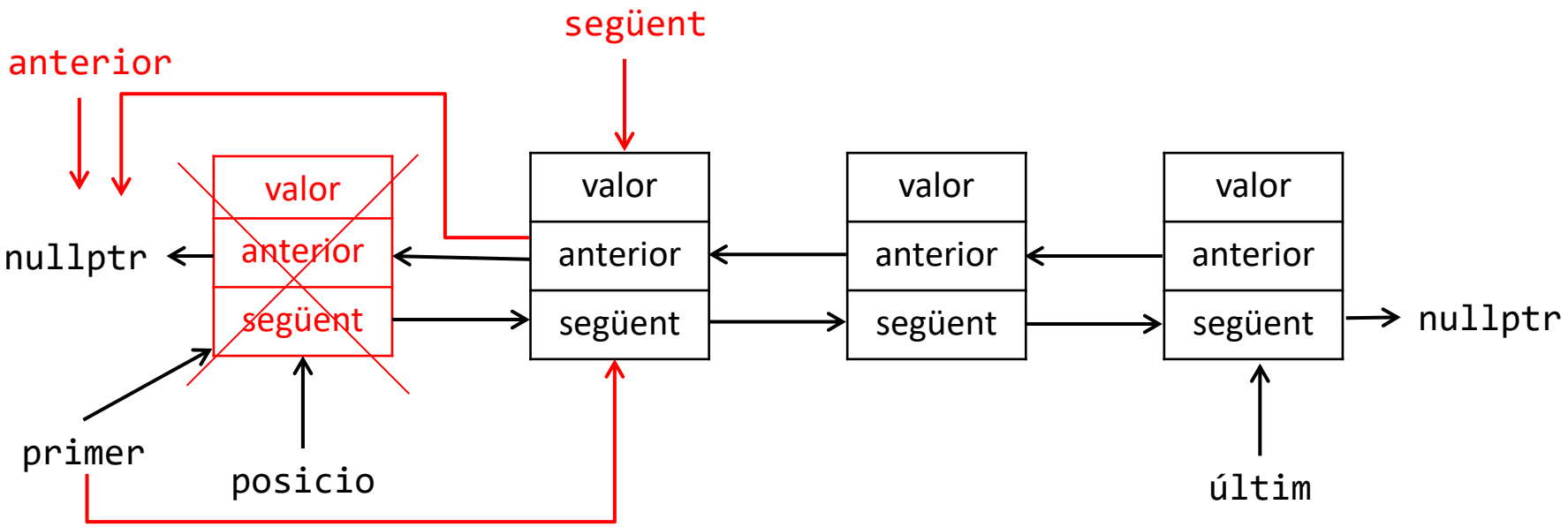


1. Recuperar apuntadors a l'element anterior i al següent
2. Modificar l'apuntador següent del node anterior i l'apuntador anterior del node següent
3. Alliberar el node de l'element a eliminar

Tema 2 – Exercici Opcional 3

```
Node* erase(Node* posicio);
```

Eliminar el primer element de la llista

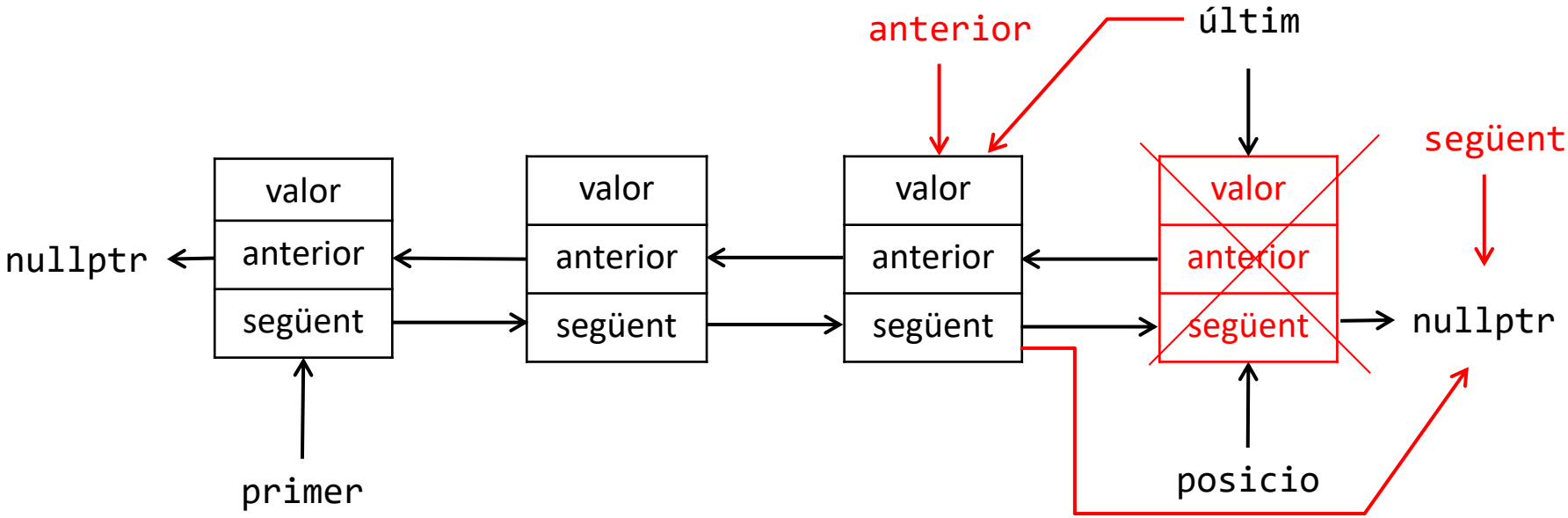


1. Recuperar apuntadors a l'element anterior i al següent
2. **Si l'element anterior és nullptr, estem eliminant el primer element. Modificar l'apuntador al primer element**
3. Modificar l'apuntador anterior del node següent
4. Alliberar el node de l'element a eliminar

Tema 2 – Exercici Opcional 3

```
Node* erase(Node* posicio);
```

Eliminar l'últim element de la llista



1. Recuperar apuntadors a l'element anterior i al següent
2. **Si l'element següent és nullptr, estem eliminant l'últim element. Modificar l'apuntador a l'últim element**
3. Modificar l'apuntador següent del node anterior
4. Alliberar el node de l'element a eliminar