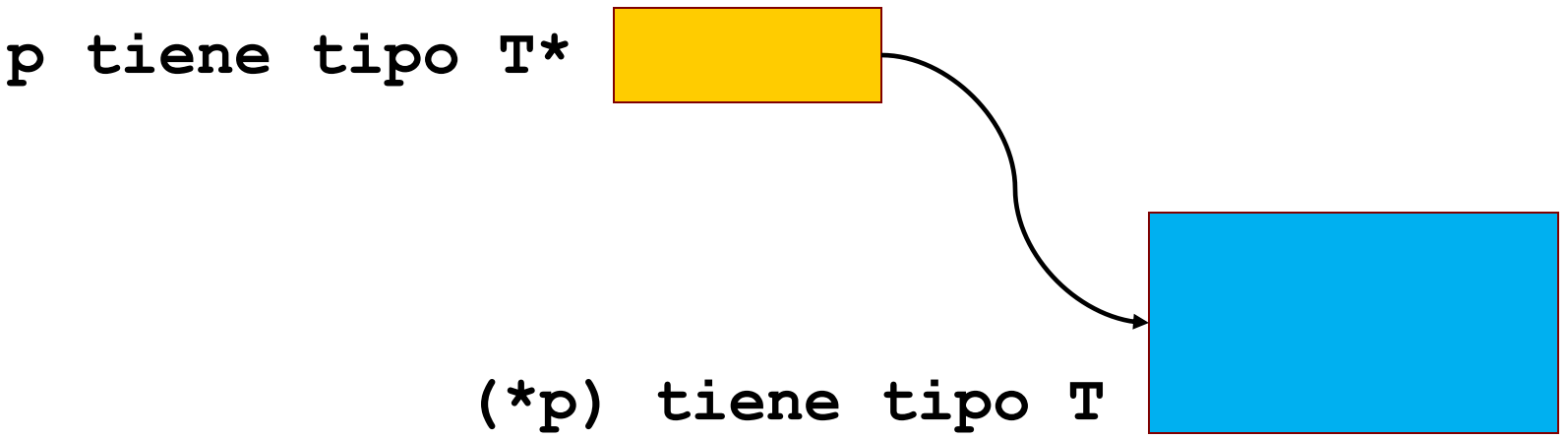
Programación 2

La Previa:

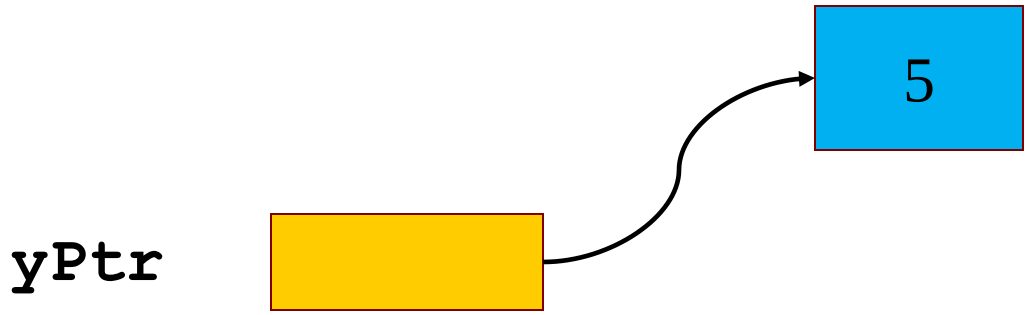
Tipos Inductivos (Recursivos)

Estructuras Dinámicas: Punteros y Listas

Punteros



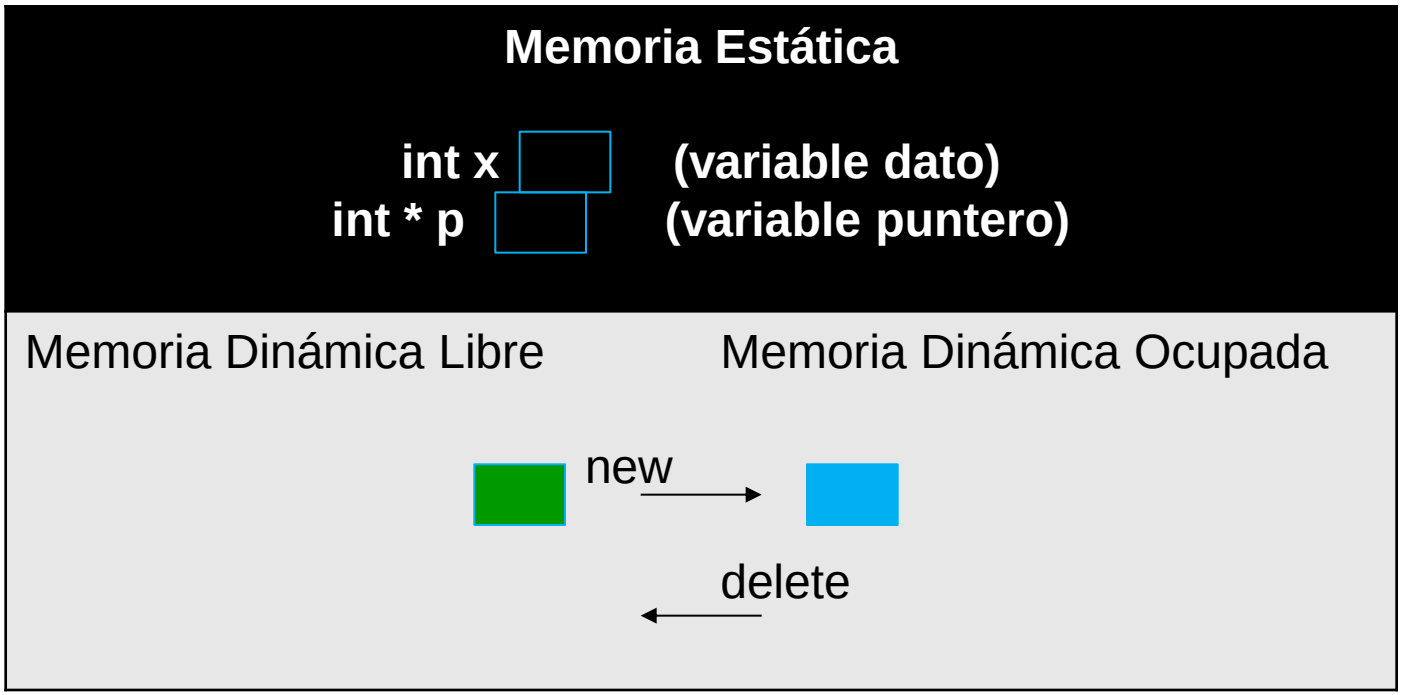
Operador de dirección




```
int y = 5;
int* yPtr;
yPtr = &y;
```

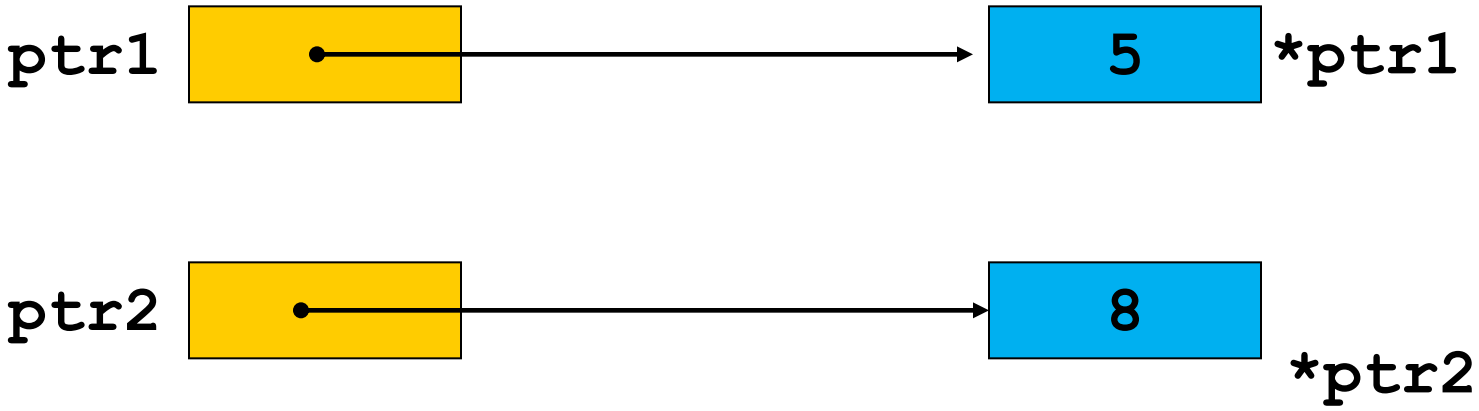
Asignación dinámica de memoria

Administración de la memoria estática y dinámica de un programa



Operaciones sobre Punteros

```
int * ptr1 = new int; *ptr1 = 5
int * ptr2 = new int; *ptr2 = 8
```



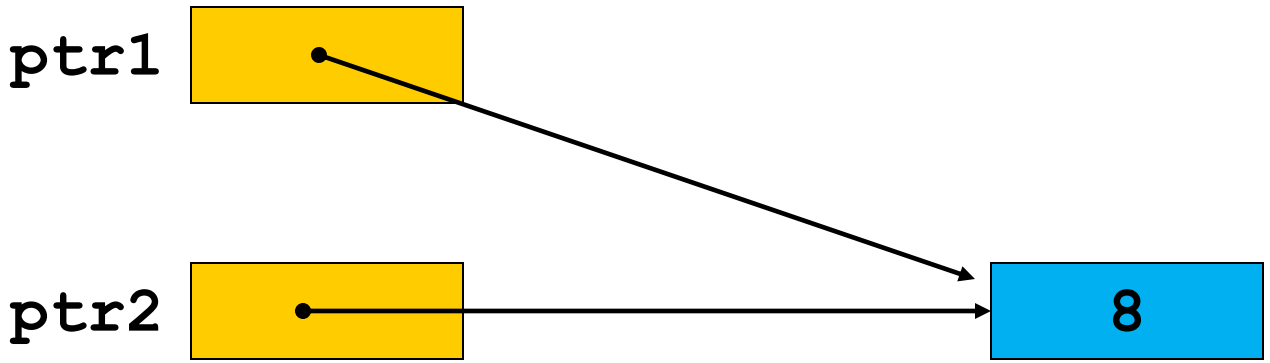
Comparar

```
ptr1 = ptr2;
```

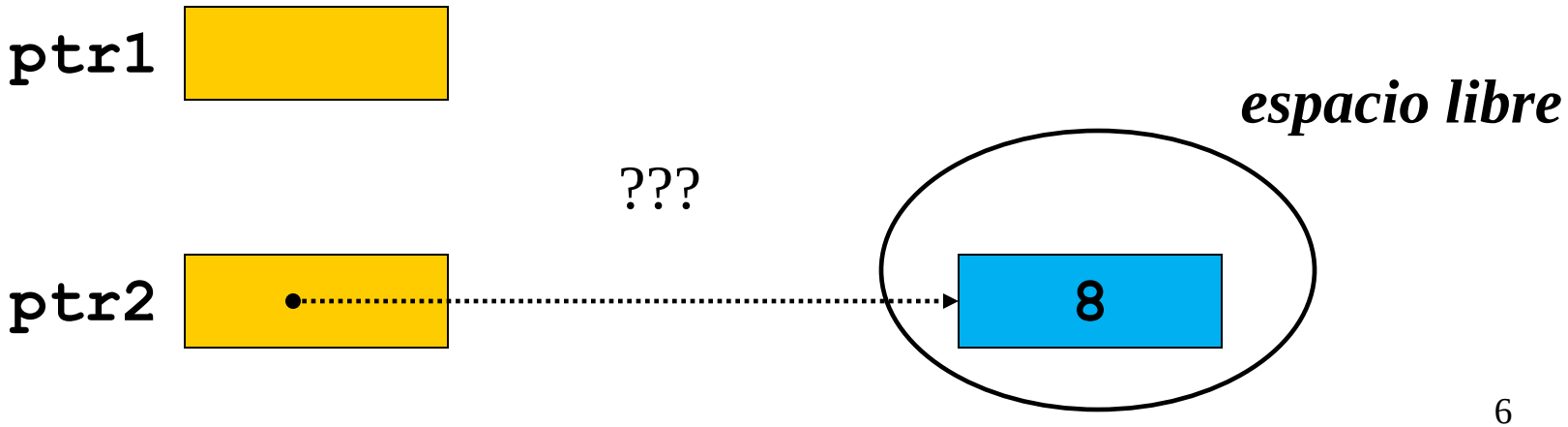
Con

```
*ptr1 = *ptr2;
```

Liberación de memoria



delete ptr1;



Cuatro formas de modificar un puntero

- Usar el procedimiento estandar **`new`**
- Asignar la dirección contenida en otro puntero del mismo tipo
- Usar el operador &
- Asignar el valor NULL

Arreglos/Vectores

¿Cómo se definen arreglos de un tamaño determinado por una constante (conocido en tiempo de compilación)?

int arreglo[cte];

// de 0 a cte-1

¿Cómo se definen arreglos de un tamaño determinado por un valor variable (en tiempo de ejecución)?

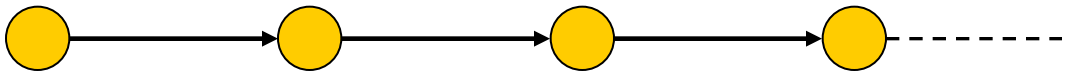
int * arreglo = new int[var];

// de 0 a var-1

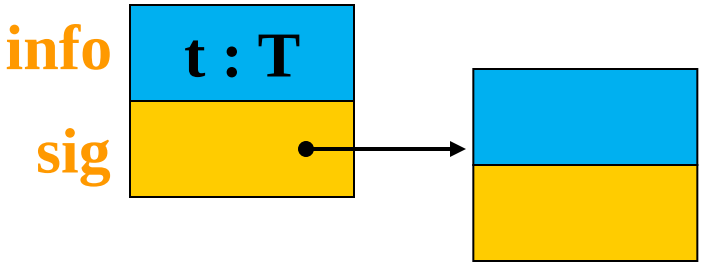
¿Cómo operar con arreglos?

arreglo[i]

Lista simplemente encadenada



```
struct nodoLista {
        T info;
        nodoLista* sig;
 }
```

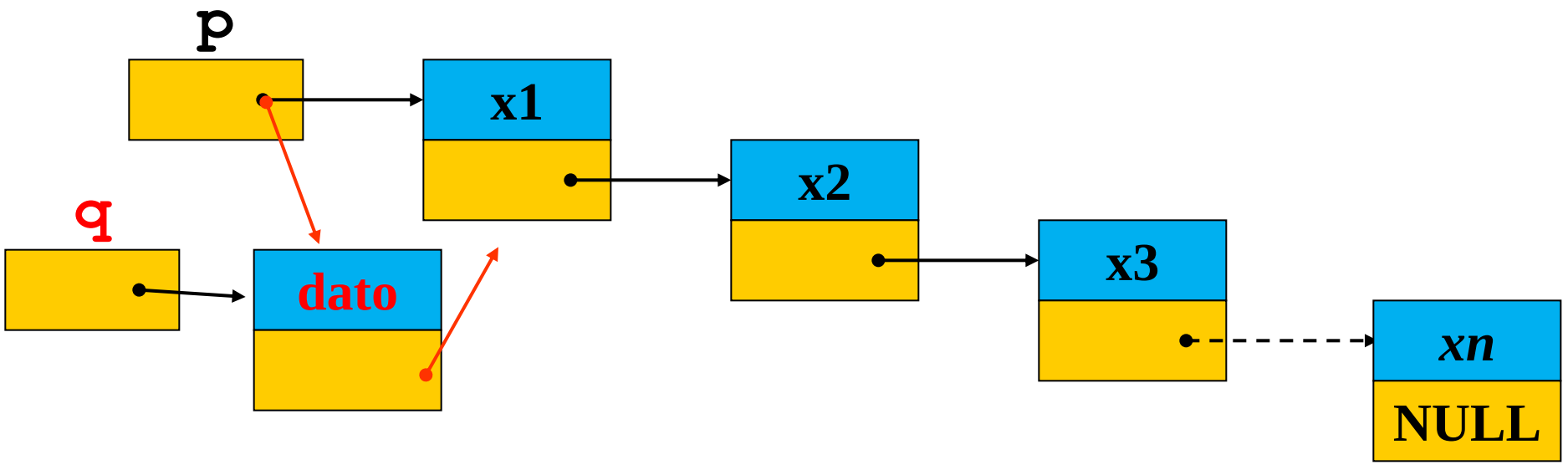



```
typedef nodoLista* Lista;
```

(Notar la autoreferencia)

Insertar en una lista

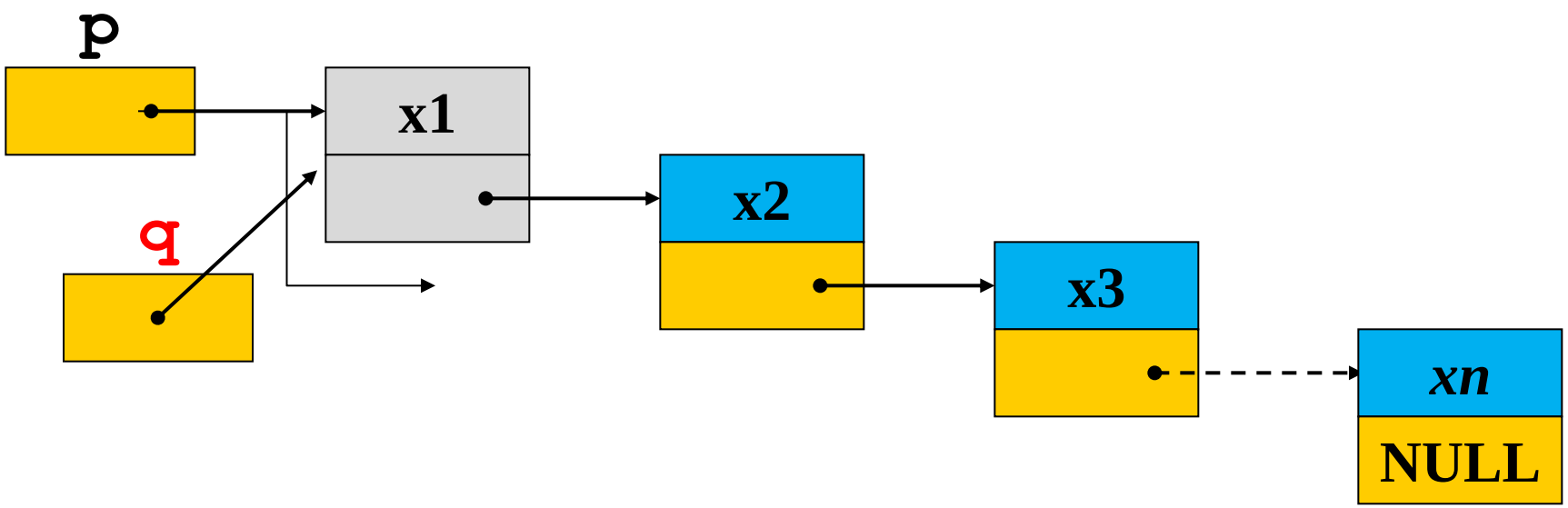
```
q = new nodoLista;
q -> info = dato; //(*q).info = …
q -> sig = p;
p = q;
```



Borrar en una lista

Para borrar el primer elemento de la lista hacemos:

```
q = p;
p = p -> sig;// borrado lógico
delete q;     // borrado físico
```

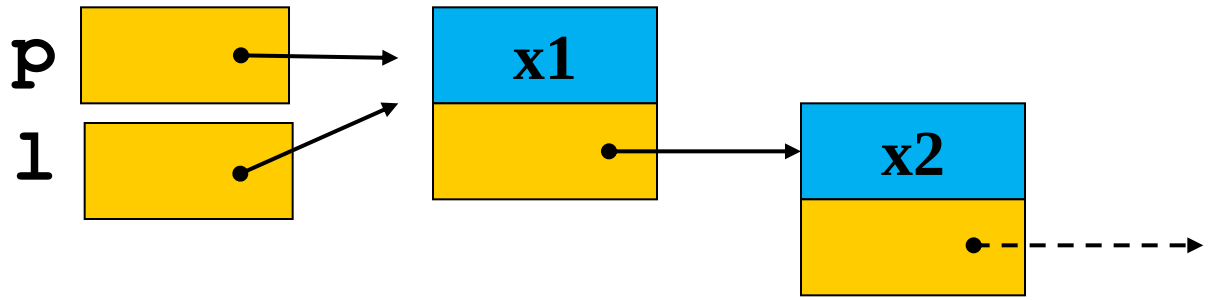


Ejemplos simples con Listas

Considerar qué pasa si se invoca a los siguientes procedimientos con una lista no vacía p:

```
void P1 (Lista l) { l = NULL; }
void P2 (Lista &l) { l = NULL; }
void P3 (Lista &l) { l -> sig = NULL; }
void P4 (Lista l) { l -> sig = NULL; }
```

Cómo queda luego p en cada caso? Analice P1 primero luego del pasaje de parámetro:

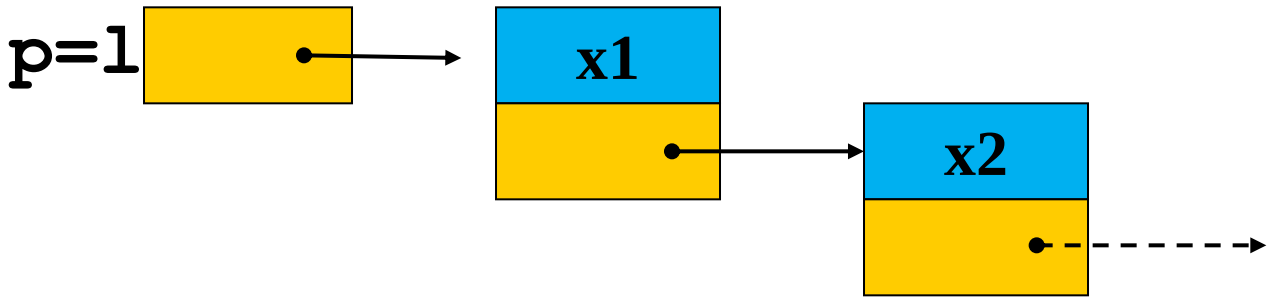


Ejemplos simples con Listas

Considerar qué pasa si se invoca a los siguientes procedimientos con una lista no vacía p:

```
void P1 (Lista l) { l = NULL; }
void P2 (Lista &l) { l = NULL; }
void P3 (Lista &l) { l -> sig = NULL; }
void P4 (Lista l) { l -> sig = NULL; }
```

Cómo queda luego p en cada caso? Analice P2 ahora luego del pasaje de parámetro:

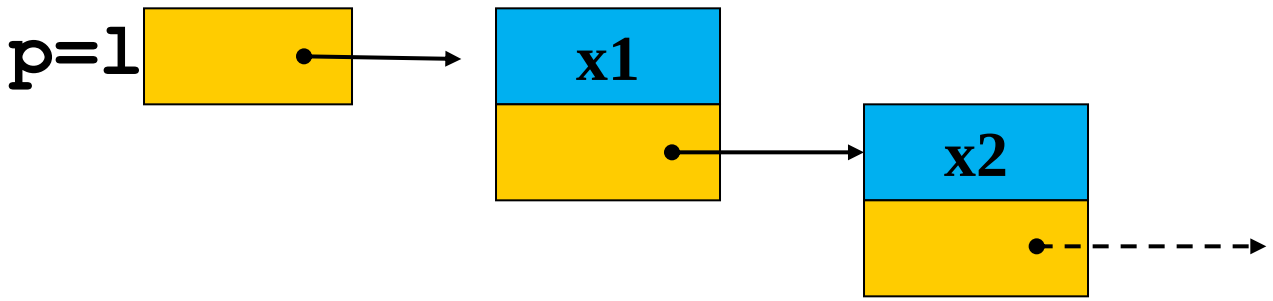


Ejemplos simples con Listas

Considerar qué pasa si se invoca a los siguientes procedimientos con una lista no vacía p:

```
void P1 (Lista l) { l = NULL; }
void P2 (Lista &l) { l = NULL; }
void P3 (Lista &l) { l -> sig = NULL; }
void P4 (Lista l) { l -> sig = NULL; }
```

Cómo queda luego p en cada caso? Analice P3 ahora luego del pasaje de parámetro:

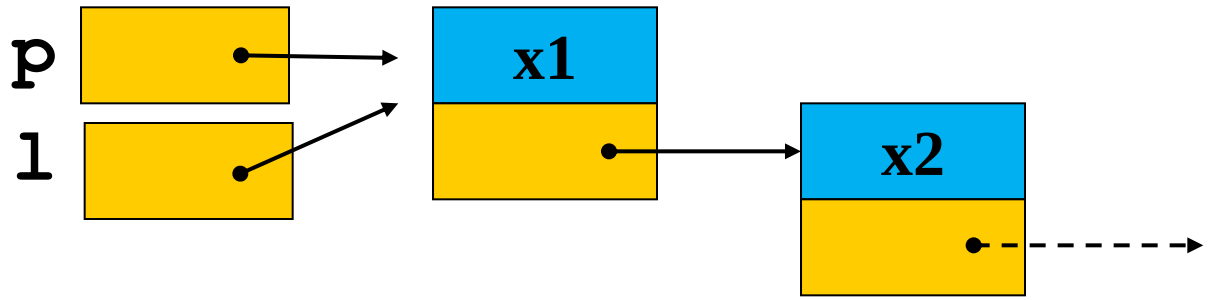


Ejemplos simples con Listas

Considerar qué pasa si se invoca a los siguientes procedimientos con una lista no vacía p:

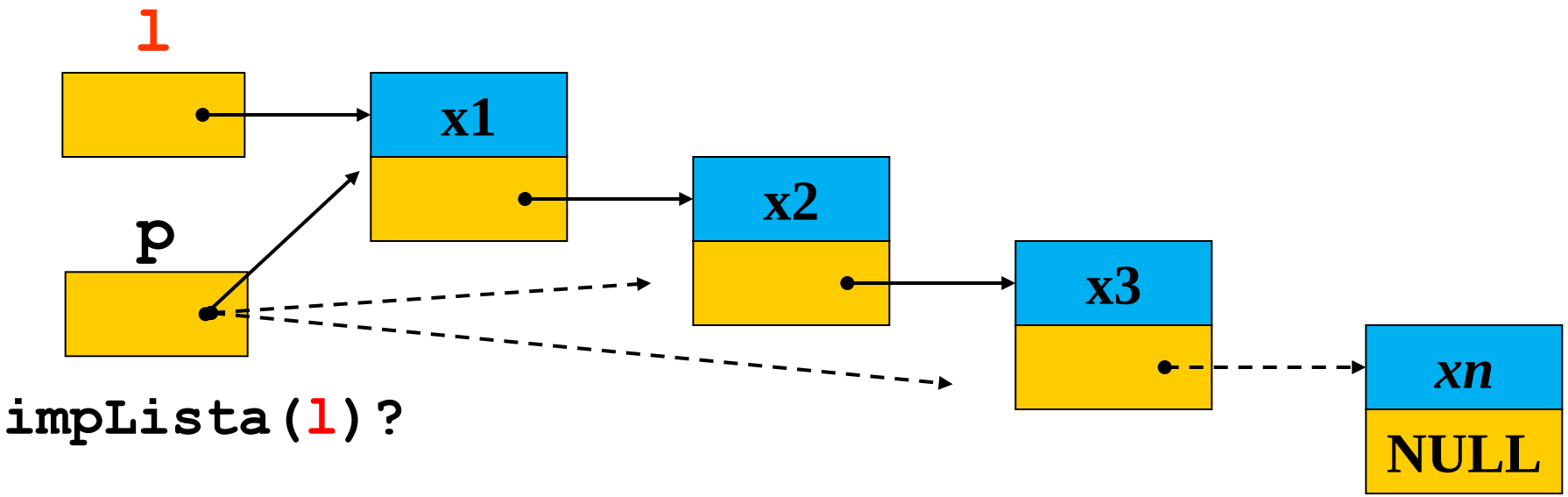
```
void P1 (Lista l) { l = NULL; }
void P2 (Lista &l) { l = NULL; }
void P3 (Lista &l) { l -> sig = NULL; }
void P4 (Lista l) { l -> sig = NULL; }
```

Cómo queda luego p en cada caso? Analice P4 ahora luego del pasaje de parámetro:



Impresión de una lista

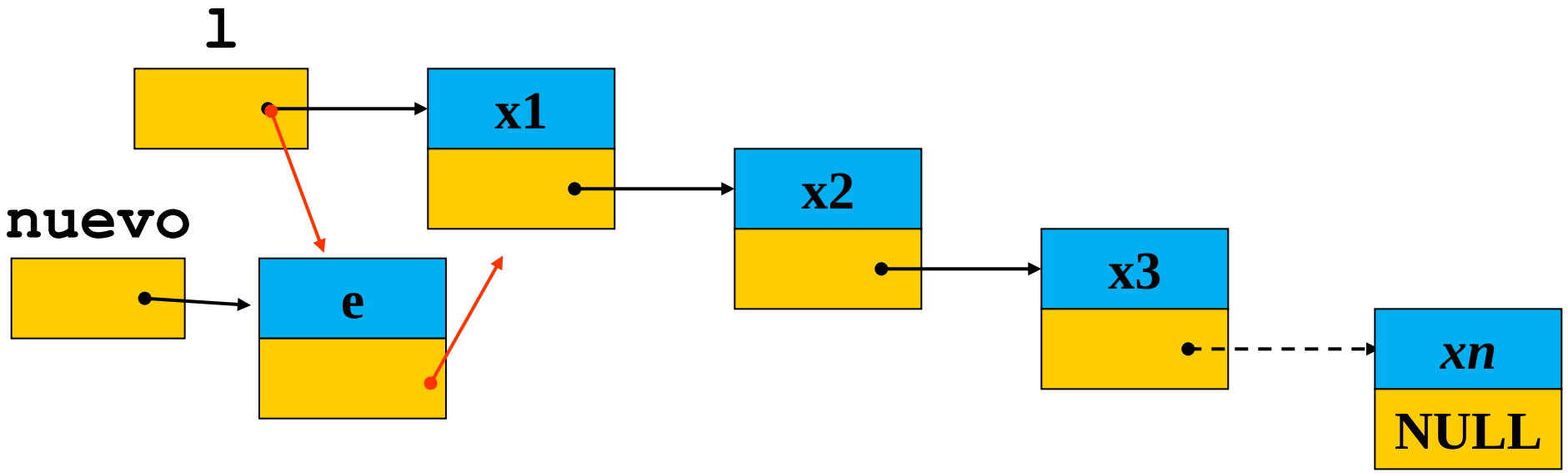
```
void impLista(Lista p){
  while (p != NULL) {
    impDatos(p -> info);
    p = p -> sig;
  }
}
```



`impLista(l)?`

Inserción al comienzo de una lista

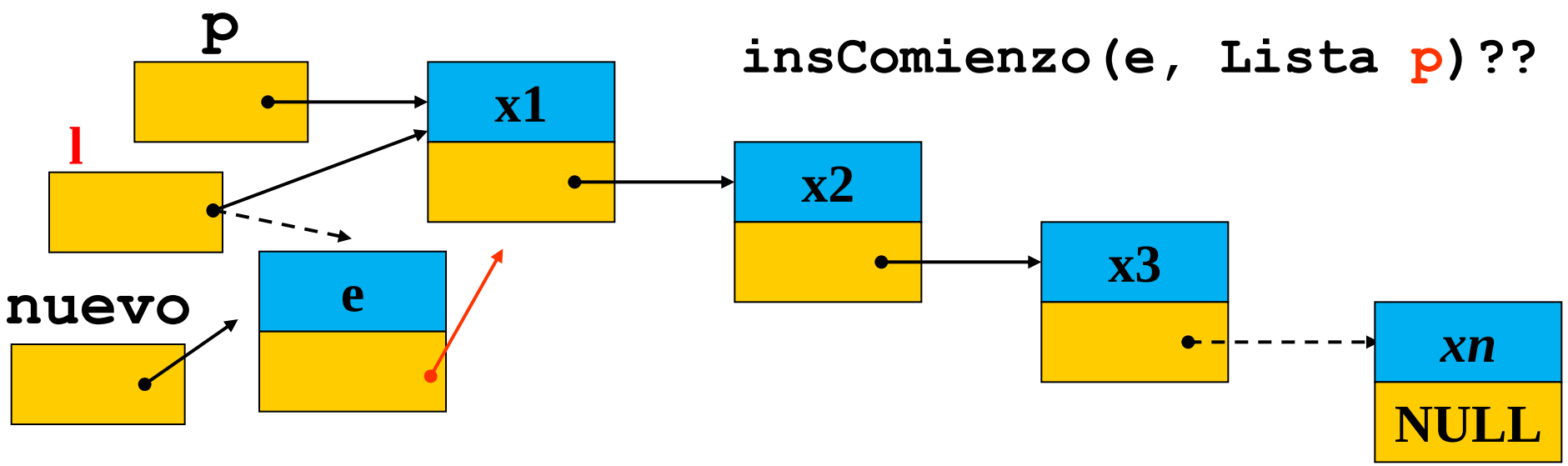
```
void insComienzo(A e, Lista & l){
   Lista nuevo = new nodoLista;
   nuevo -> info = e;
   nuevo -> sig = l;
   l = nuevo;
}
```



Inserción al comienzo de una lista ??

```
void insComienzo(A e, Lista l){//por copia
   Lista nuevo = new nodoLista;
   nuevo -> info = e;
   nuevo -> sig = l;
   l = nuevo;
}
```

insComienzo(e, Lista p)??



Concatenar dos listas

Usando la notación de tipos inductivos escribir en pseudocódigo la función *concat*.

[] **x.S**

concat: ALista x ALista → ALista

concat ([], l2) = …..

concat (x.S, l2) = ….. concat (S, l2)

Concatenar dos listas (cont.)

concat: ALista x AListа → Alista

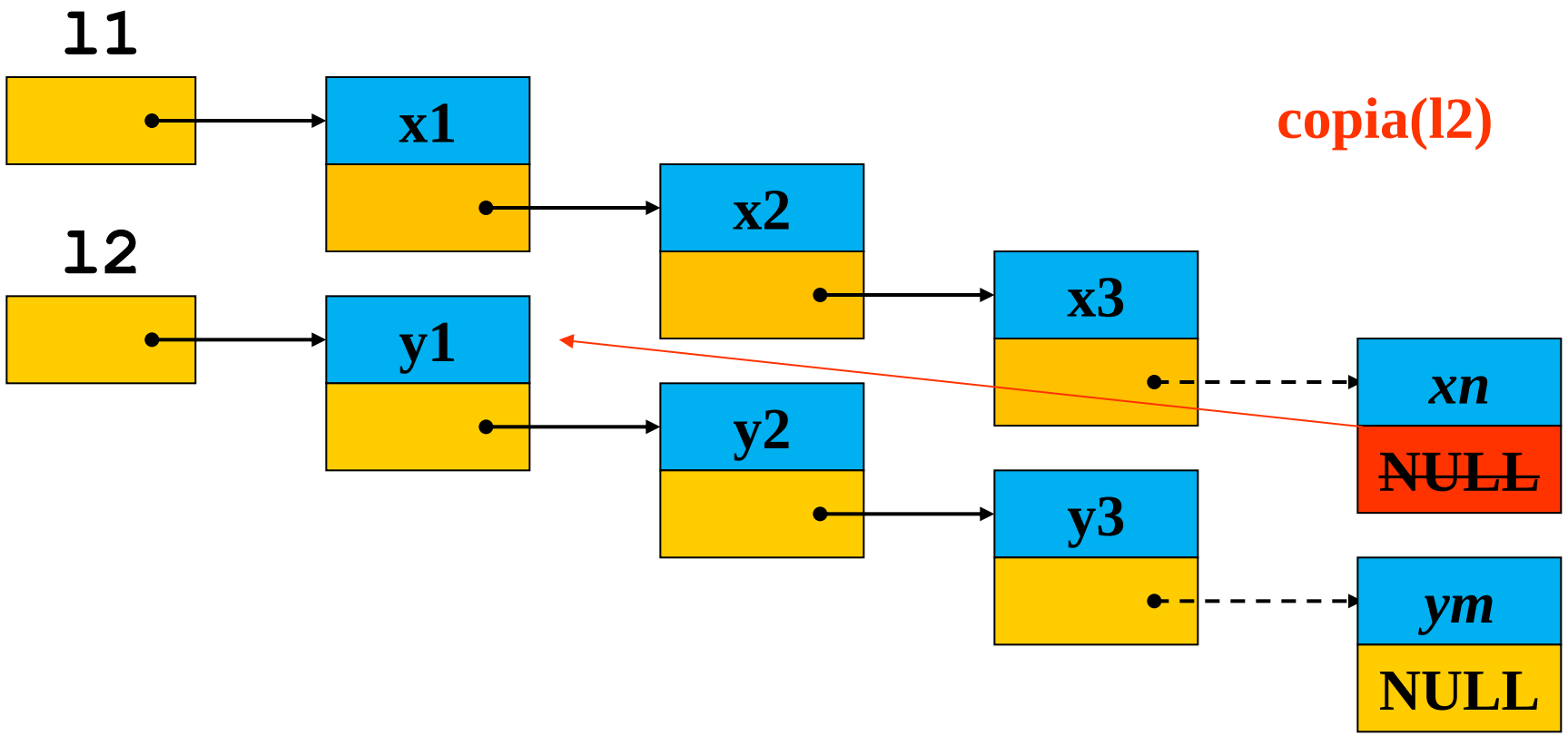
concat ([], l2) = l2

concat (x.S, l2) = x.concat (S, l2)

```
void concat(Lista & l1, Lista l2){
  if (l1 == NULL) l1 = l2;
  else concat(l1 -> sig, l2);
}
```

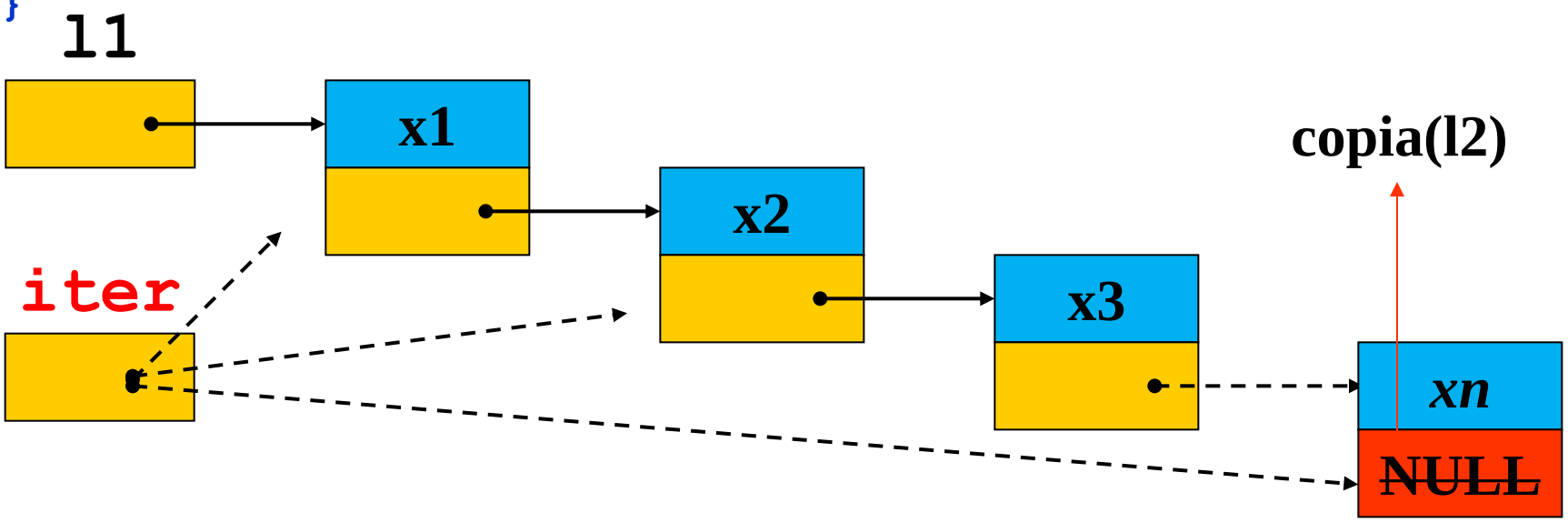
Concatenar dos listas (cont.)

```
void concat(Lista & l1, Lista l2){
  if (l1 == NULL) l1 = l2; //l1 = copia(l2)?
  else concat(l1 -> sig, l2);
}
```



Concatenar dos listas (cont.)

```
//iterativa
void concat(Lista & l1, Lista l2){
  if (l1 == NULL) l1 = copia(l2);
  else{
       Lista iter = l1;
       while (iter->sig != NULL){
              iter = iter->sig;
       }
       iter->sig = copia(l2);
}
```



Inserción ordenada

Insertar de manera ordenada un elemento en una lista ordenada (visto en el resumen de teórico pasado).

insOrd: A x ALista → ALista

insOrd (e, []) = e.[]

insOrd (e, x.S) = e.x.S, Si e<=x

insOrd (e, x.S) = x.insOrd(e, S), Sino

```
void insOrd(A e, Lista & l){
  if (l == NULL) insComienzo(e, l);
  else{
     if (e <= l->info) insComienzo(e, l);
     else insOrd(e, l->sig);
  }
}
```

Insert Sort

Ordenar una lista usando la función previa insOrd.

Ord: ALista → ALista

Ord ([]) = []

Ord (x.S) = insOrd(x, Ord(S))

```
// Insert Sort iterativa pero usando insOrd
Lista Ord(Lista l){
  Lista lres = NULL;
  while (l != NULL){
      insOrd(l->info, lres);
      l = l->sig;
  }
  return lres;
}
```

Eliminar el último elemento de una lista

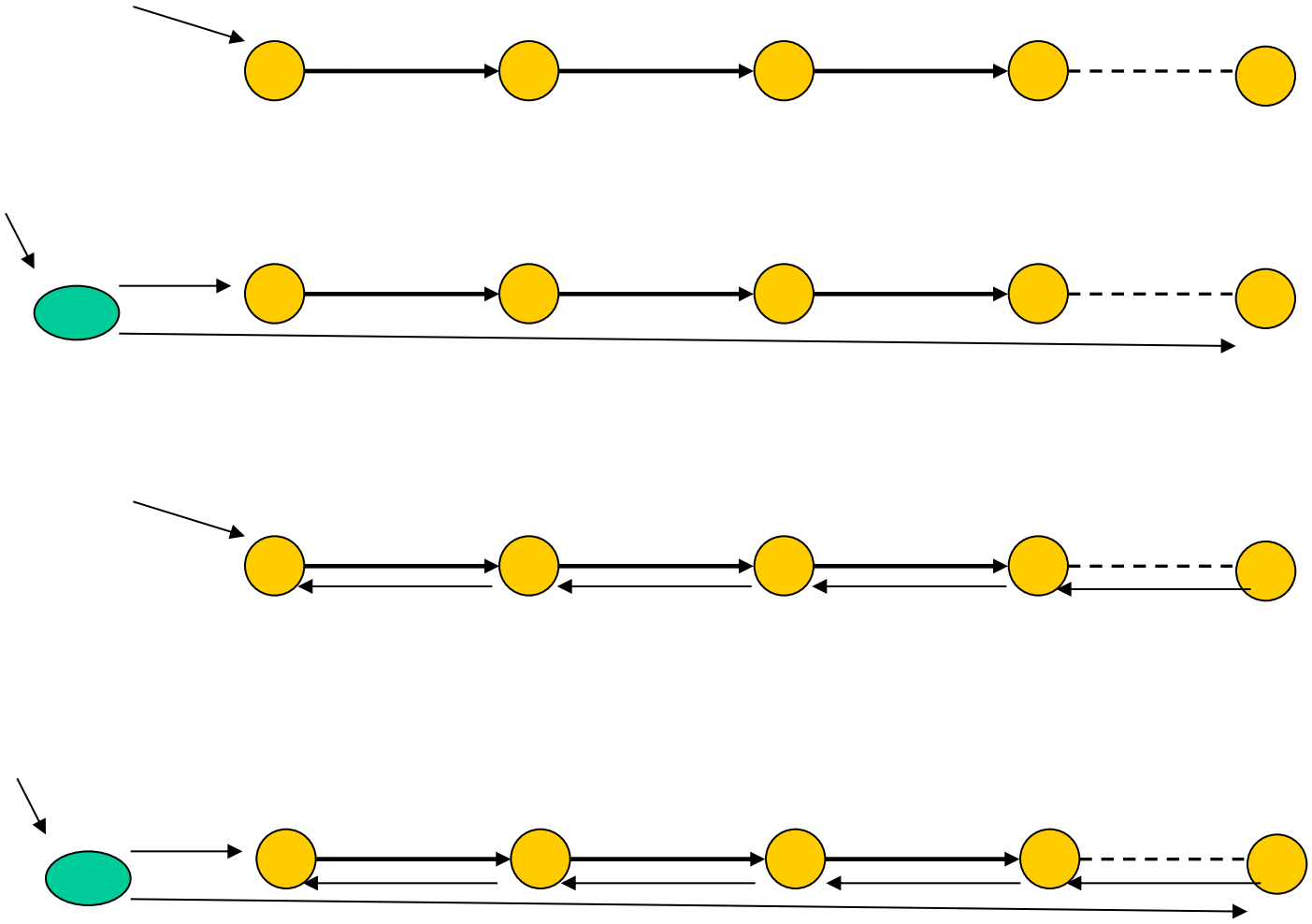
```
void elimUlt(Lista & l){
```

- **Iterativo**

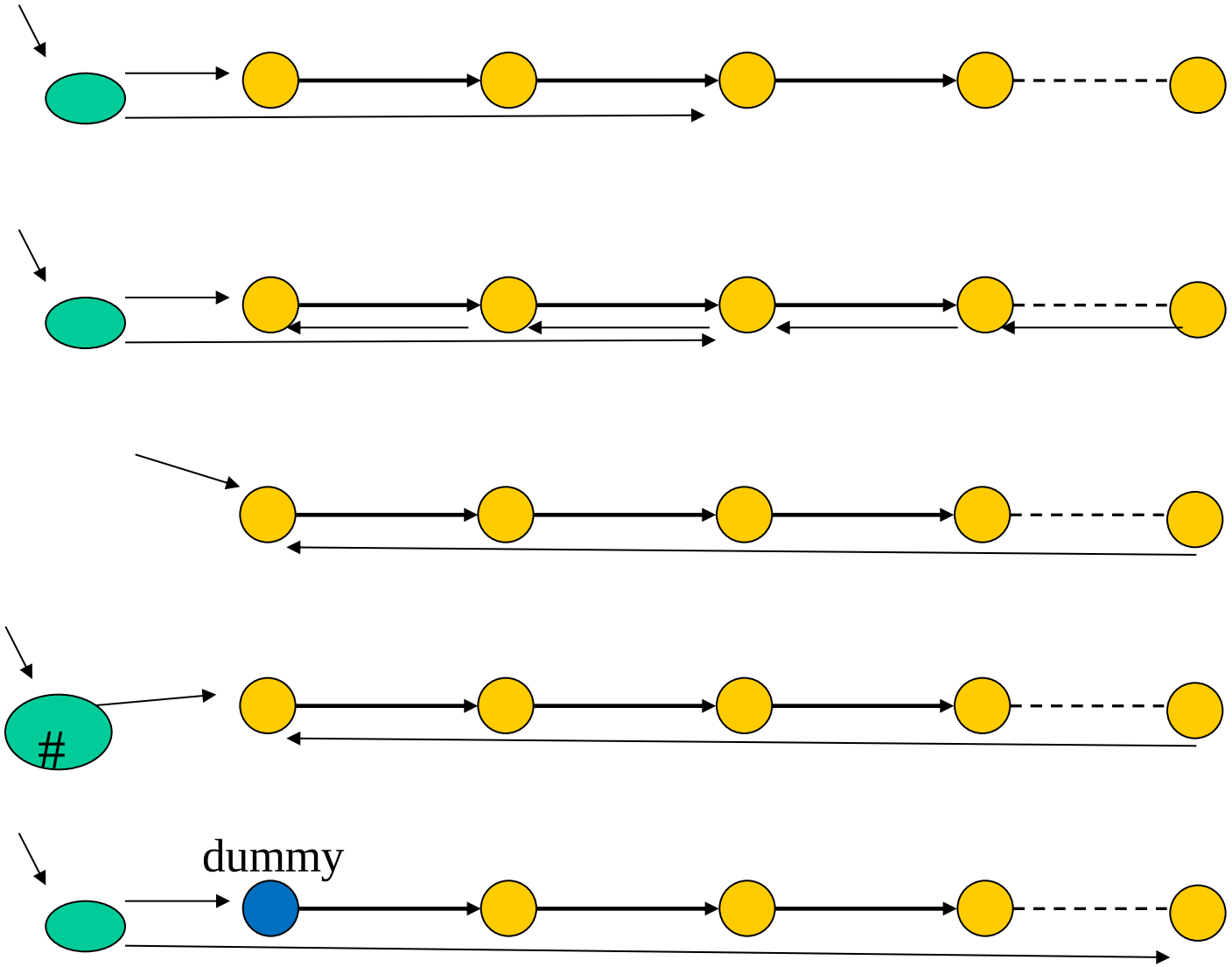
- **Recursivo**

Soluciones y comparación en el pizarrón

Algunas variantes de Listas



Algunas variantes de Listas



Lista doblemente encadenada con punteros al inicio y al final

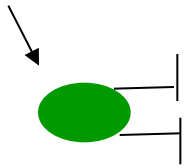
```
struct nodoDoble {
        T dato;
        nodoDoble* sig;
        nodoDoble* ant;
}

struct cabezal {
        nodoDoble* inicio;
        nodoDoble* final;
}
```

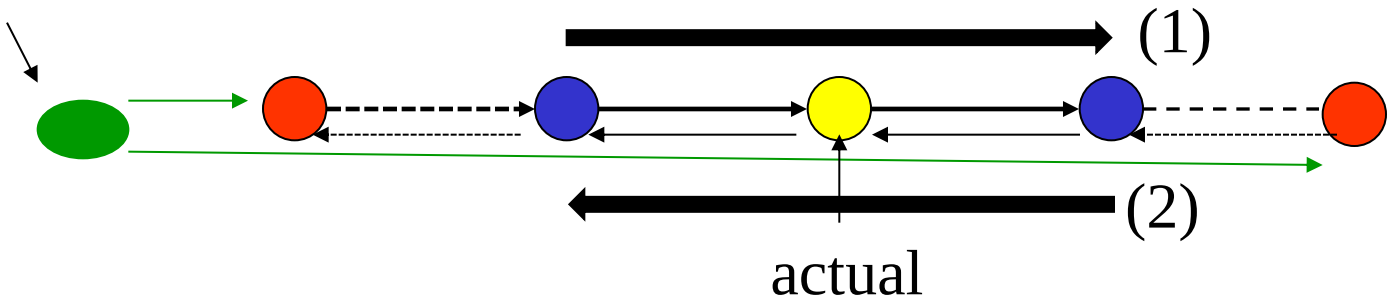
Lista doblemente encadenada con punteros al inicio y al final

struct nodoDoble { T dato; nodoDoble* sig; nodoDoble* ant; }

struct cabezal {nodoDoble* inicio; nodoDoble* final; }



Lista vacía: inicio y final en NULL



actual->ant->sig = actual->sig; // (1)
actual->sig->ant = actual->ant; // (2)

Insertar al final

```
void insFinal(cabezal* & l, T e)
Donde:
        struct nodoDoble {
            T dato;
            nodoDoble* sig;
            nodoDoble* ant;
        }
        struct cabezal {
            nodoDoble* inicio;
            nodoDoble* final;
        }
Solución en el pizarrón
```

Insertar al final

```
void insFinal(cabezal* & l, T e)
```

Donde:

```
struct nodoDoble {
    T dato;
    nodoDoble* sig;
    nodoDoble* ant;
}

struct cabezal {
    nodoDoble* inicio;
    nodoDoble* final;
}
```

Algo de la Tarea 2

```
#include "producto.h“
typedef struct rep_carritoProductos  *TCarritoProductos;

// Devuelve un carrito de productos vacío.
TCarritoProductos crearCarritoProductosVacio();

/* Inserta el producto en el carrito, ordenado de menor a mayor
   por ID producto. */
// PRE: no existe producto con el mismo id en la carrito
void insertarProductoCarritoProductos(TCarritoProductos
   &carritoProductos, TProducto producto);

…
```

Algo de la Tarea 2

```
struct rep_carritoProductos{
    TProducto producto;
    TCarritoProductos sig;
};
typedef struct rep_carritoProductos  *TCarritoProductos;

// Devuelve un carrito de productos vacío.
TCarritoProductos crearCarritoProductosVacio();

/* Inserta el producto en el carrito, ordenado de menor a mayor
    por ID producto. */
// PRE: no existe producto con el mismo id en el carrito
void insertarProductoCarritoProductos(TCarritoProductos
    &carritoProductos, TProducto producto);
```