Algoritmos Aleatorizados

Skip List

Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Idea Inicial

- La búsqueda en un ABB es eficiente, mientras que la inserción puede provocar desbalanceo del árbol y dar como peor caso búsqueda secuencial. Por otro lado, la inserción en una lista es fácil, mientras que la búsqueda será siempre secuencial.
- La idea consiste en construir una especie de supercarretera con varios carriles, desde alta velocidad hasta mínima velocidad.
- Yendo en los carriles de alta es posible recorrer mas nodos, pero también es fácil que nos pasemos. Ir por los carriles de baja significa ir mas lento, pero llegar seguro.

Qué es?

- Una Skip List es una Estructura de Datos, basada en listas enlazadas paralelas. Fueron "descubiertas" por William Pugh en 1990.
- Su mayor cualidad es que tiene una eficiencia comparable a la de un árbol binario (orden O (Log n) para la mayoría de las operaciones).
- Básicamente, una skip list es un aumento de una lista enlazada (ordenada) con enlaces adicionales, añadidos de manera aleatoria con una distribución Geométrica/Negativa Binomial (?).
- Las operaciones de inserción, búsqueda y borrado son ejecutadas en un tiempo logarítmico aleatorio.

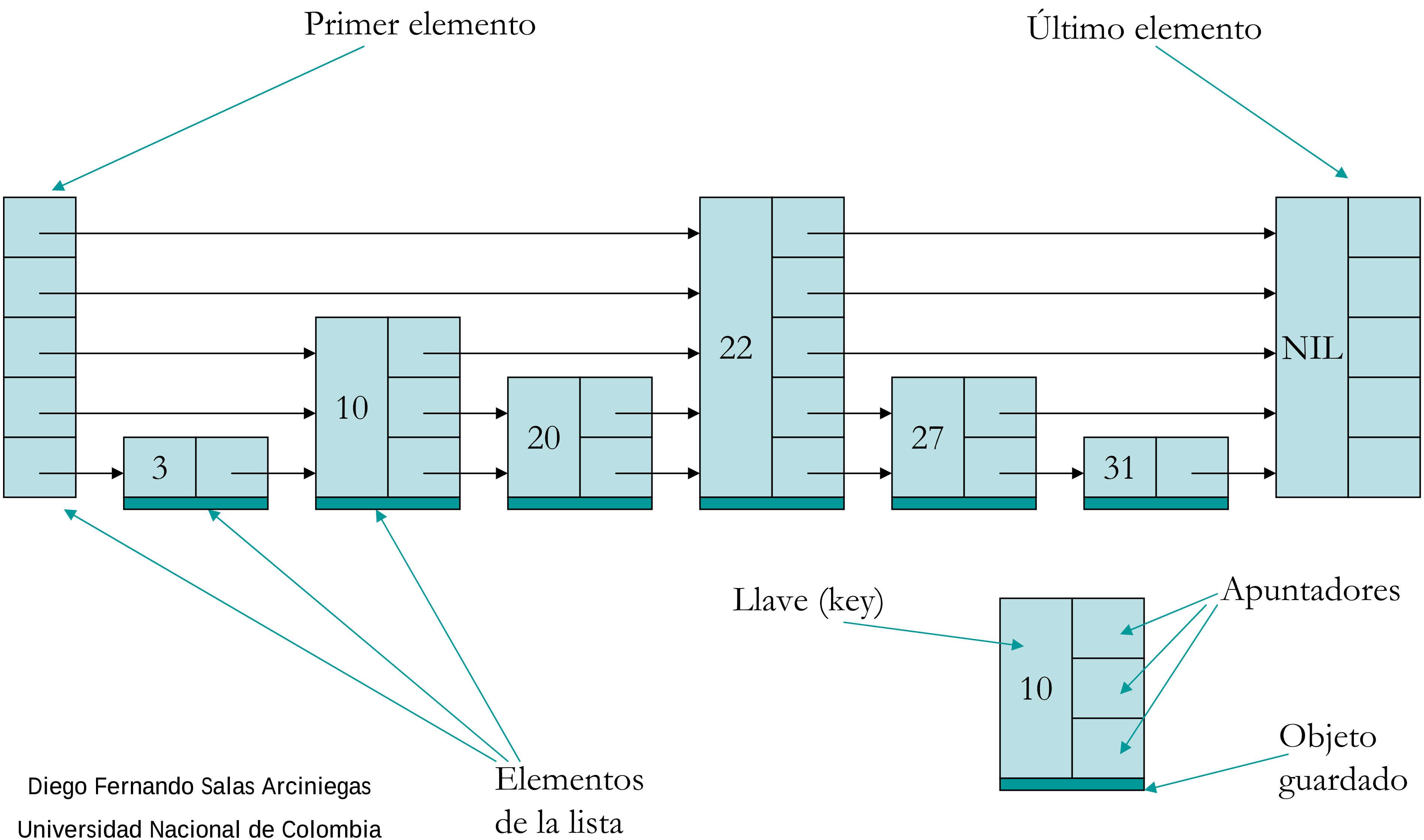
Cómo es?

- Una Skip List es construida por “capas”. La capa más baja es una lista enlazada ordenada ordinaria. A partir de esta capa, se construyen capas superiores que trabajan como “caminos expresos” para las listas más bajas.

- Las capas superiores se construyen de la siguiente manera: cada elemento en la capa i aparece en la capa $i+1$ con una probabilidad definida p (normalmente $1/2$). Así, en promedio, cada elemento aparece en $1/(1-p)$ listas.

- Los elementos almacenados en una Skip List deben tener una llave mediante la cual se puede realizar la búsqueda de manera eficiente (de la misma manera que se realiza en un árbol binario ordenado).

Cómo es?


Primer elemento
Último elemento
22
NIL
10
20
27
3
31
Llave (key)
Apuntadores
10
Objeto guardado
Elementos de la lista

Ventajas

- Su implementación es más sencilla que la de un ABB.
- Es necesario menos espacio de almacenamiento para su manejo (en operaciones intermedias) que en un ABB.
- En las operaciones de inserción y eliminación no es necesario reconstruir la estructura.
- Soporta fácilmente la mezcla, unión y diferencia de listas.

Desventajas

- No siempre la operación de búsqueda (e inserción) es más rápida (tiempo máquina) que en los ABB (por el mismo hecho de ser una estructura aleatoria).

- En muchos casos la dependencia de algoritmos con carácter aleatorio puede dificultar el seguimiento de los movimientos en la estructura y depuración de operaciones.

Búsqueda

- La búsqueda de elementos (como ya se dijo), tiene un tiempo que pertenece a O(log(n)) en muchos de los casos. No se puede decir que en todos los casos por el mismo carácter aleatorio de la estructura.

- Para buscar se comienza con el elemento “cabeza” (el cual está en todas las listas) y con la lista más superior (que es la más rápida). Vamos recorriendo cada lista enlazada hasta que encontramos el último elemento que es menor o igual que el elemento que buscamos.

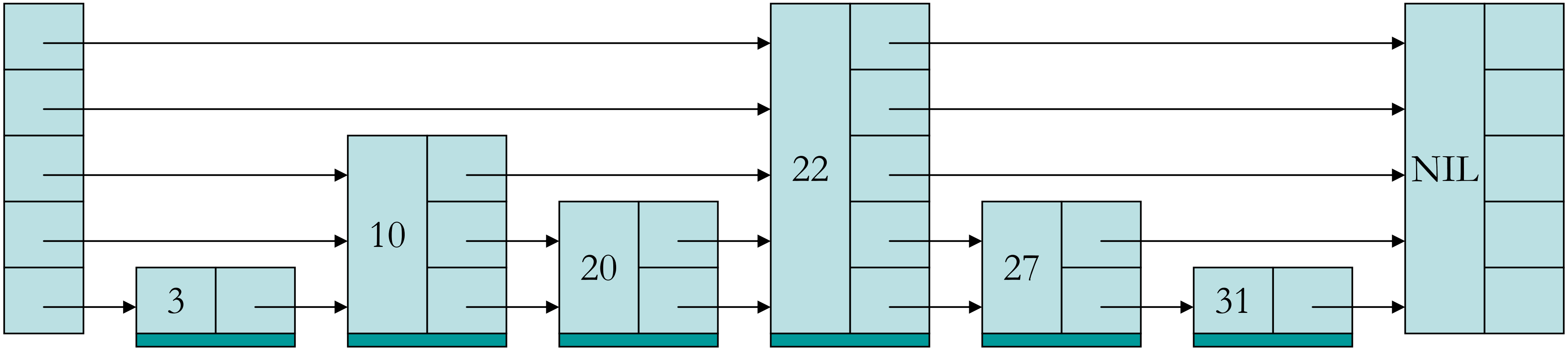
- El número esperado de pasos es $1/p$.

- El costo total de la búsqueda es $O(\text{Log}_{(1/p)}(n/p))$, el cual es $O(\log n)$ cuando p es una constante (lo cual ya vimos que se cumple).

Búsqueda

Ejemplo: queremos buscar el objeto cuyo key sea 20

Partimos con el primer puntero del primer elemento

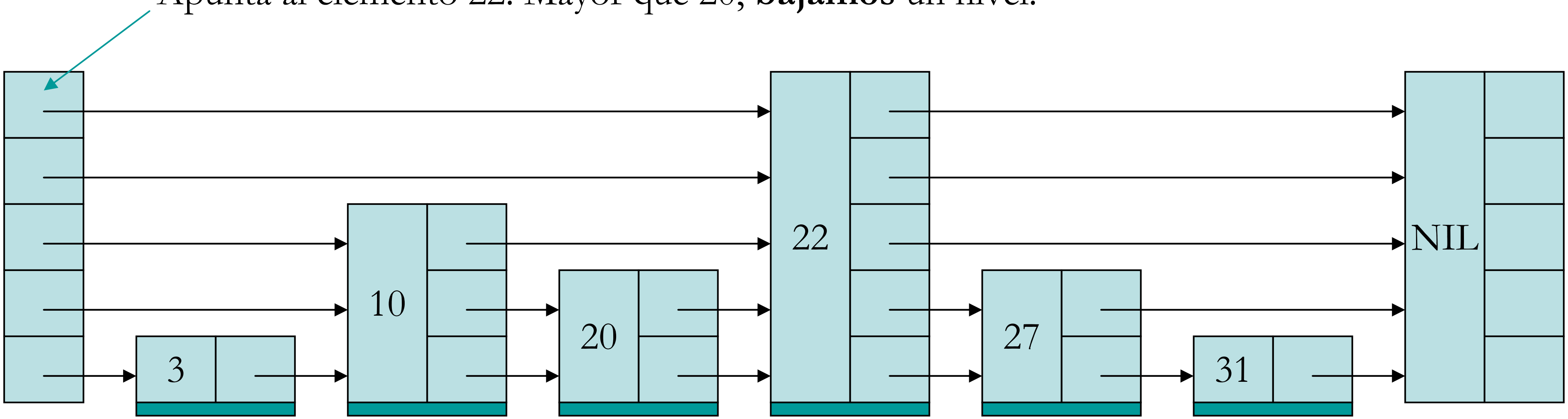


Búsqueda

Ejemplo: queremos buscar el objeto cuyo key sea 20

Partimos con el primer puntero del primer elemento

Apunta al elemento 22. Mayor que 20, **bajamos** un nivel.



Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

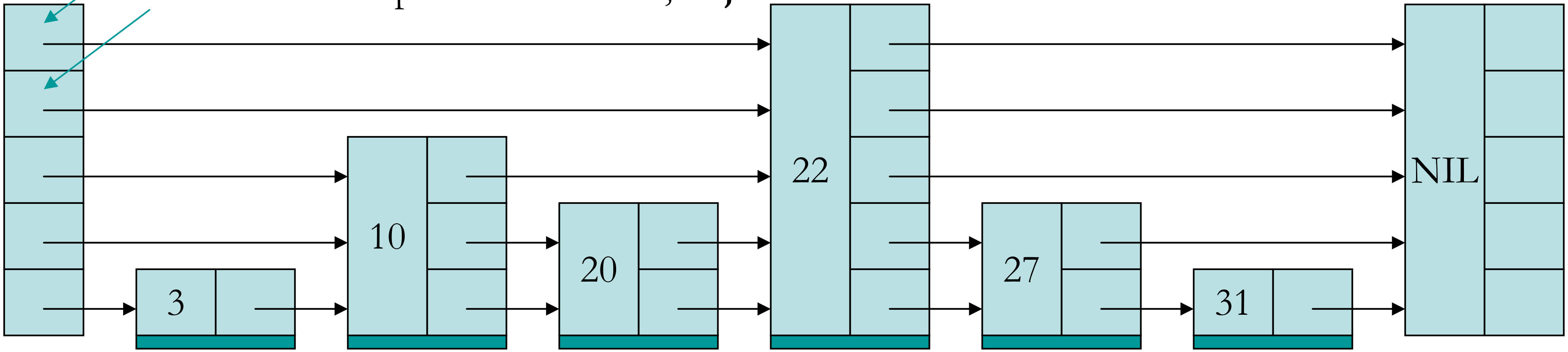
Búsqueda

Ejemplo: queremos buscar el objeto cuyo key sea 20

Partimos con el primer puntero del primer elemento

Apunta al elemento 22. Mayor que 20, **bajamos** un nivel.

Nuevamente el puntero indica 22, **bajamos** de nivel.



Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Búsqueda

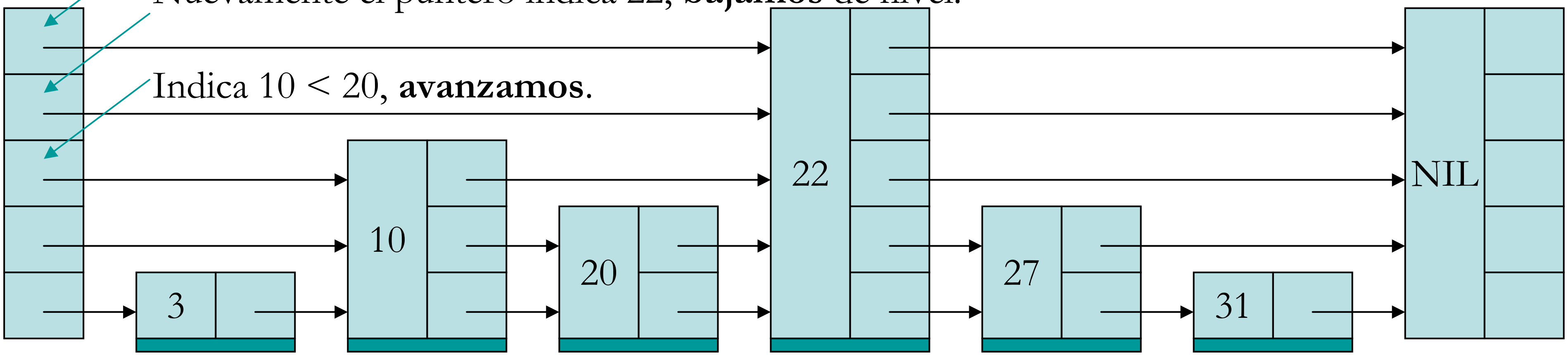
Ejemplo: queremos buscar el objeto cuyo key sea 20

Partimos con el primer puntero del primer elemento

Apunta al elemento 22. Mayor que 20, **bajamos** un nivel.

Nuevamente el puntero indica 22, **bajamos** de nivel.

Indica 10 < 20, **avanzamos.**



Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Búsqueda

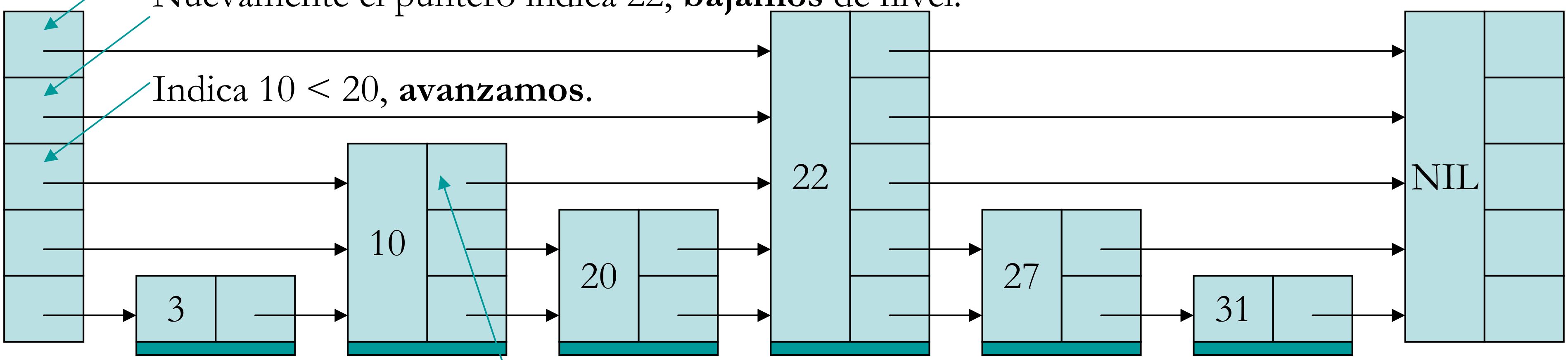
Ejemplo: queremos buscar el objeto cuyo key sea 20

Partimos con el primer puntero del primer elemento

Apunta al elemento 22. Mayor que 20, **bajamos** un nivel.

Nuevamente el puntero indica 22, **bajamos** de nivel.

Indica 10 < 20, **avanzamos.**



Indica 22 > 20, **bajamos.**

Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Búsqueda

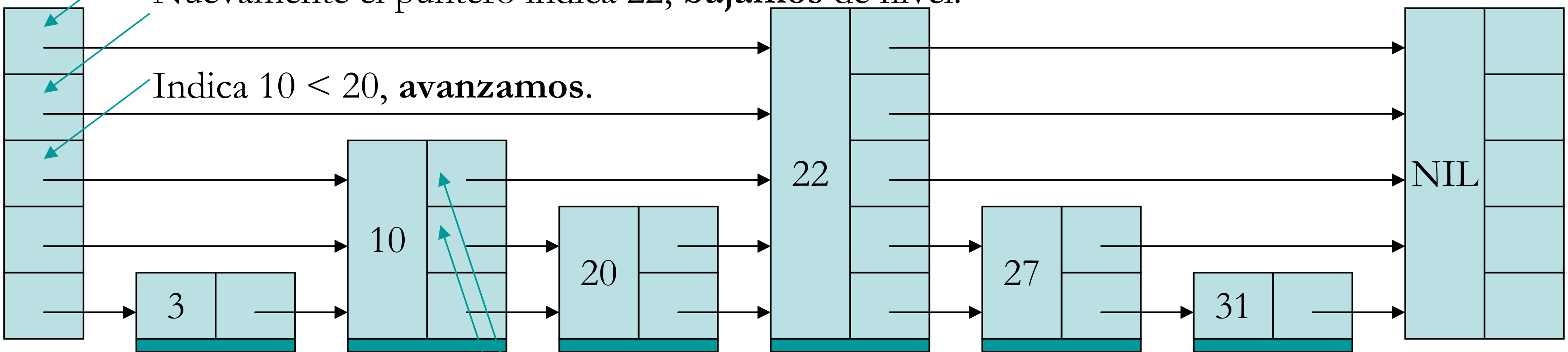
Ejemplo: queremos buscar el objeto cuyo key sea 20

Partimos con el primer puntero del primer elemento

Apunta al elemento 22. Mayor que 20, **bajamos** un nivel.

Nuevamente el puntero indica 22, **bajamos** de nivel.

Indica 10 < 20, **avanzamos.**



Indica 22 > 20, **bajamos.**

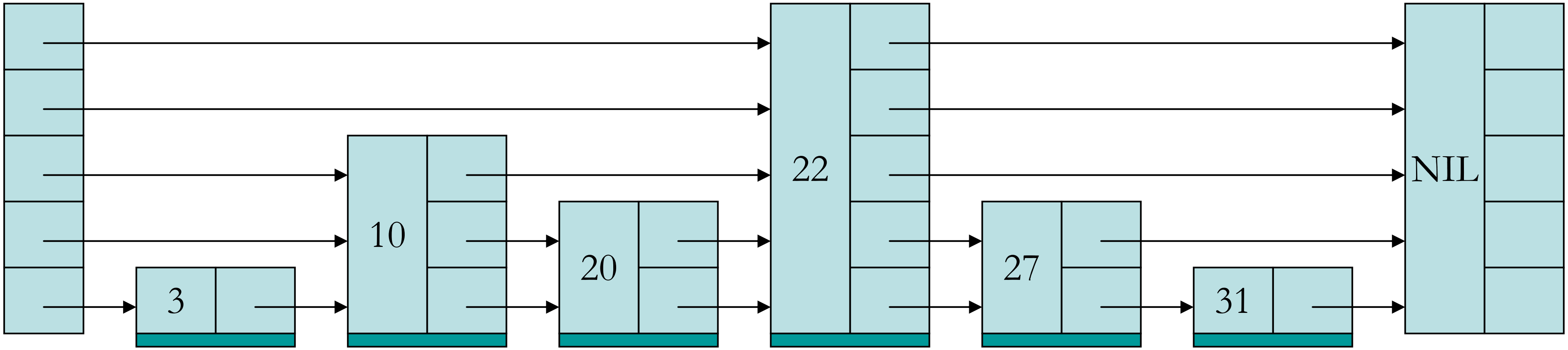
Indica 20, hemos encontrado el elemento buscado, **retornamos.**

Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Búsqueda

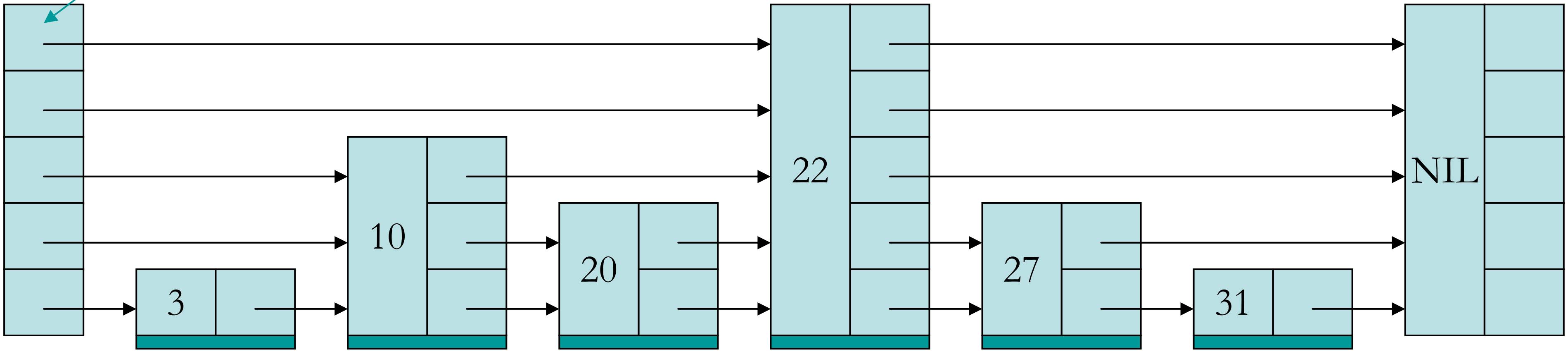
Ejemplo: queremos buscar el objeto cuyo key sea 30



Búsqueda

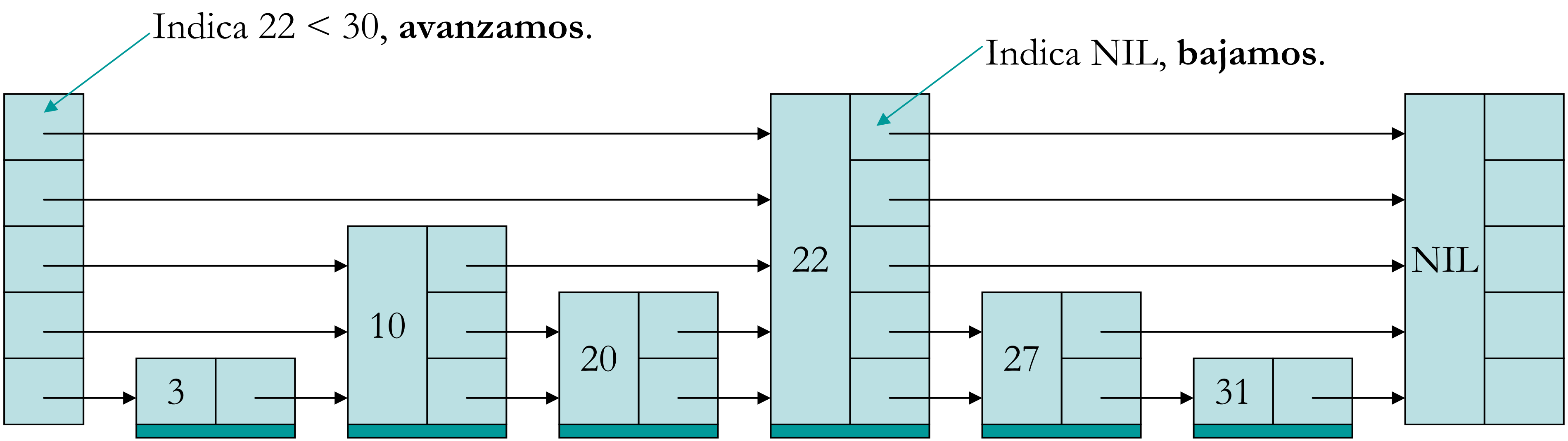
Ejemplo: queremos buscar el objeto cuyo key sea 30

Indica 22 < 30, **avanzamos**.



Búsqueda

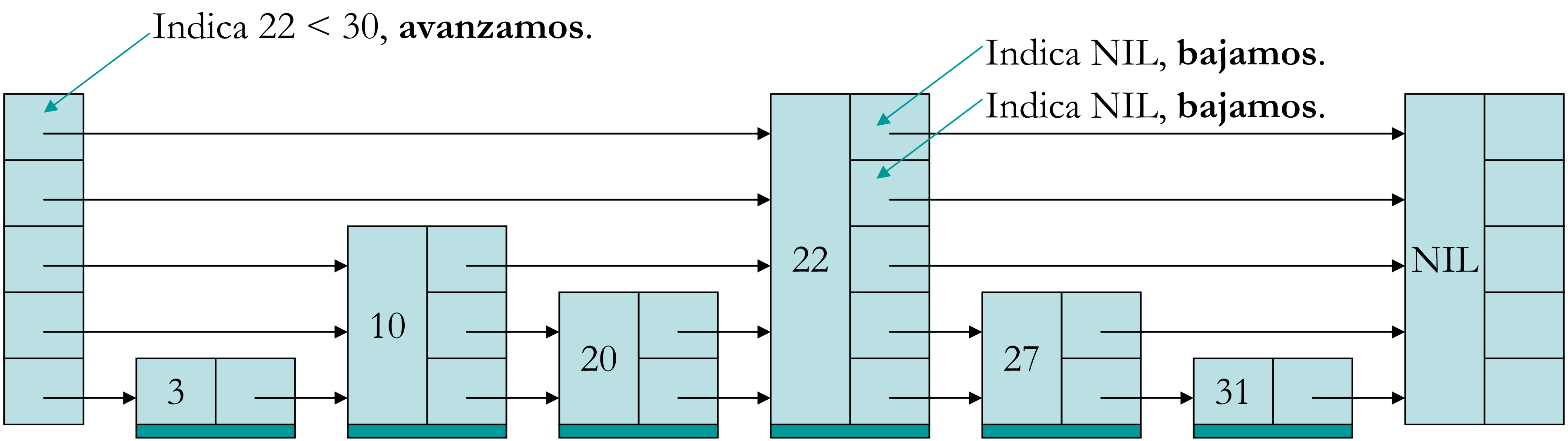
Ejemplo: queremos buscar el objeto cuyo key sea 30



Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

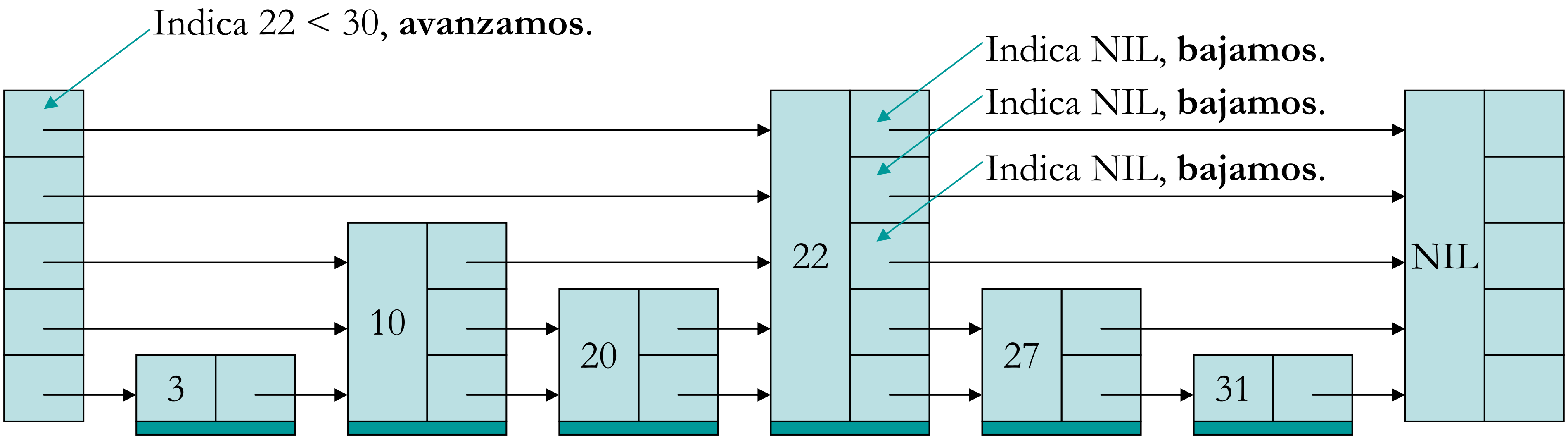
Búsqueda

Ejemplo: queremos buscar el objeto cuyo key sea 30



Búsqueda

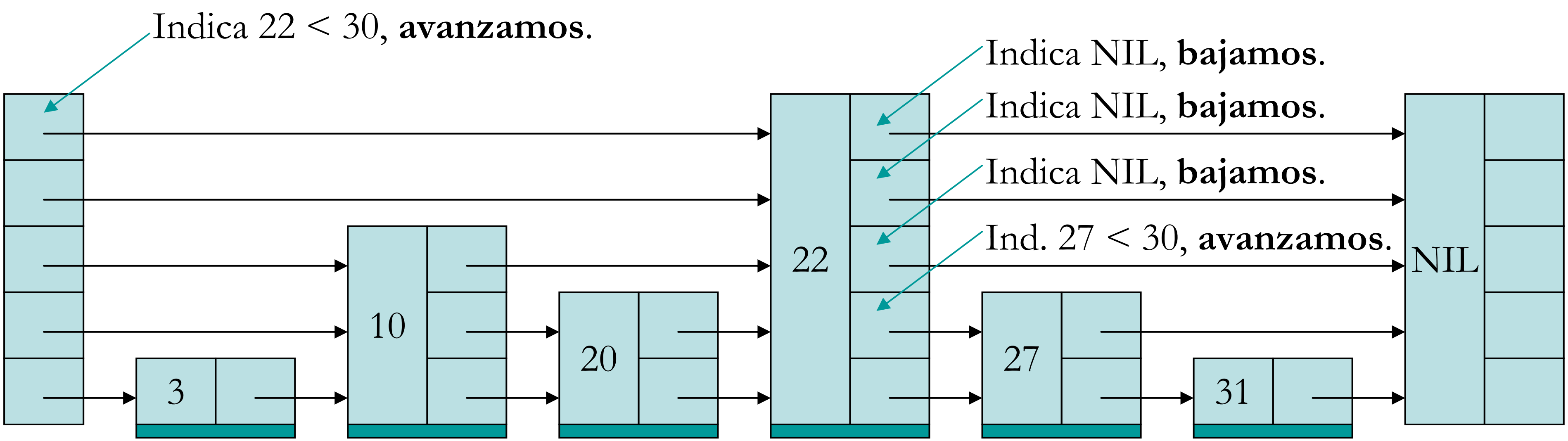
Ejemplo: queremos buscar el objeto cuyo key sea 30



Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Búsqueda

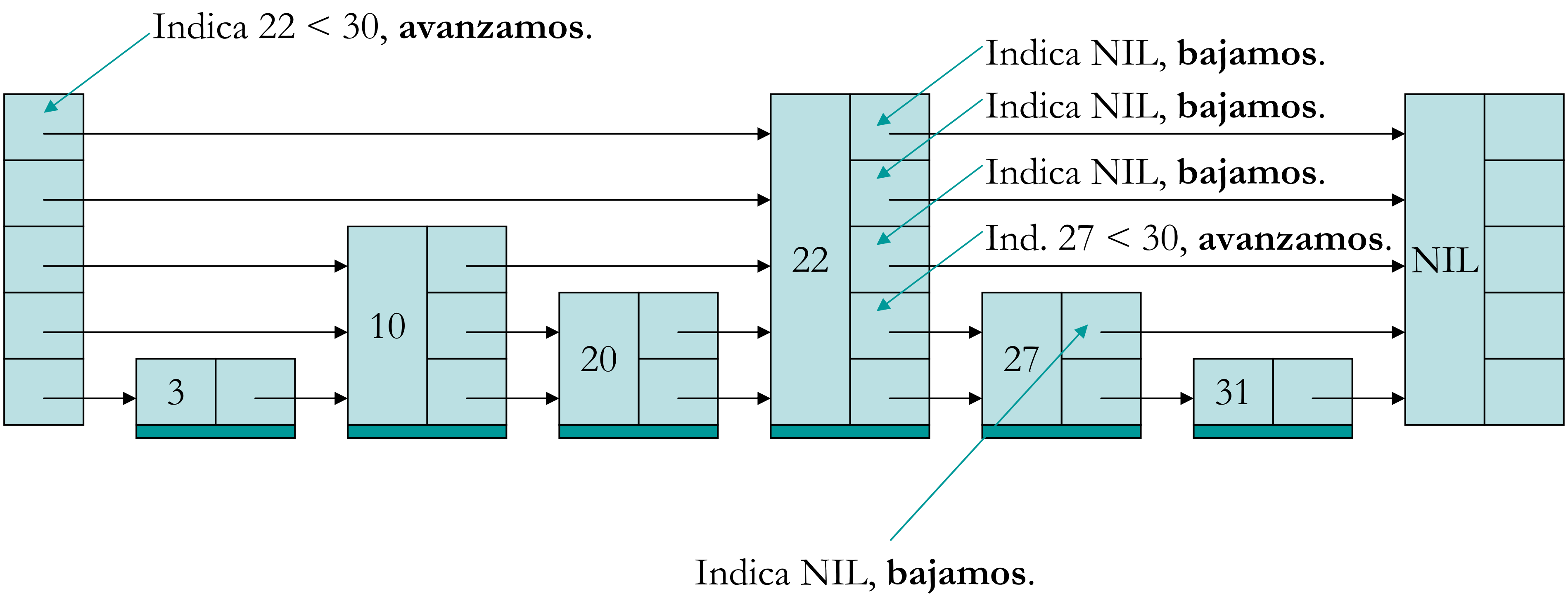
Ejemplo: queremos buscar el objeto cuyo key sea 30



Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

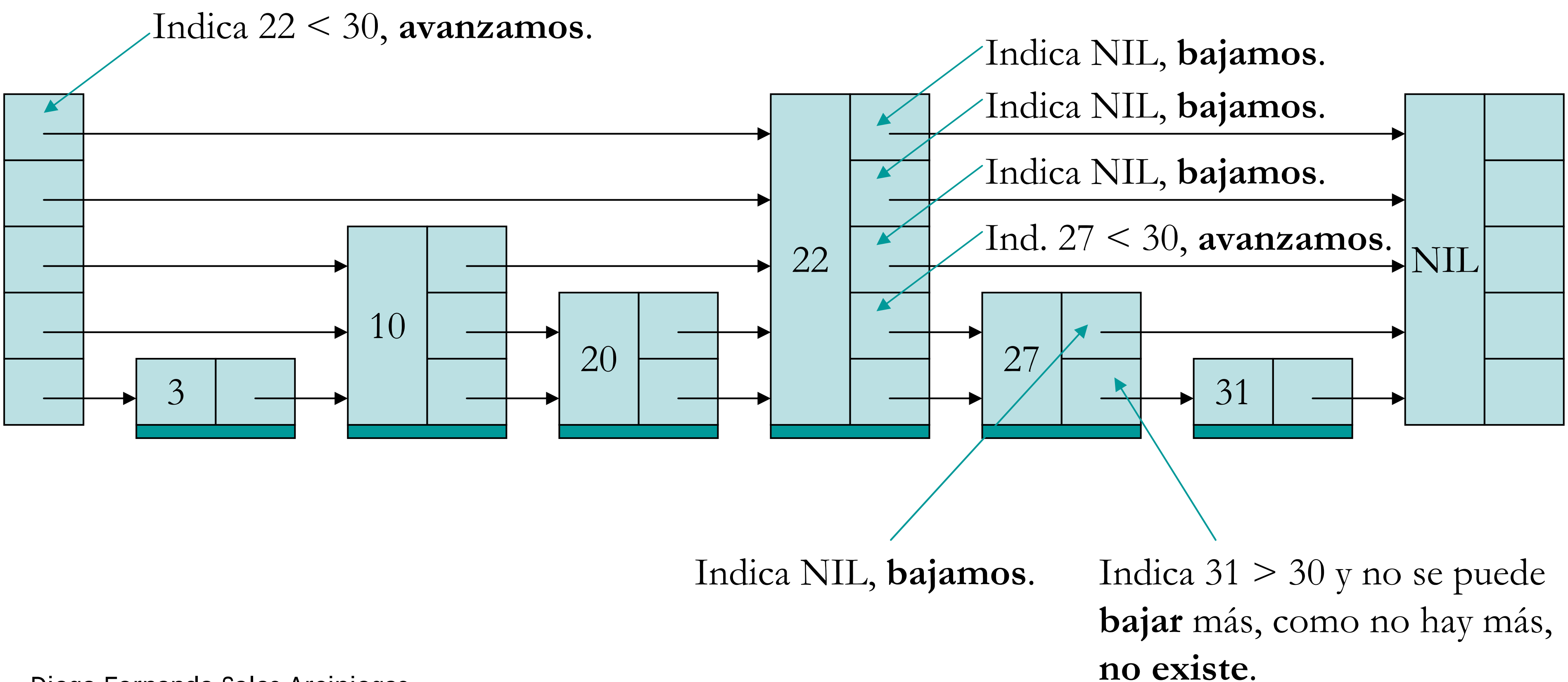
Búsqueda

Ejemplo: queremos buscar el objeto cuyo key sea 30



Búsqueda

Ejemplo: queremos buscar el objeto cuyo key sea 30



Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Búsqueda

```
Actual = Primer Nodo
Enlace = Máximo Numero de enlaces
repetir mientras Actual != Nodo Buscado o
        (Primer enlace referencia Nulo o Nodo Mayor)
  mientras Enlace sea Nulo
     Enlace = Enlace – 1
  fin mientras
  si Actual = Nodo Siguiente entonces
     Actual = Nodo Siguiente
  en caso contrario
     Enlace = Enlace - 1
  fin si
fin mientras
```

Mantenimiento

- El mantenimiento de la lista con saltos y en especial las operaciones de añadido y eliminación, tendrán una mayor complejidad cuando el numero de saltos establecidos sea mayor.

- La razón de ello radica en que el mayor número de saltos implicará un mayor número de sublistas en distintos niveles que es necesario mantener y por lo tanto un mayor número de enlaces a modificar.

- Tanto en la Inserción como en la eliminación, la primera parte de la operación es la búsqueda del elemento; tras comprobar su existencia proceder a la operación de modificación si existe o no existe (dado el caso).

Inserción

- Se crea un nodo cabecera con el máximo nivel de enlaces posible.

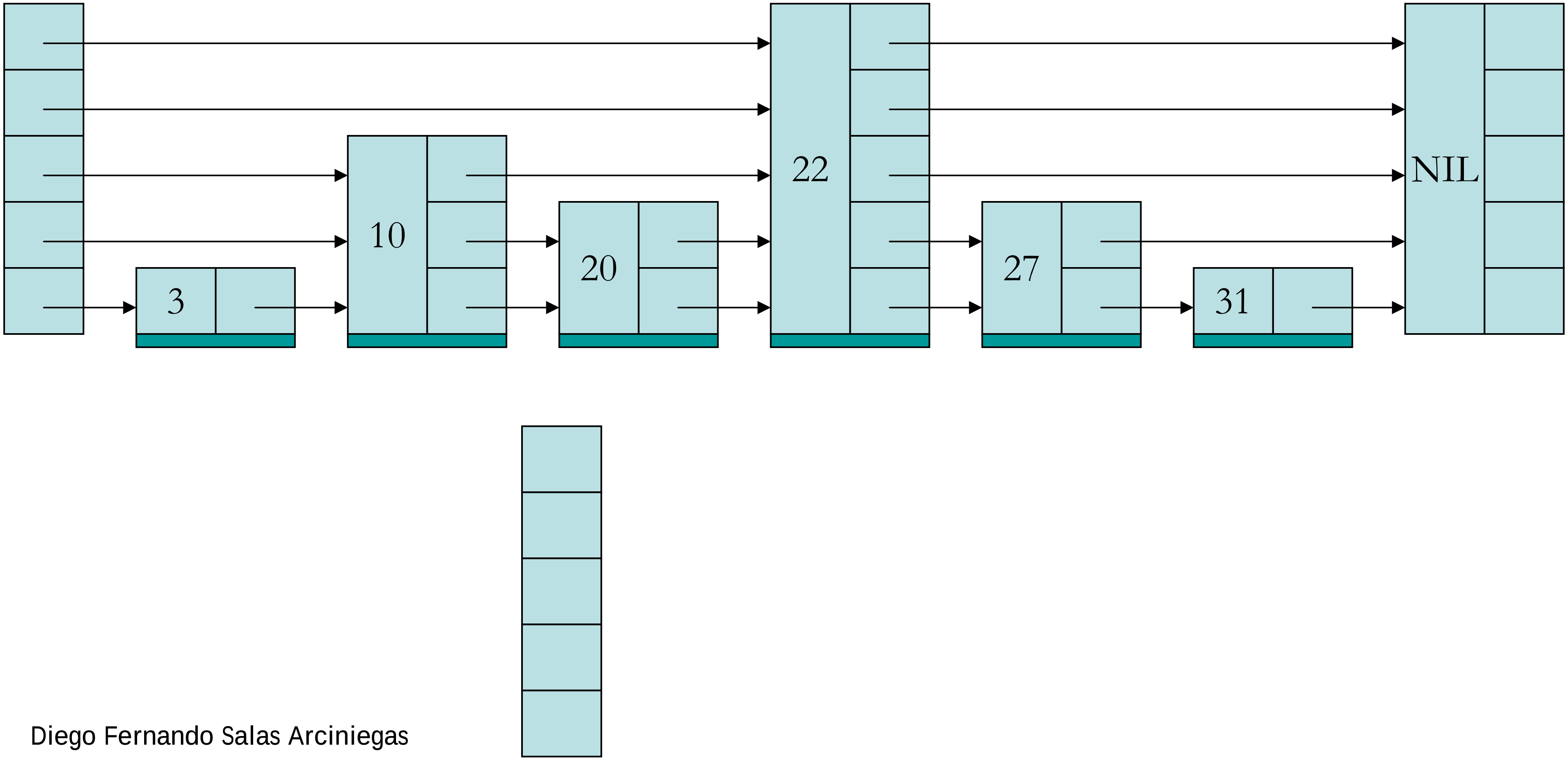
- Se realiza una búsqueda en la que se mantendrá el historial de los nodos recorridos (sus referencias), con el fin de poder modificar el nuevo camino tras la inserción.

- Se calcula el nivel del nuevo nodo (como ya vimos, empezando desde el nivel más bajo y con una probabilidad *p* de estar en un nivel superior) y se crea dicho nodo.

- A partir del historial de búsqueda se crearan los enlaces al nuevo nodo (insertando el mismo en las listas de cada nivel que el nuevo nodo tenga).

Eliminación

Ejemplo: queremos eliminar el objeto cuyo key sea 20



Arreglo de punteros 'anterior'

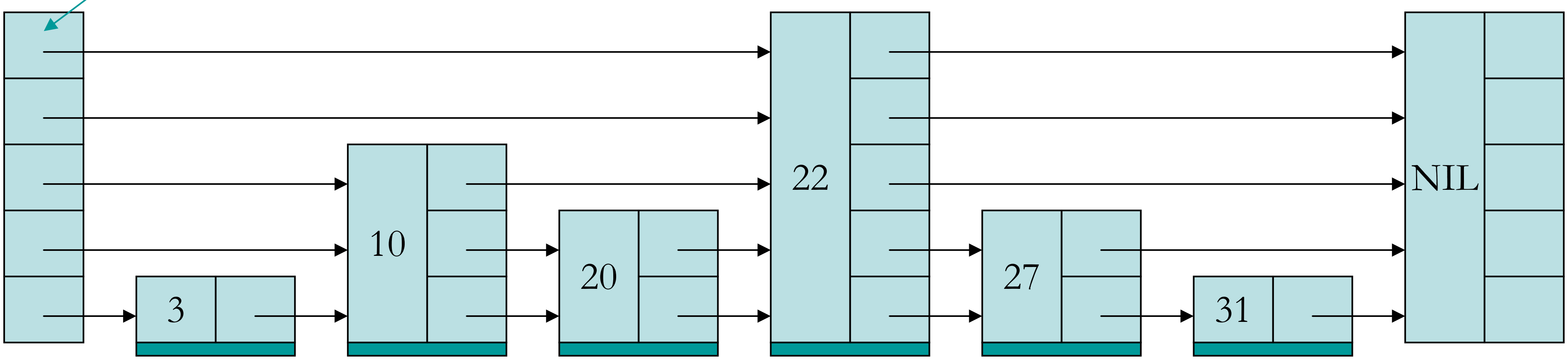
Diego Fernando Salas Arciniegas

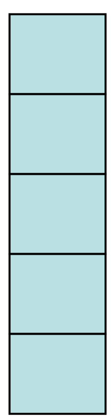
Universidad Nacional de Colombia

Eliminación

Ejemplo: queremos eliminar el objeto cuyo key sea 20

22 > 20, **bajamos**





Arreglo de punteros 'anterior'

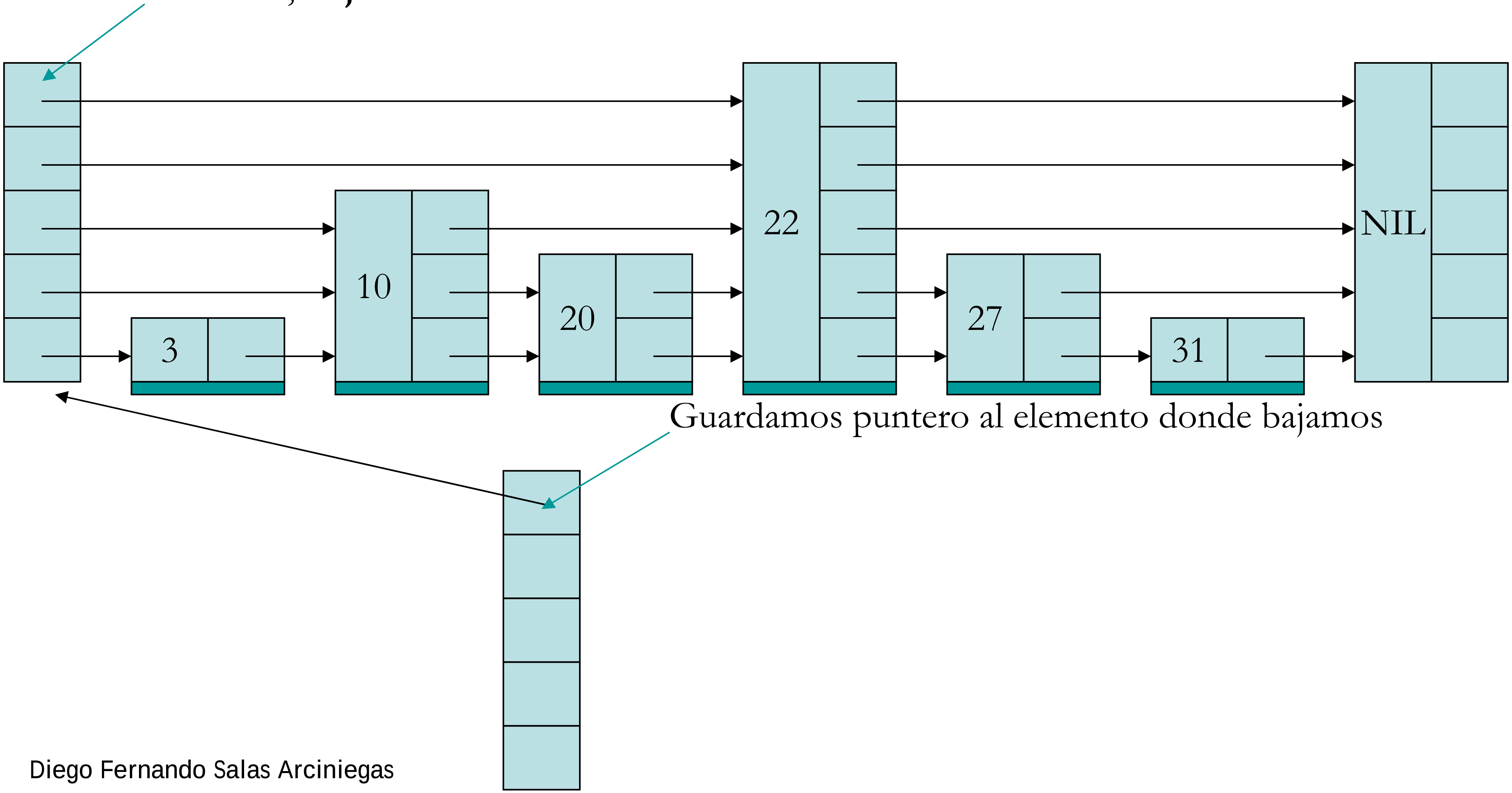
Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Eliminación

Ejemplo: queremos eliminar el objeto cuyo key sea 20

22 > 20, **bajamos**



Guardamos puntero al elemento donde bajamos

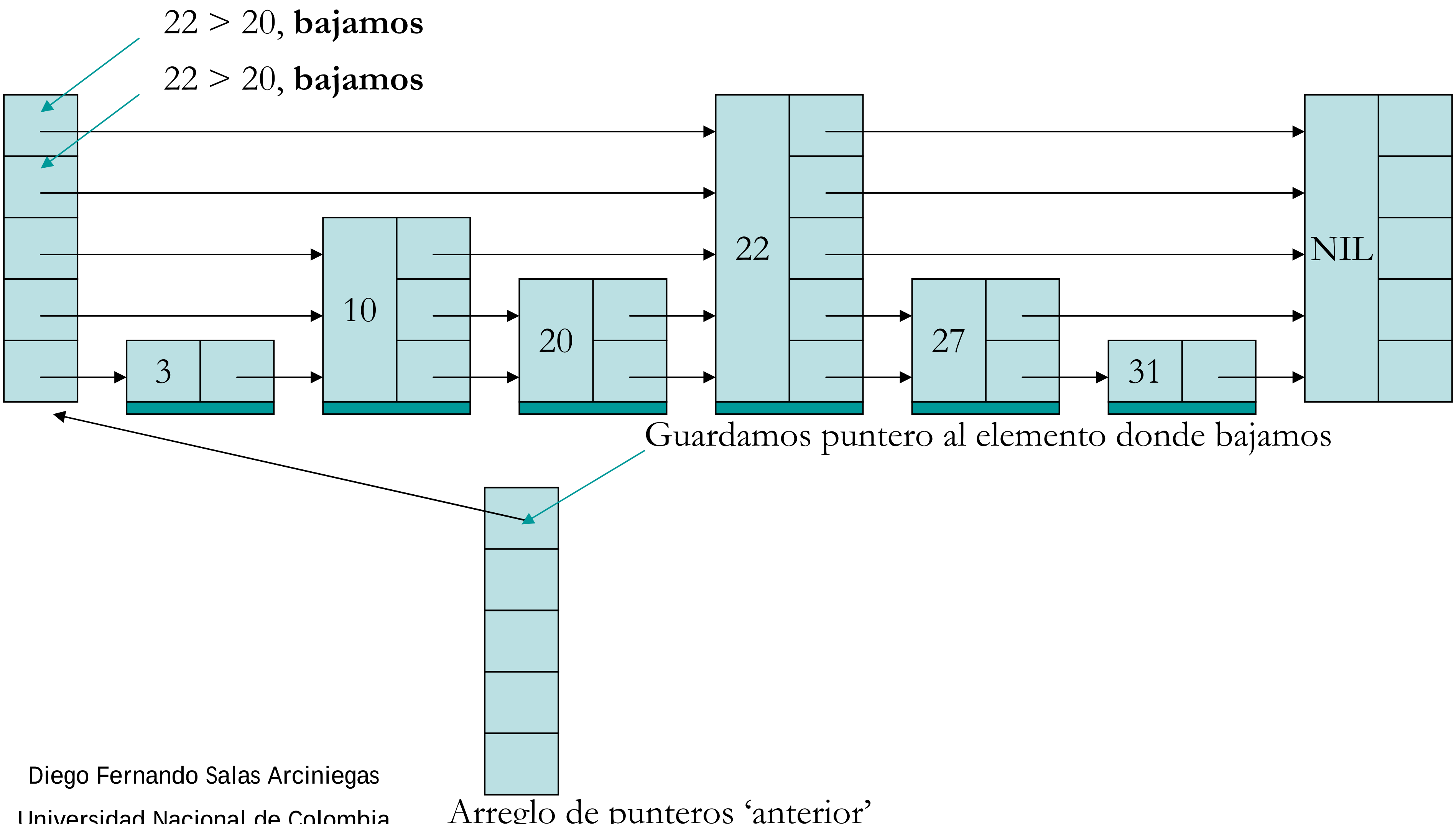
Arreglo de punteros 'anterior'

Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Eliminación

Ejemplo: queremos eliminar el objeto cuyo key sea 20

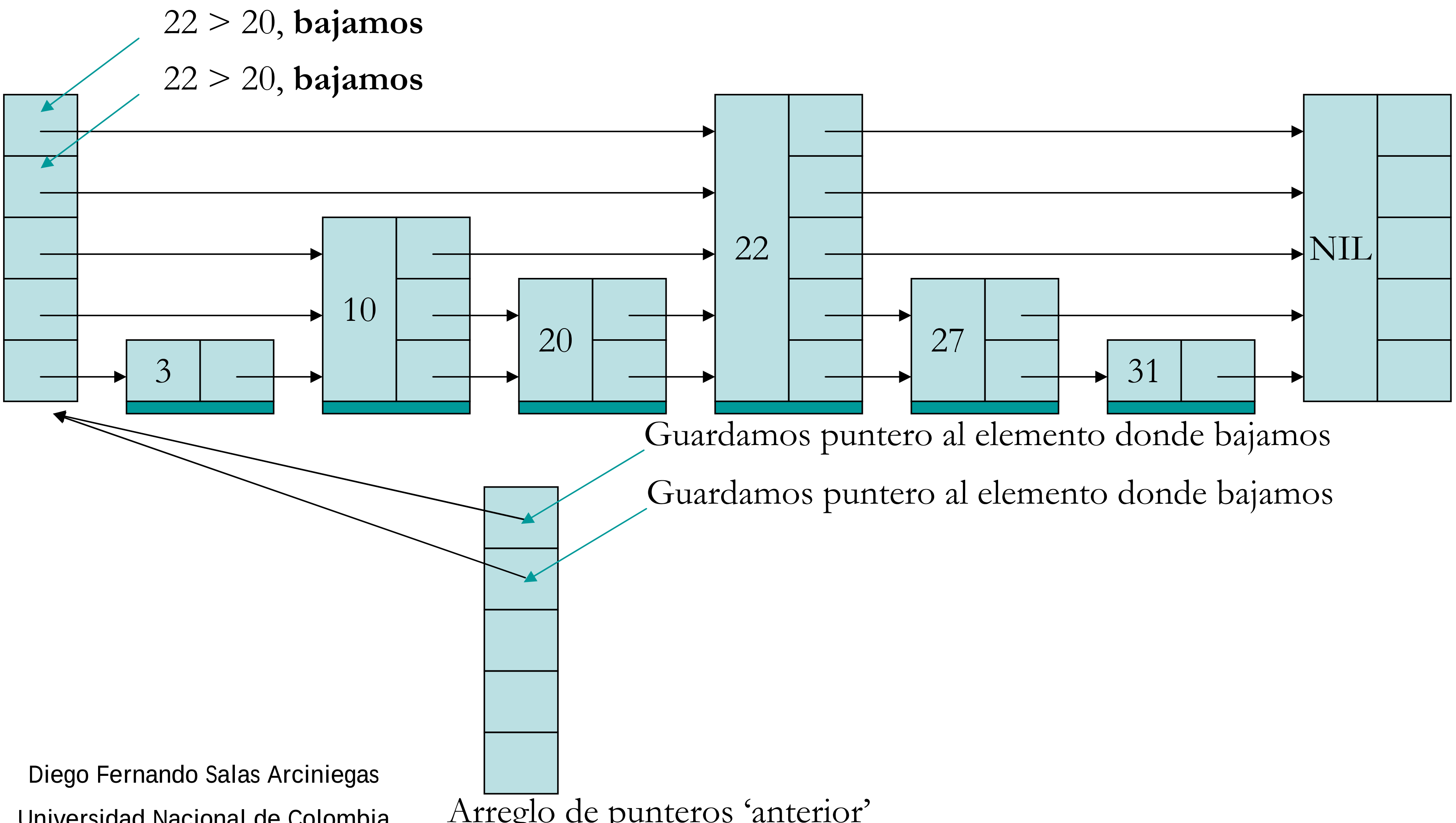


Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Eliminación

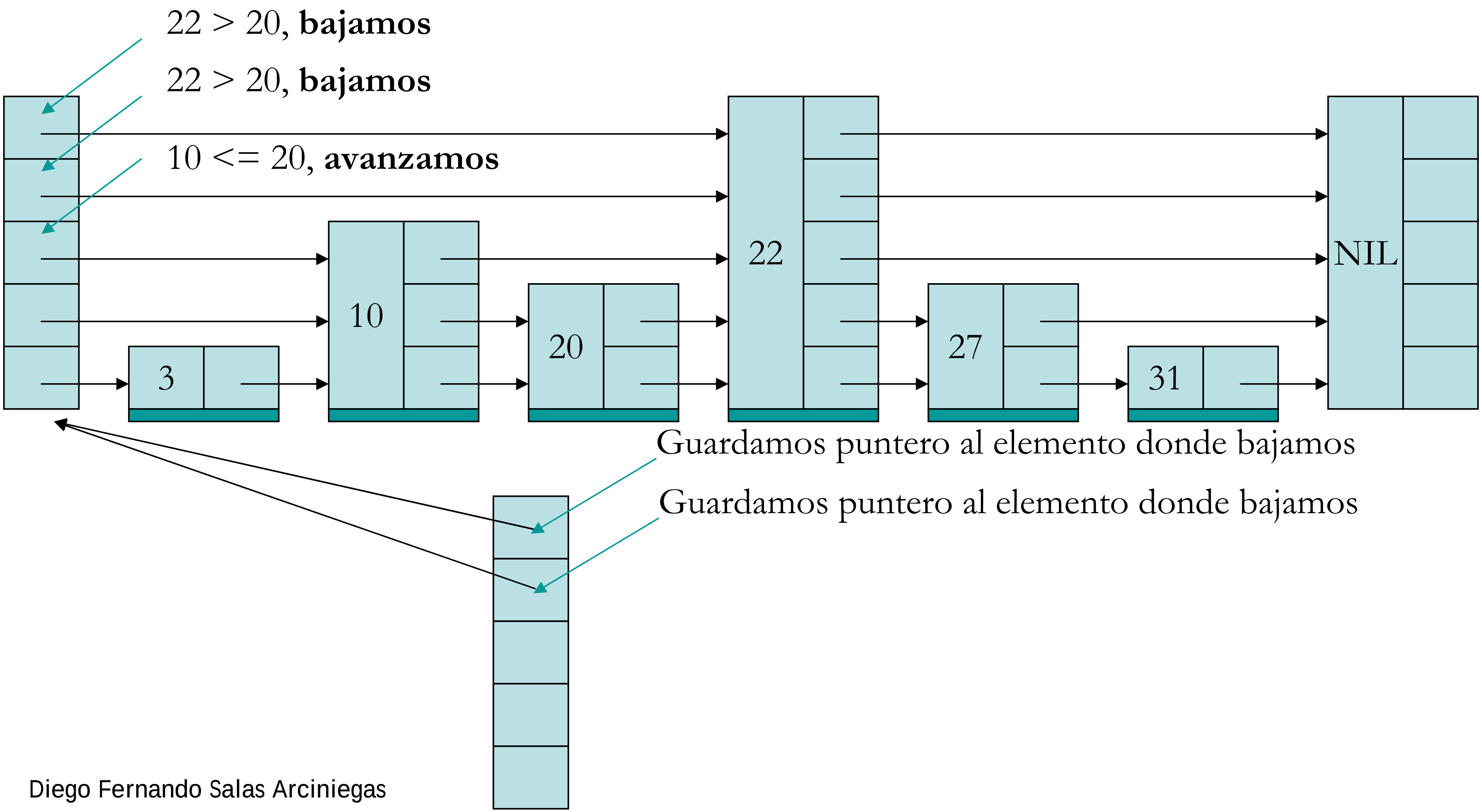
Ejemplo: queremos eliminar el objeto cuyo key sea 20



Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Eliminación

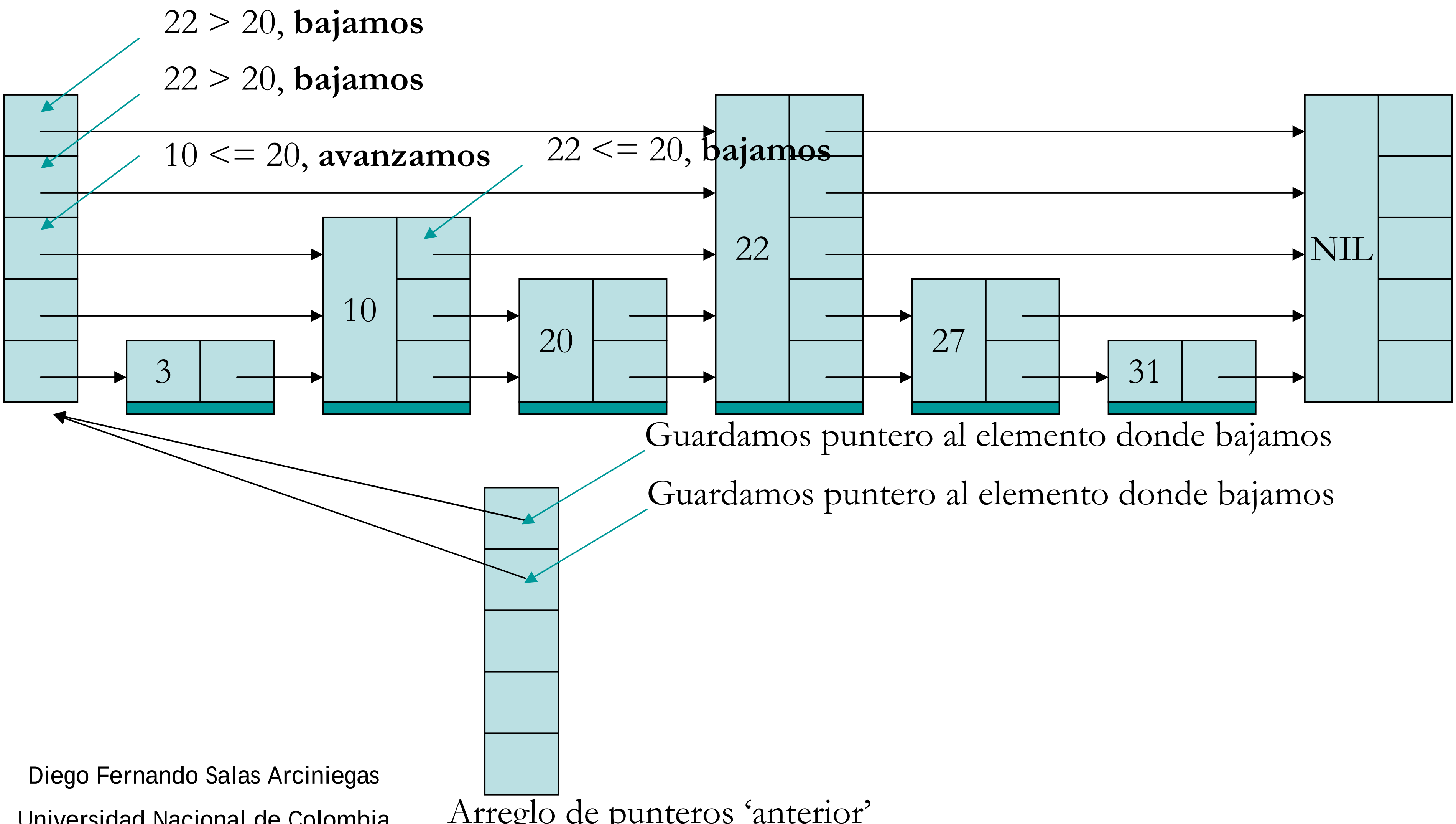
Ejemplo: queremos eliminar el objeto cuyo key sea 20



Arreglo de punteros ‘anterior’

Eliminación

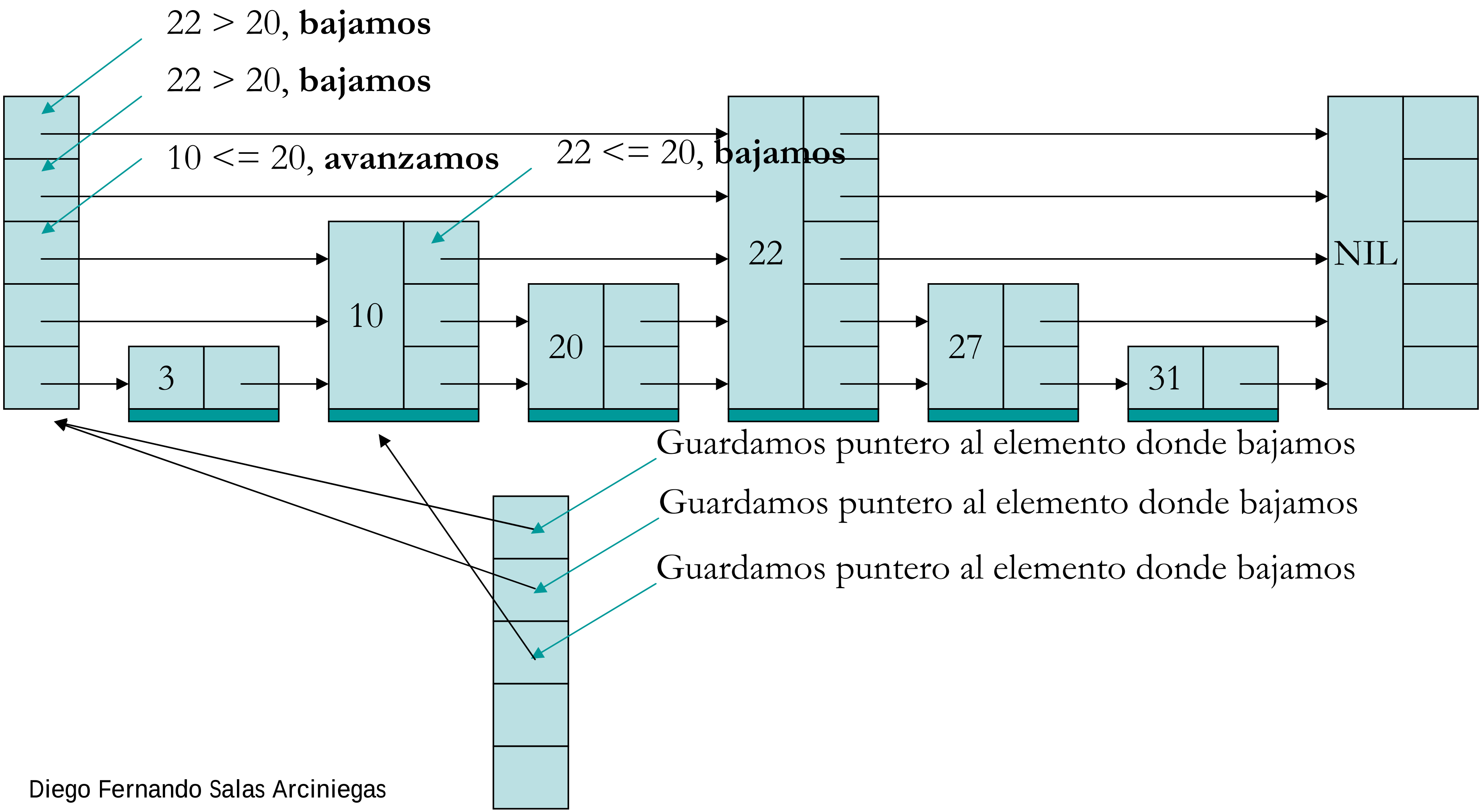
Ejemplo: queremos eliminar el objeto cuyo key sea 20



Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Eliminación

Ejemplo: queremos eliminar el objeto cuyo key sea 20

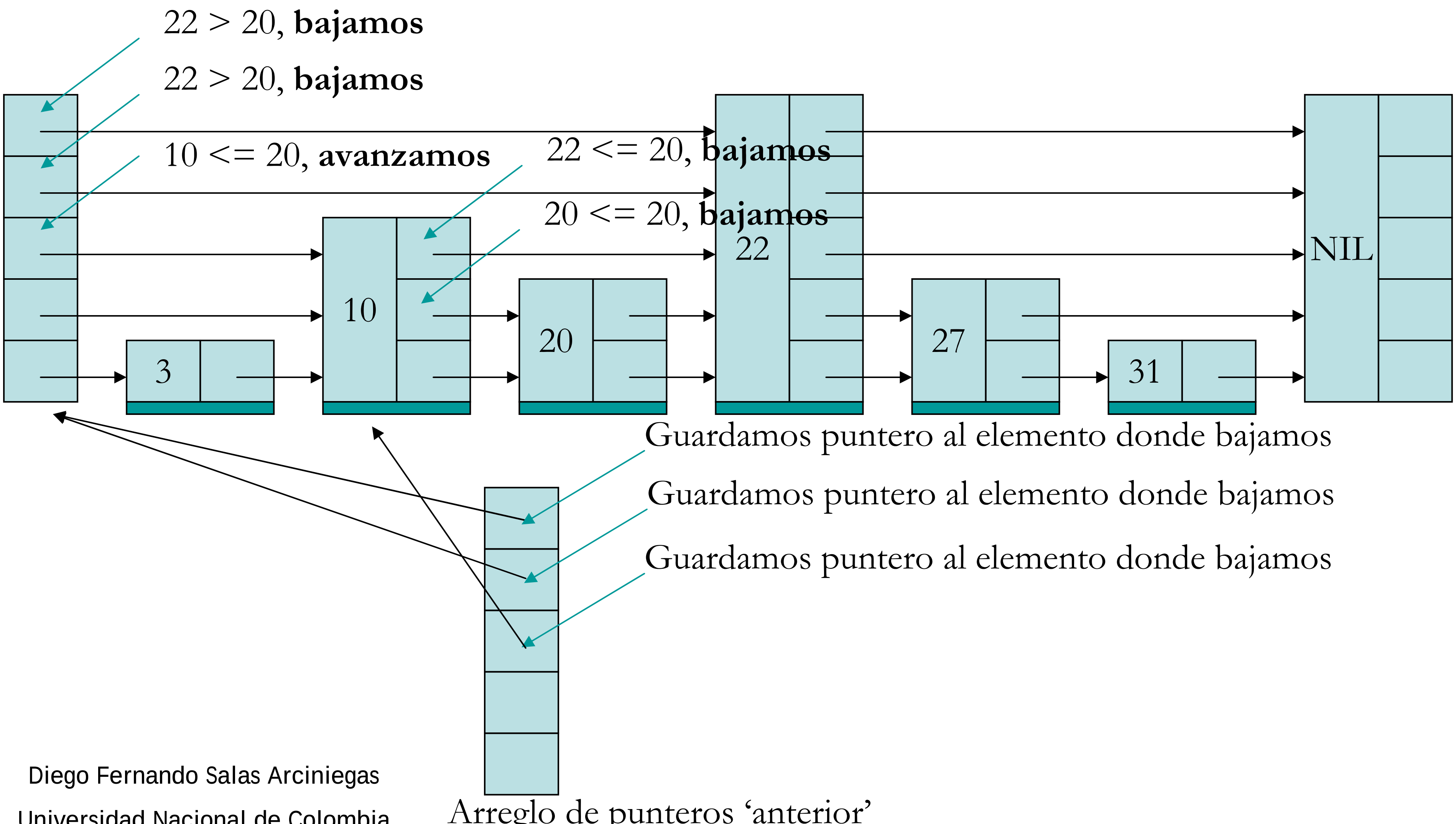


Arreglo de punteros 'anterior'

Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Eliminación

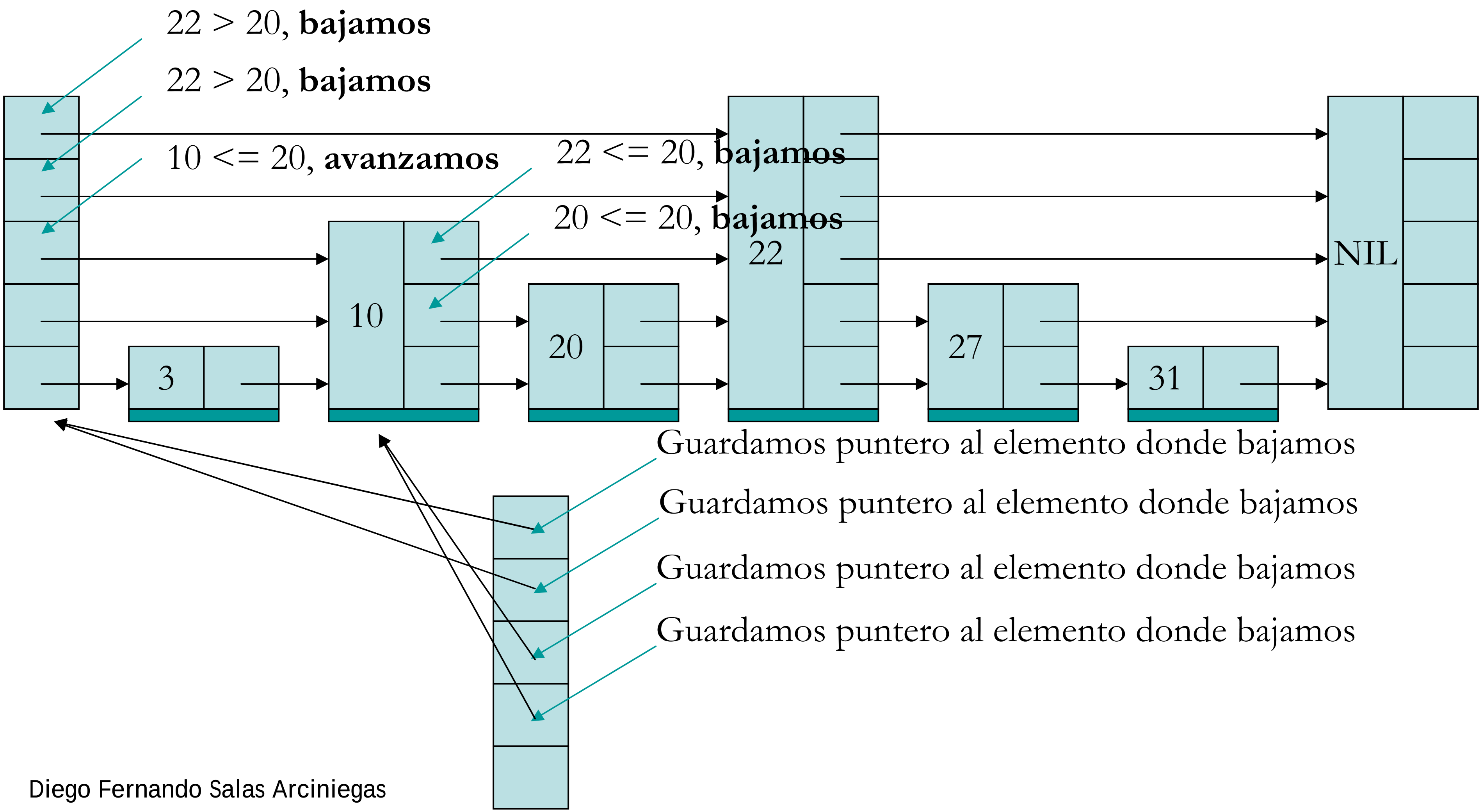
Ejemplo: queremos eliminar el objeto cuyo key sea 20



Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Eliminación

Ejemplo: queremos eliminar el objeto cuyo key sea 20



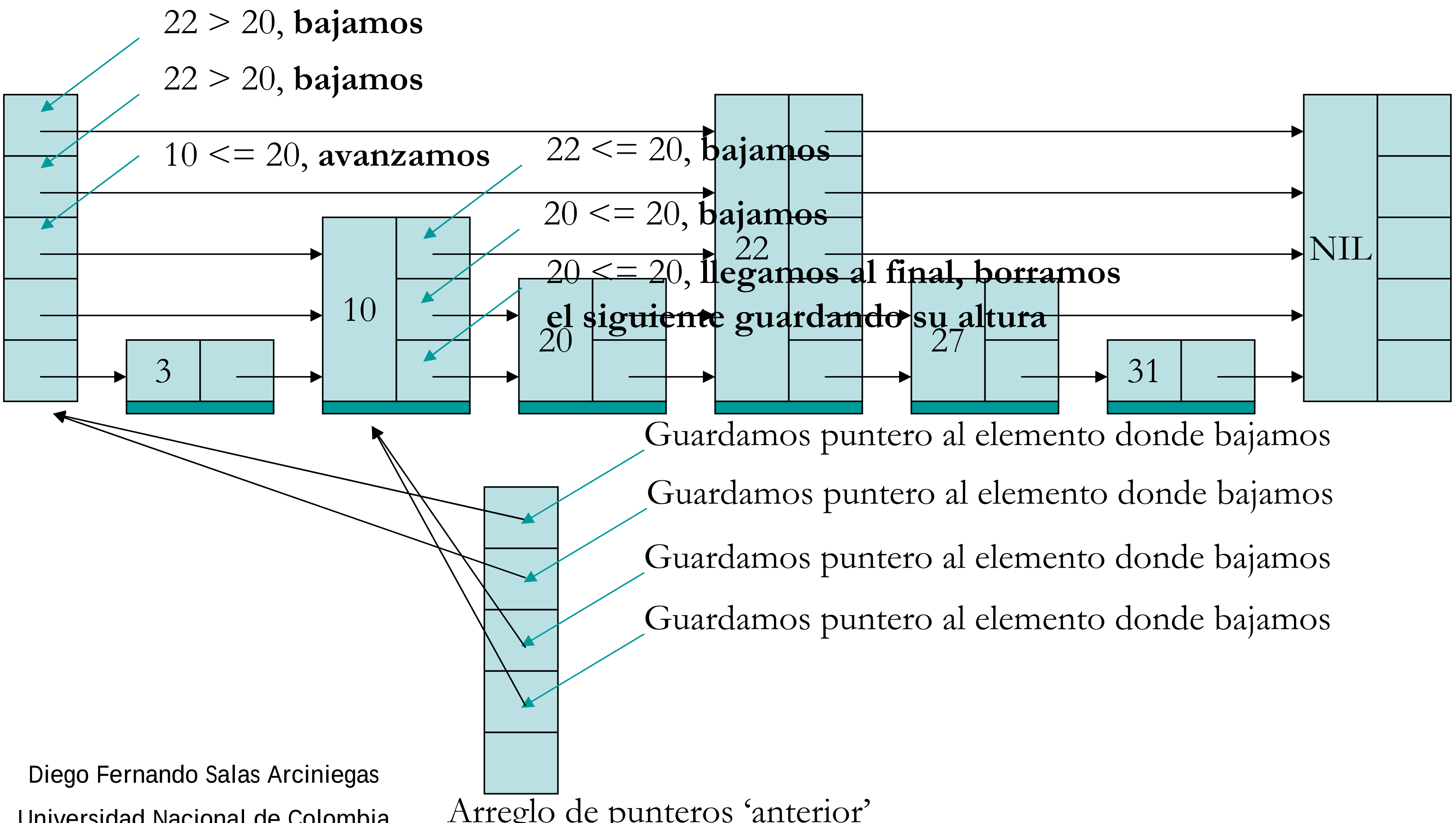
Arreglo de punteros ‘anterior’

Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Eliminación

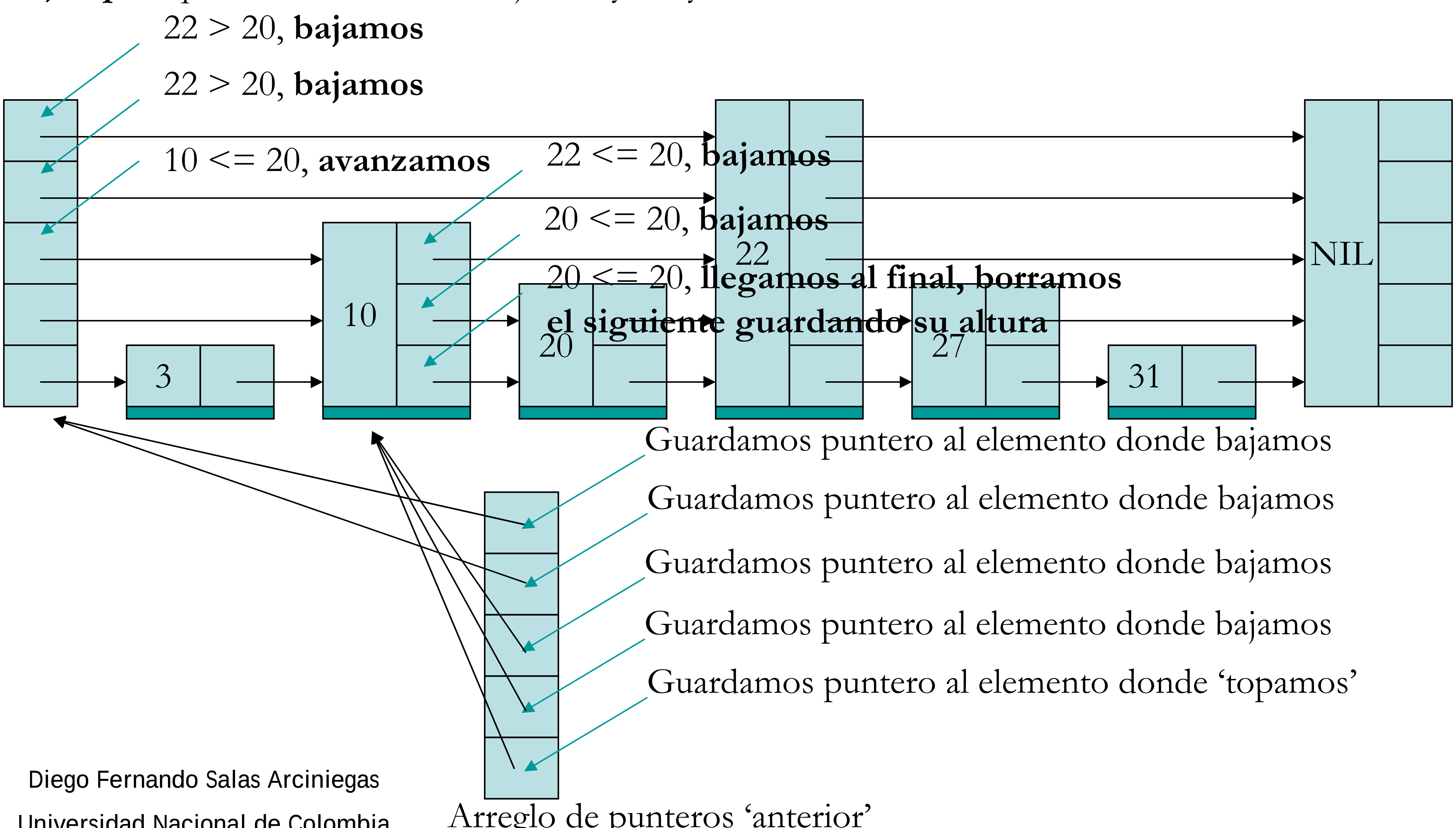
Ejemplo: queremos eliminar el objeto cuyo key sea 20



Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

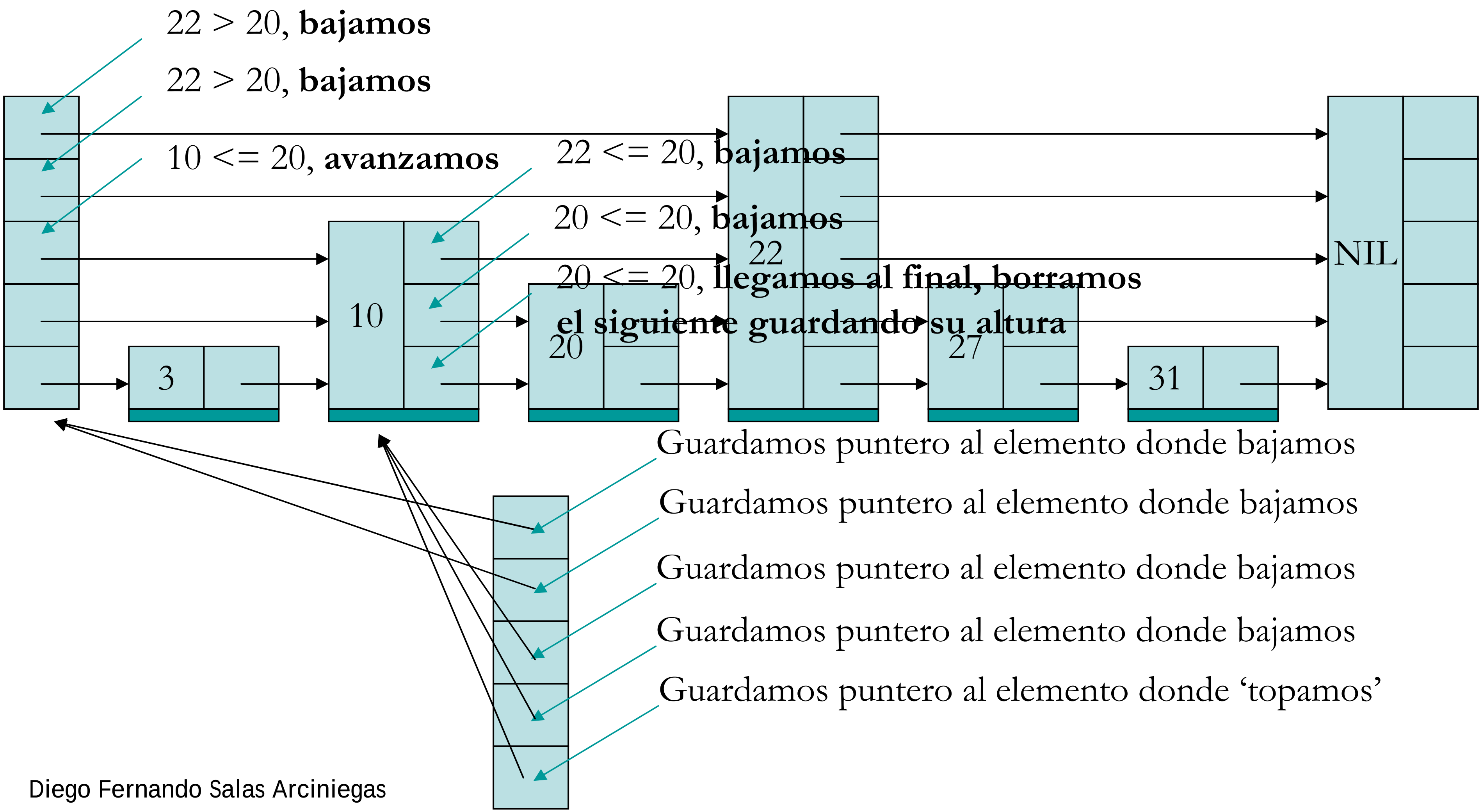
Eliminación

Ejemplo: queremos eliminar el objeto cuyo key sea 20



Eliminación

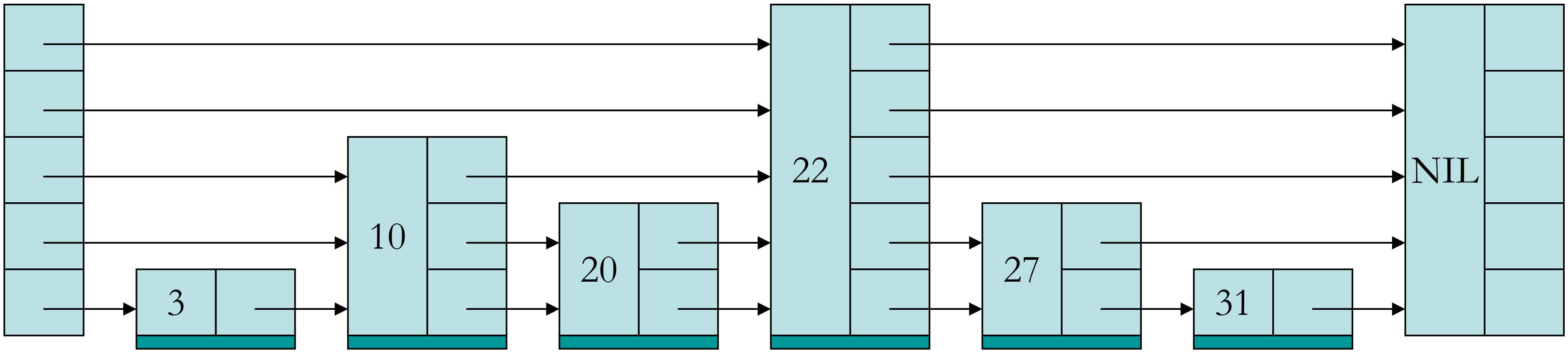
Ejemplo: queremos eliminar el objeto cuyo key sea 20



Arreglo de punteros 'anterior'

<u>Altura del elemento a borrar: 2</u>

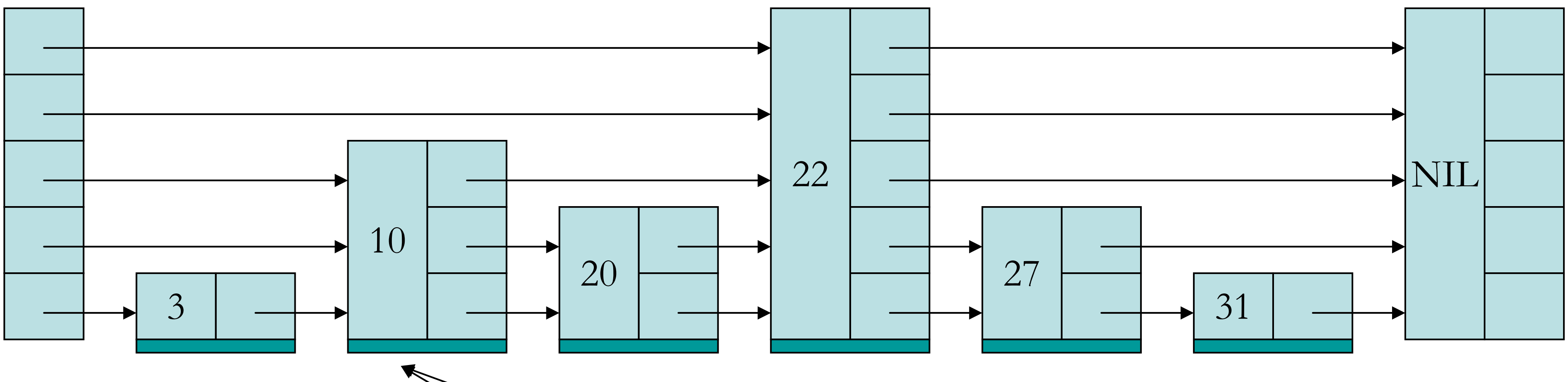
Eliminación



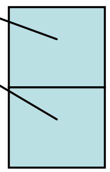
Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Altura del elemento a borrar: 2

Eliminación



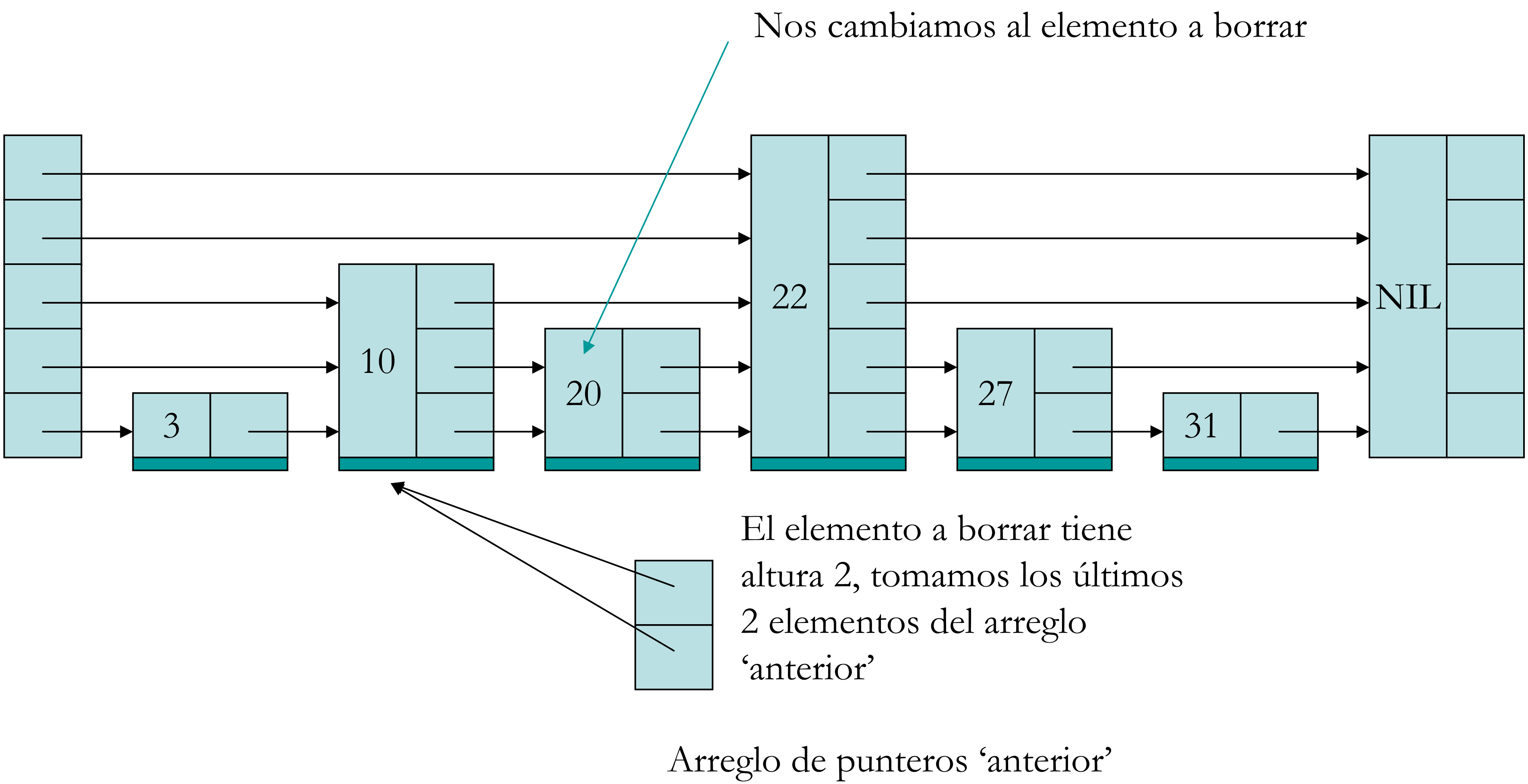
El elemento a borrar tiene altura 2, tomamos los últimos 2 elementos del arreglo 'anterior'



Arreglo de punteros 'anterior'

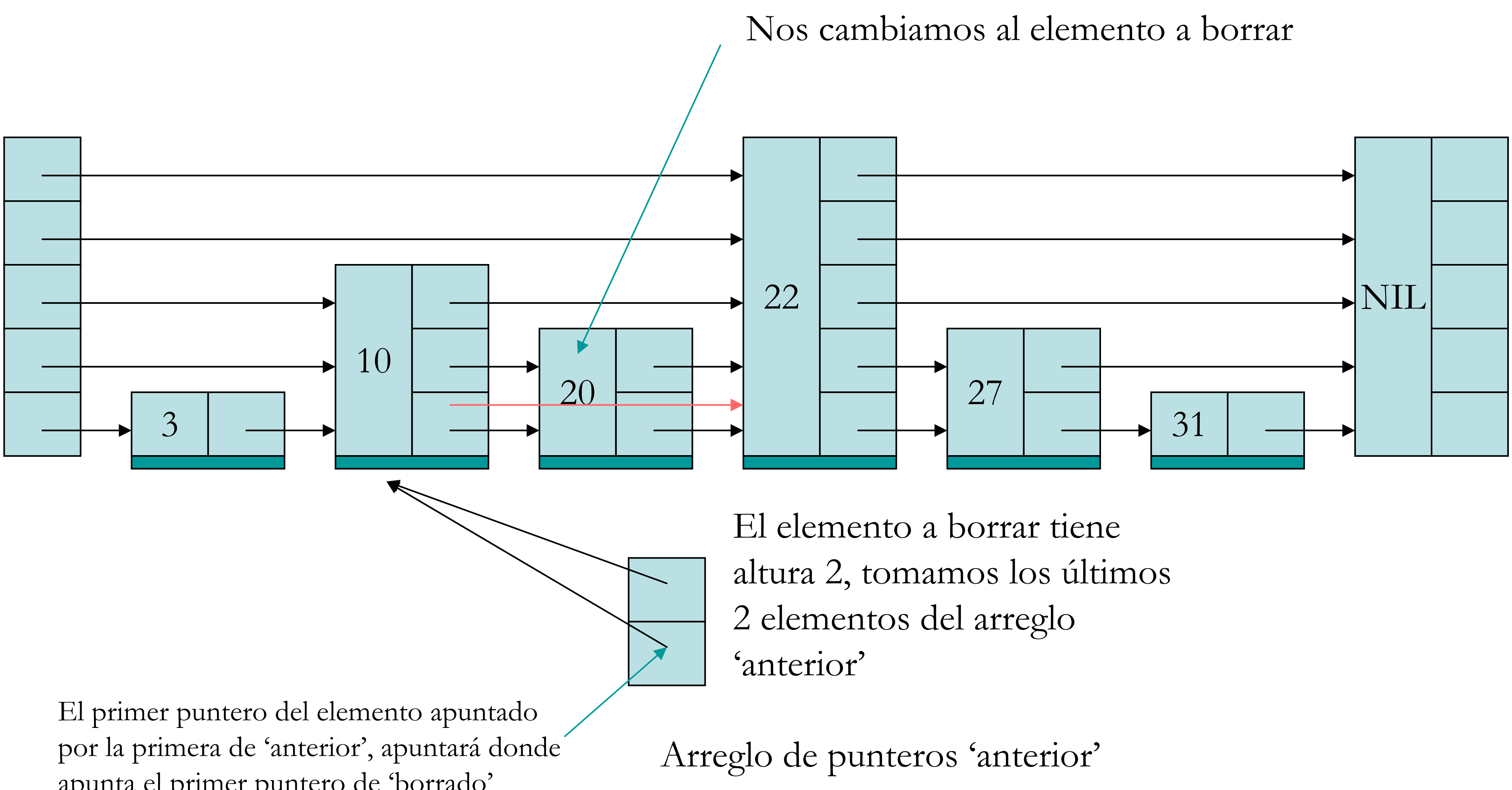
Altura del elemento a borrar: 2

Eliminación



Altura del elemento a borrar: 2

Eliminación

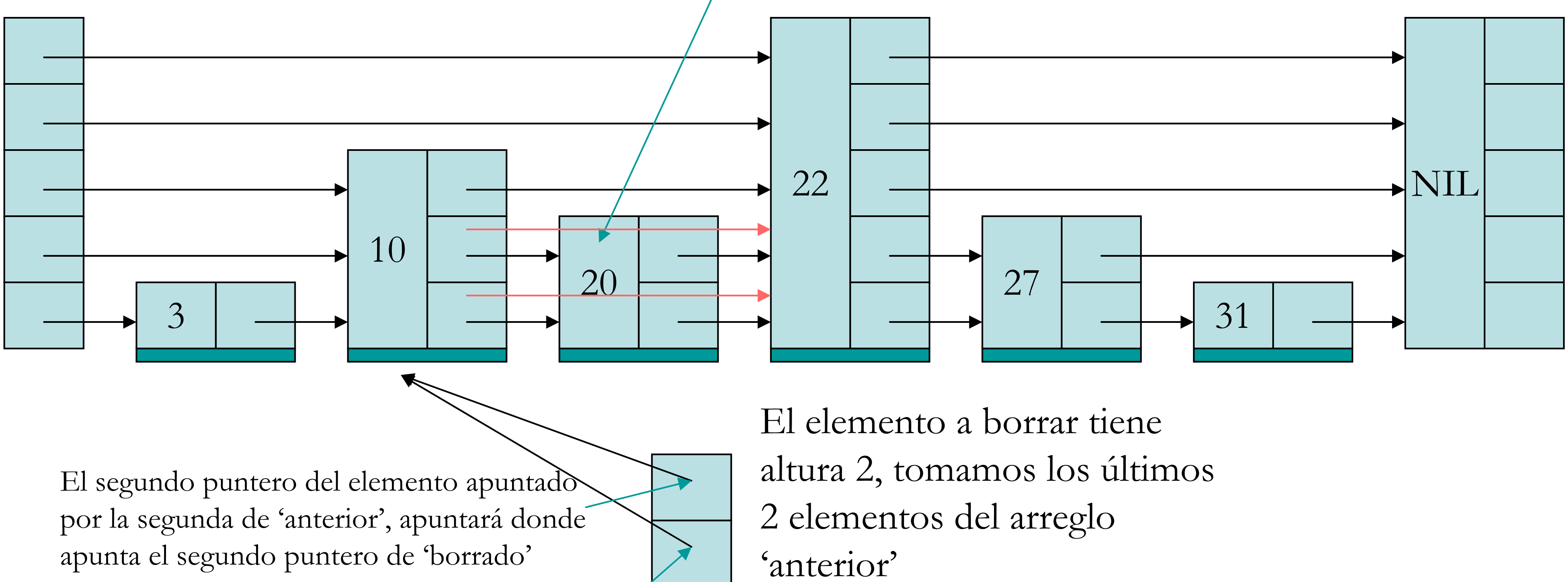


Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Altura del elemento a borrar: 2

Eliminación

Nos cambiamos al elemento a borrar



El elemento a borrar tiene altura 2, tomamos los últimos 2 elementos del arreglo ‘anterior’

El segundo puntero del elemento apuntado por la segunda de ‘anterior’, apuntará donde apunta el segundo puntero de ‘borrado’

El primer puntero del elemento apuntado por la primera de ‘anterior’, apuntará donde apunta el primer puntero de ‘borrado’

Arreglo de punteros ‘anterior’

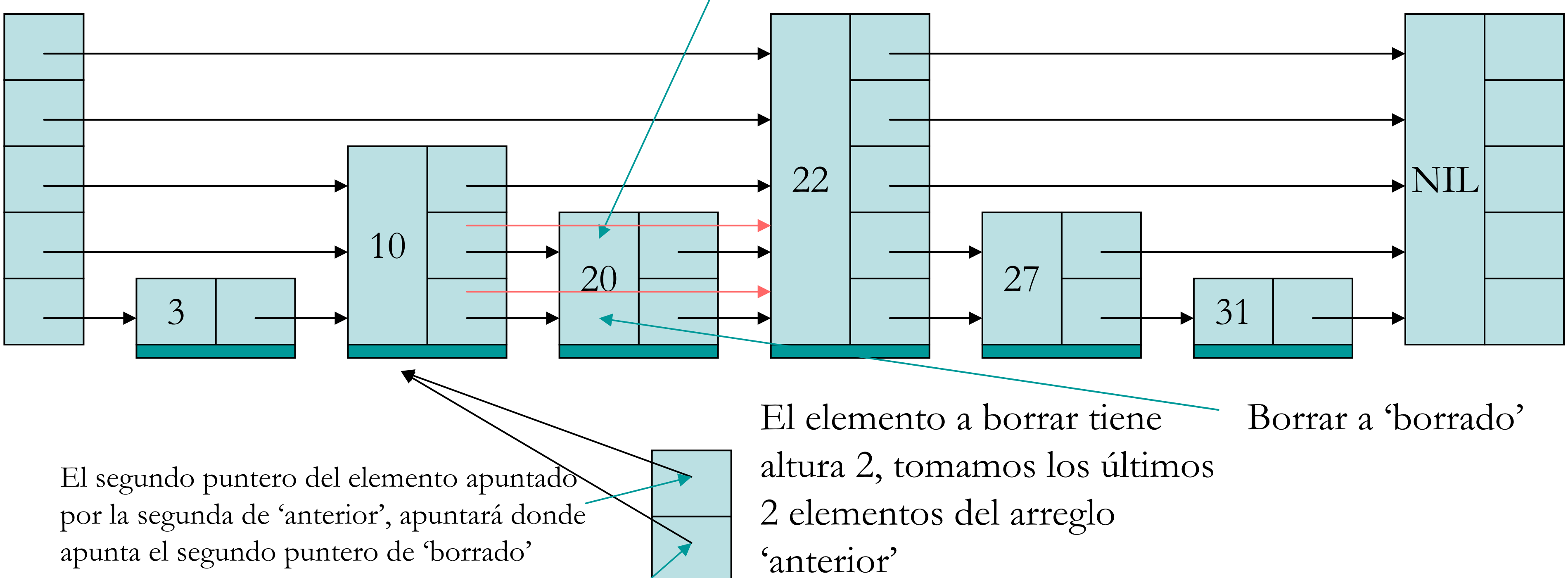
Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Altura del elemento a borrar: 2

Eliminación

Nos cambiamos al elemento a borrar



El elemento a borrar tiene altura 2, tomamos los últimos 2 elementos del arreglo ‘anterior’

Borrar a ‘borrado’

El segundo puntero del elemento apuntado por la segunda de ‘anterior’, apuntará donde apunta el segundo puntero de ‘borrado’

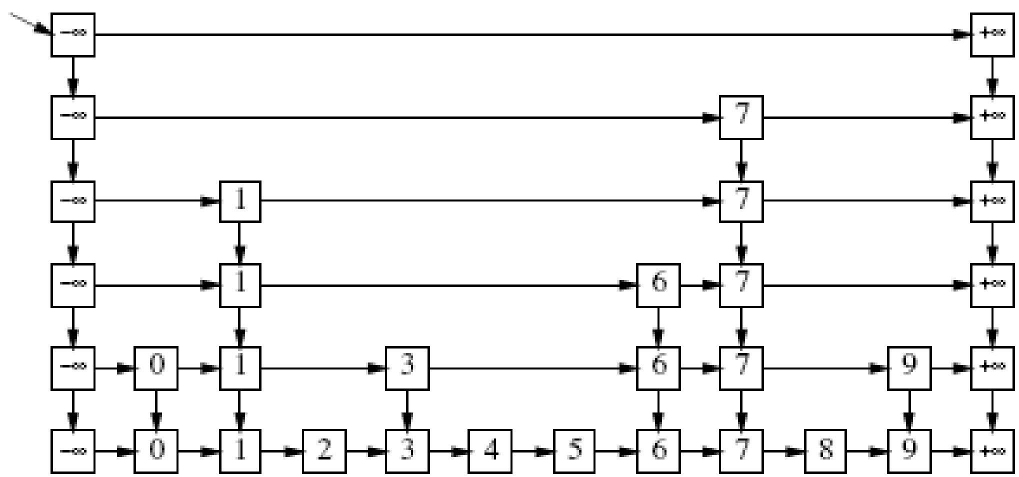
El primer puntero del elemento apuntado por la primera de ‘anterior’, apuntará donde apunta el primer puntero de ‘borrado’

Arreglo de punteros ‘anterior’

Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Altura del elemento a borrar: 2

Implementación



Implementación

```
public class IntSkipListNode {
        public int key;
        public IntSkipListNode[] next;
        IntSkipListNode(int i, int n) {
                key = i;
                next = new IntSkipListNode[n];
                for (int j = 0; j < n; j++) {
                        next[j] = null;
                }
        }
}
```

Seguro es *Log n*?

- Asumamos que las llaves son los números enteros de 1 a n. Sea L(x) el número de niveles de una Skip List que contiene un elemento *x*, sin contar el nivel más bajo.

- Cada nueva copia de *x* es creada con una probabilidad de *1/2*. Es decir, tenemos *p=1/2*.

- El valor esperado puede ser calculado recursivamente; al comienzo, tiramos una moneda, con probabilidad *1/2* cae cara (asumimos que es nuestro falso) y *L(x)=0,* por otro lado, con probabilidad *1/2* cae sello, por lo tanto, aumentamos *L(x)* y volvemos a calcular.

$$E\left[L(x)\right] = \frac{1}{2} * 0 + \frac{1}{2}\left(1 + E\left[L(x)\right]\right)$$

Diego Fernando Salas Arciniegas

Universidad Nacional de Colombia

Seguro es *Log n*?

- La recursión anterior nos da como resultado

$$E\left[L(x)\right] = 1$$

- Para poder analizar el costo esperado necesitamos una cota del número máximo de niveles que se pueden producir en una Skip List. Pero esta cota no puede ser calculada. En cambio, vamos a calcular que este número de niveles es O*(log n)* con una "alta probabilidad".

- El término "alta probabilidad" es un término técnico que significa que la probabilidad es al menos $1\text{-}1/n^c$. Para una constante $c>=1$.

Seguro es *Log n*?

• Para que una clave *x* aparezca en el nivel *k*, se debieron haber lanzado *k* sellos seguidos (ya que el nivel del nodo más alto es el nivel de la lista). Por lo tanto:

$$\Pr[L(x) \geq k] = 2^{-k}$$

•De esta manera, veamos la probabilidad de que nuestra algún nodo tenga *Log n* niveles.

$$\Pr[L(x) \geq 2\log n] = \frac{1}{n^2}$$

• Ahora, la Skip List tiene ese número de niveles, sí y sólo si alguno de sus nodos lo tiene.

Seguro es *Log n*?

- Así, por la definición dada, tenemos que:

$$\Pr[L \geq 2\log n] = \Pr[(L(1) \geq 2\log n) \vee (L(2) \geq 2\log n) \vee \ldots \vee (L(n) \geq 2\log n)]$$

$$\Pr[L \geq 2\log n] \leq \sum_{x=1}^{n} \Pr[L(x) \geq 2\log n] = \sum_{x=1}^{n} \frac{1}{n^2} = \frac{1}{n}$$

- Así, la Skip List tiene una "alta probabilidad" de tener O*(Log n)* niveles.

Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia

Algoritmos Aleatorizados

Muchas Gracias!

Diego Fernando Salas Arciniegas
Universidad Nacional de Colombia