

VERİ YAPILARILARI VE ALGORİTMALAR

Linked List

Giriş

1. Bağlı liste nedir?
2. Bağlı liste soyut veri türü
3. Neden bağlı listeler?
4. Bağlı listelerin dizi ve dinamik diziler iyi kıyaslanması
5. Singly Linked List
6. Doubly Linked List
7. Diğer
 1. Circular linked list
 2. Hafıza-Verimli çift yönlü bağlı listeler
 3. Unrolled Linked List
 4. Skip List

Bağlı Liste Nedir?

Linked List

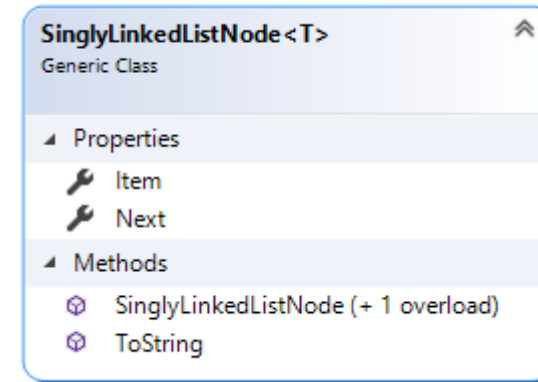
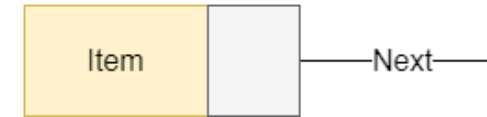
- İçindeki elemanların **doğrusal** olarak düzenlendiği veri yapısıdır.
- Dizilere benzer bir yapısı vardır ancak içindeki **elemanlara ulaşma yaklaşımı** ile dizilerden ayrılmaktadır.
- Dizilerde elemanlara ulaşmak için indisler kullanılırken; bağlı listelerde **işaretçiler** kullanılır.

Veri Yapısı Tanımlama

```
public class SinglyLinkedListNode<T>
{
    public T Item { get; set; }
    public SinglyLinkedListNode<T> Next { get; set; }
    public SinglyLinkedListNode()
    {

    }
    public SinglyLinkedListNode(T item)
    {
        Item = item;
    }
    public override string ToString()
    {
        return $"{Item}";
    }
}
```

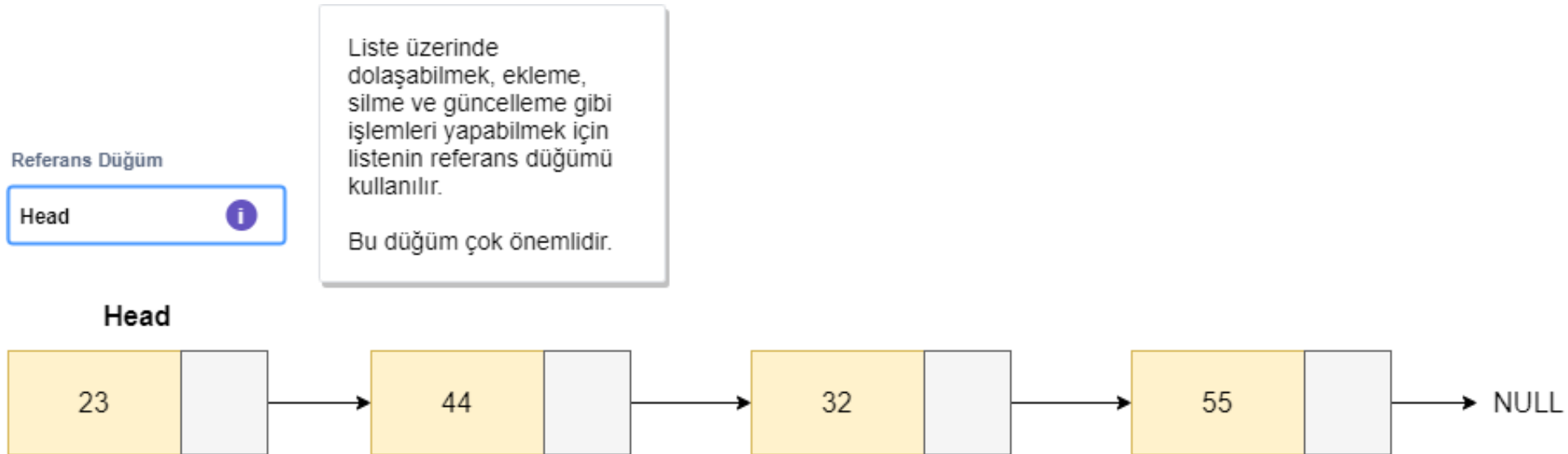
SinglyLinkedListNode<T>



- Bir bağlı liste düğüm tasarım örneği.

Liste Tanımlama

- İlgili veri türlerinin bir araya getirilmesiyle bir bağlı liste yapısı oluşturulabilir.



Bağlı Liste Soyut Veri Türü

(Abstract Data Type, ADT)

- **Bağlı Liste İşlevleri** (Main linked lists operations)
 - **Insert:** Listeye eleman ekleme.
 - **Delete:** Listedeki eleman silme.
- **Yardımcı Bağlı Liste İşlevleri** (Auxiliary linked lists operations)
 - **Delete List:** Listedeki tüm elemanları siler ve listeyi yok eder.
 - **Count:** Listedeki eleman sayısını döner.
 - **Find:** Listedeki bir düğümü işaret eder.

Diziler

Avantajları

- Basit ve kullanımı kolay
- Elemanlara doğrudan erişim imkanı sunar, $O(1)$.

Dezavantajları

- Önceden bellek tahsisi ve dizideki boş elemanları için belleğin işgal edilmesi.
- Sabit boyut
- Bir blok tahsisi (One block allocation)
- Konuma dayalı karmaşık ekleme (Complex position-based insertion)

Bağlı Listeler

Avantajları

- Bağlı liste sabit zamanda genişletilebilir, $O(1)$.
- Önceden bellek tahsisini engeller (preallocates)

Dezavantajları

- Erişim zamanı (access time)
- En kötü durumda bir elemana erişmenin maliyeti, $O(n)$.

Bağlı Listeler

Parameter	Linked list	Array	Dynamic array
Indexing	$O(n)$	$O(1)$	$O(1)$
Insertion/deletion at beginning	$O(1)$	$O(n)$, if array is not full (for shifting the elements)	$O(n)$
Insertion at ending	$O(n)$	$O(1)$, if array is not full	$O(1)$, if array is not full $O(n)$, if array is full
Deletion at ending	$O(n)$	$O(1)$	$O(n)$
Insertion in middle	$O(n)$	$O(n)$, if array is not full (for shifting the elements)	$O(n)$
Deletion in middle	$O(n)$	$O(n)$, if array is not full (for shifting the elements)	$O(n)$
Wasted space	$O(n)$ (for pointers)	0	$O(n)$

Singly Linked Lists

Tek Bağlantılı Listeler

Singly Linked Lists

Temel İşlevler (Main operations)

Listede dolaşma

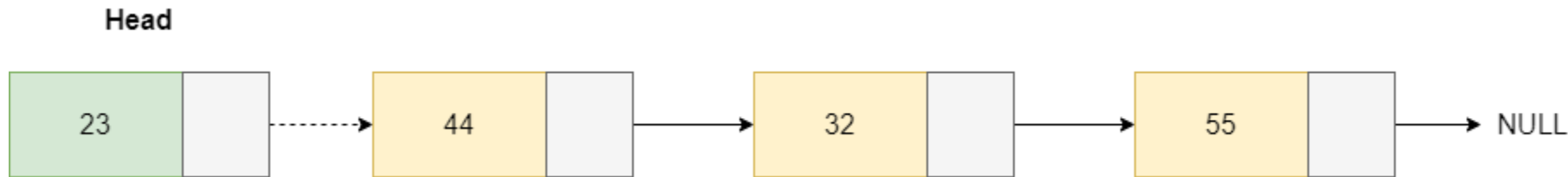
Listeye bir öge ekleme

Listeden bir öge silme

Singly Linked Lists

Listeye Ekleme (Inserting)

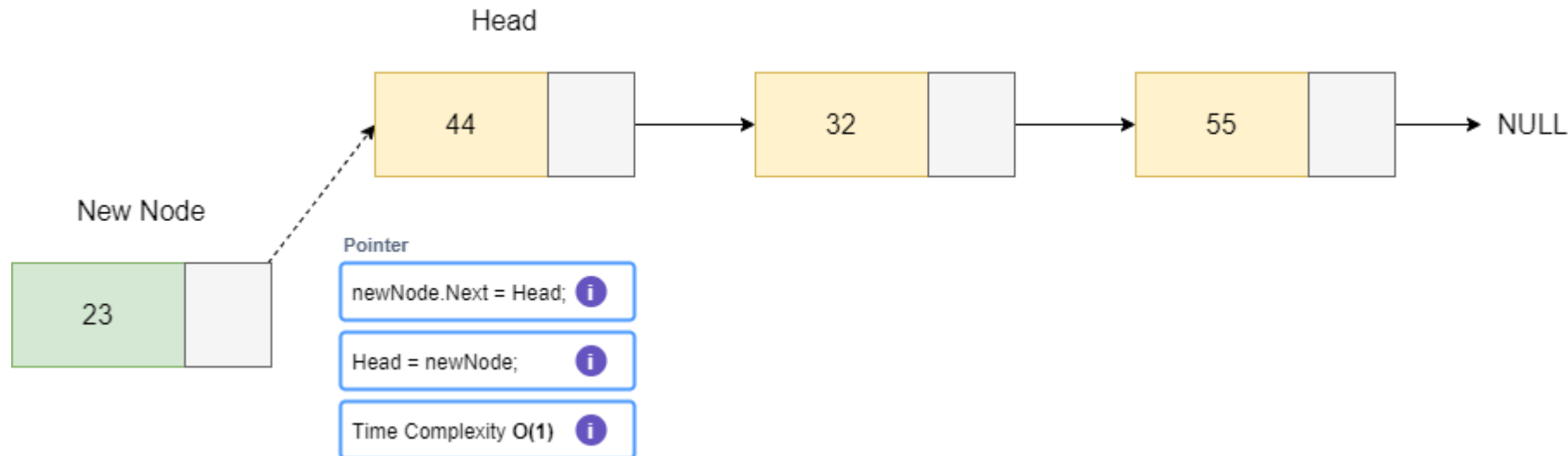
- Liste başına ekleme yapma
- Kuyruğa ekleme (liste sonuna) ekleme yapma
- Araya ekleme yapma



Singly Linked Lists

Liste Başına Ekleme (Inserting at the beginning)

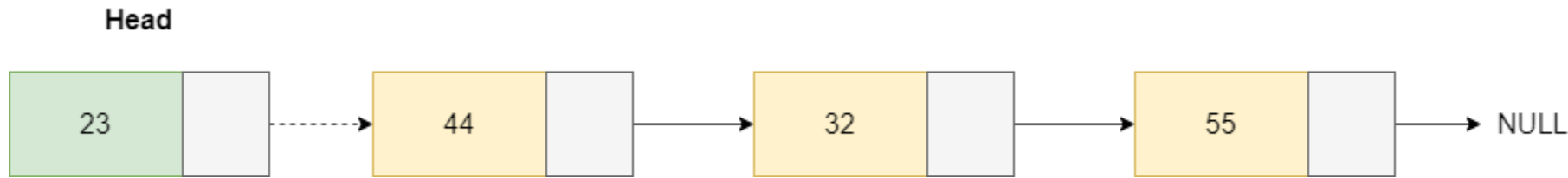
- Yeni düğümün ileri işaretçisi mevcut kök düğümü (head) işaret edecek şekilde güncellenir.
- Yeni düğüm, kök düğüm (head) olarak işaretlenir.



Singly Linked Lists

Liste Başına Ekleme (Inserting at the beginning)

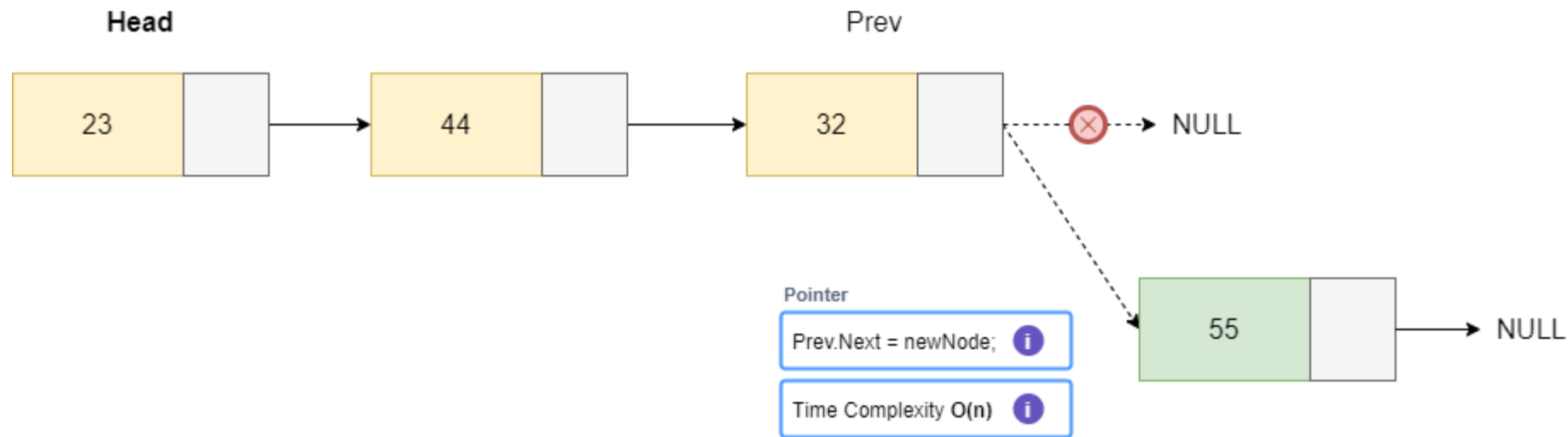
- Yeni düğümün ileri işaretçisi mevcut kök düğümü (**Head**) işaret edecek şekilde güncellenir.
- Yeni düğüm, kök düğüm (**Head**) olarak işaretlenir.



Singly Linked Lists

Liste Sonuna Ekleme (Inserting at the ending)

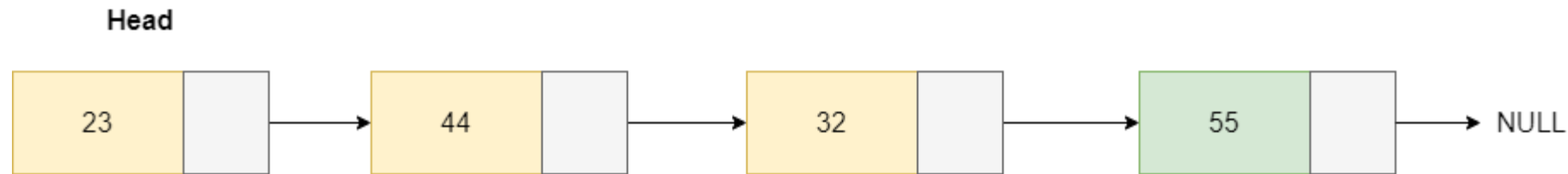
- İleri işaretçisi **NULL** olan yeni düğüm eklenir.
- Son düğümün işaretçisi yeni düğümü gösterecek şekilde güncellenir.



Singly Linked Lists

Liste Sonuna Ekleme (Inserting at the ending)

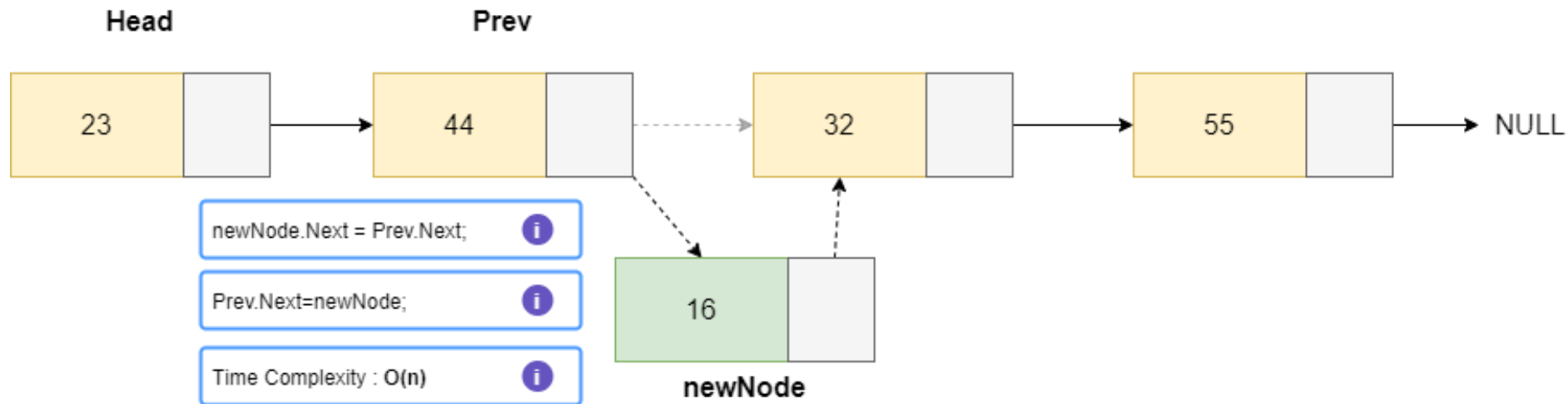
- İleri işaretçisi **NULL** olan yeni düğüm eklenir.
- Son düğümün işaretçisi yeni düğümü gösterecek şekilde güncellenir.



Singly Linked Lists

Listede Araya Ekleme (Inserting at the middle)

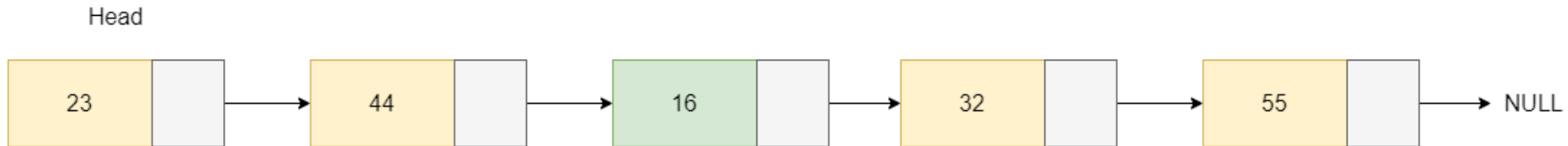
- Ekleme yapılacak pozisyonundan önceki eleman (**Prev**) bulunur.
- Yeni düğümün işaretçisi (**newNode.Next = prev.Next**) güncellenir.
- Daha sonra önceli düğümün işaretçisi (**Prev.Next = newNode**) güncellenir.



Singly Linked Lists

Listede Araya Ekleme (Inserting at the middle)

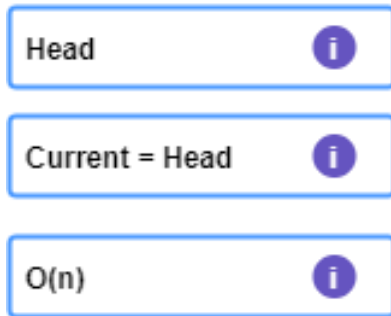
- Ekleme yapılacak pozisyondan önceki eleman (**Prev**) bulunur.
- Yeni düğümün işaretçisi (**newNode.Next = prev.Next**) güncellenir.
- Daha sonra önceli düğümün işaretçisi (**Prev.Next = newNode**) güncellenir.



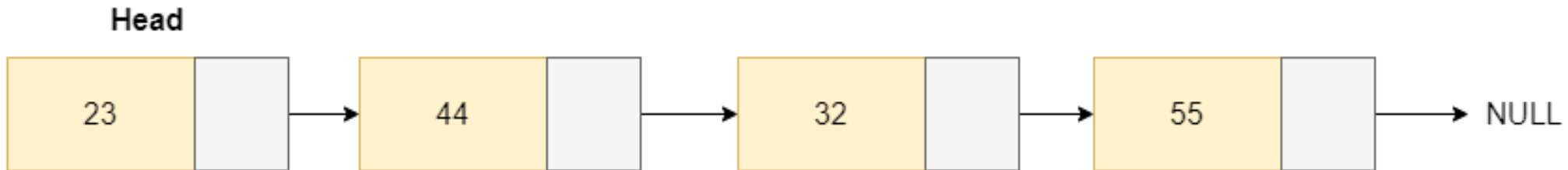
Singly Linked Lists

Listeyi Dolaşmak (Traversing the list)

Referans Düğüm



```
if(current==null) { ... }  
while(current!=null)  
{  
    current = current.Next;  
    ...  
}
```



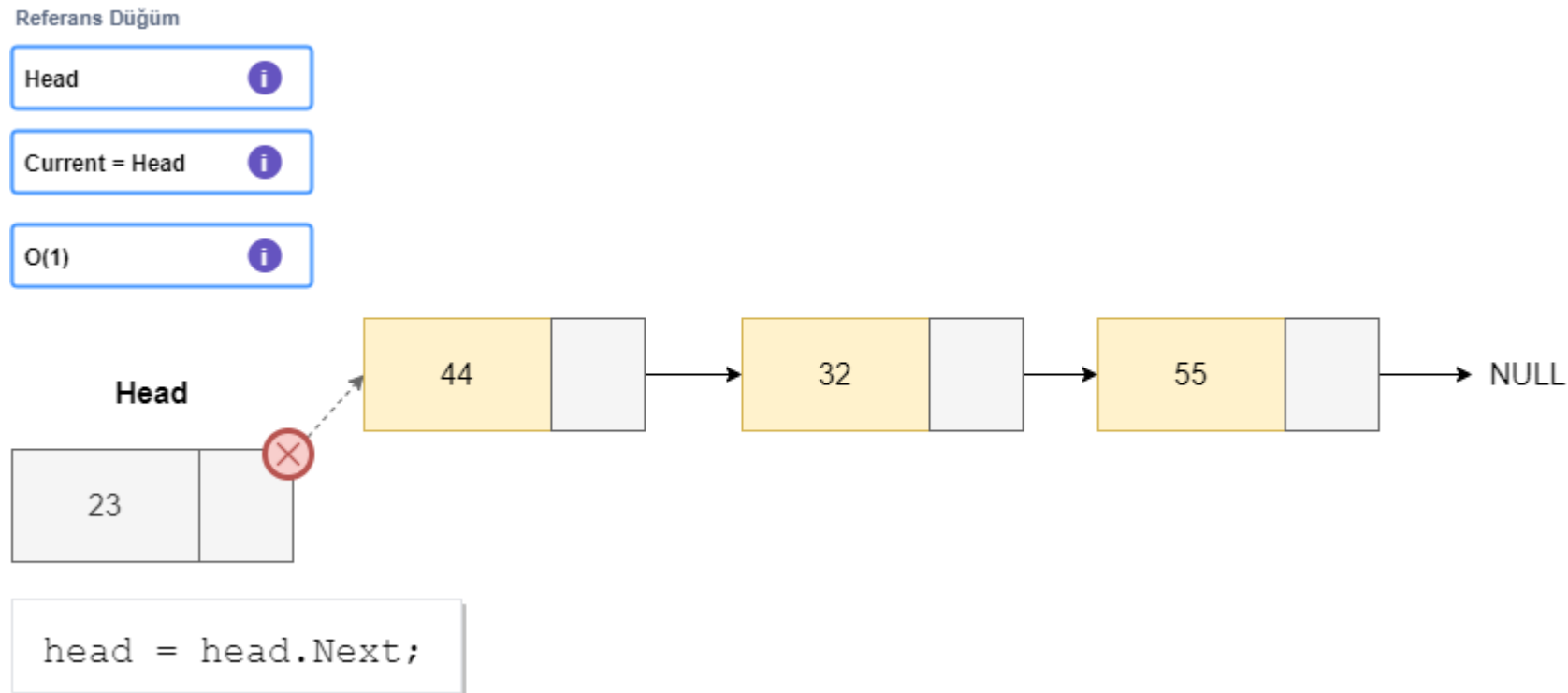
- **Current** ile **Head** referans alınarak gezintiye başlanır.
- Düğüm içeriği yazdırılır ve daha sonra **Current!=null** iken çevrime devam edilir.
- **NULL** değer ile karşılaşılması durumunda çevrimden çıkılır.
- Gezinmenin maliyeti $O(n)$.

Singly Linked Lists

İlk Düğümü Silme

(Deleting first node in the singly linked list)

- Head düğümünün işaretçisi sonraki elemanı gösterecek şekilde güncellenir.



Singly Linked Lists

İlk Düğümü Silme

(Deleting first node in the singly linked list)

- Head düğümünün işaretçisi sonraki elemanı gösterecek şekilde güncellenir.

Referans Düğüm

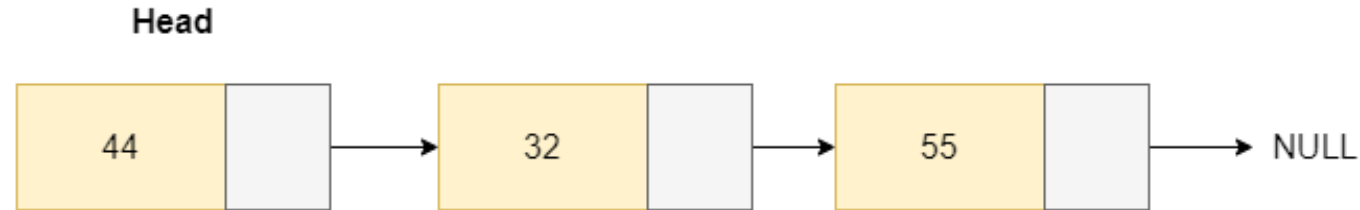
Head



Current = Head



O(1)



Singly Linked Lists

Son Düğümü Silme (Deleting the last node in the singly linked list)

- Liste üzerinde dolaşılır ve liste sonundaki elemandan önceki elemanın (**Prev**) tutulması sağlanır.

Referans Düğüm

Head



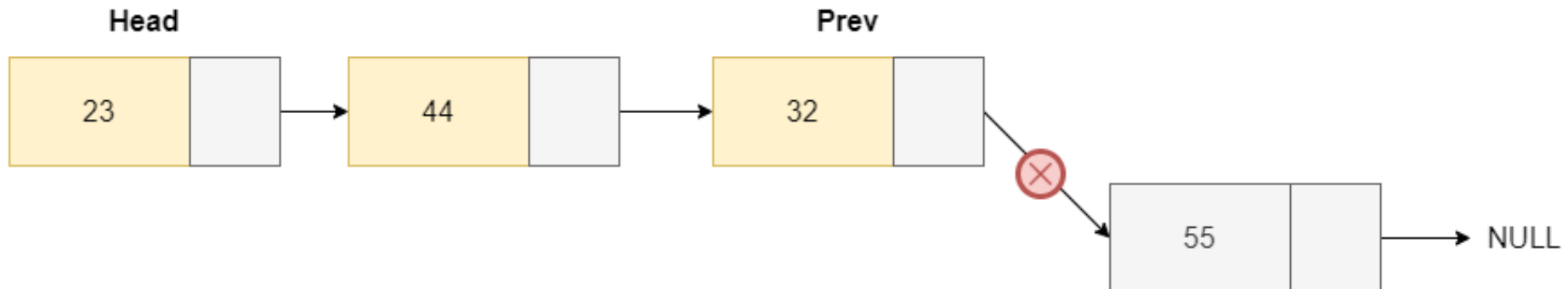
Current = Head



O(n)



```
while(current.Next!=null)
{
    prev = current;
    current = current.Next;
}
prev.Next=null;
```



Singly Linked Lists

Son Düğümü Silme (Deleting the last node in the singly linked list)

- Liste üzerinde dolaşılır ve liste sonundaki elemandan önceki elemanın (**Prev**) tutulması sağlanır.

Referans Düğüm

Head



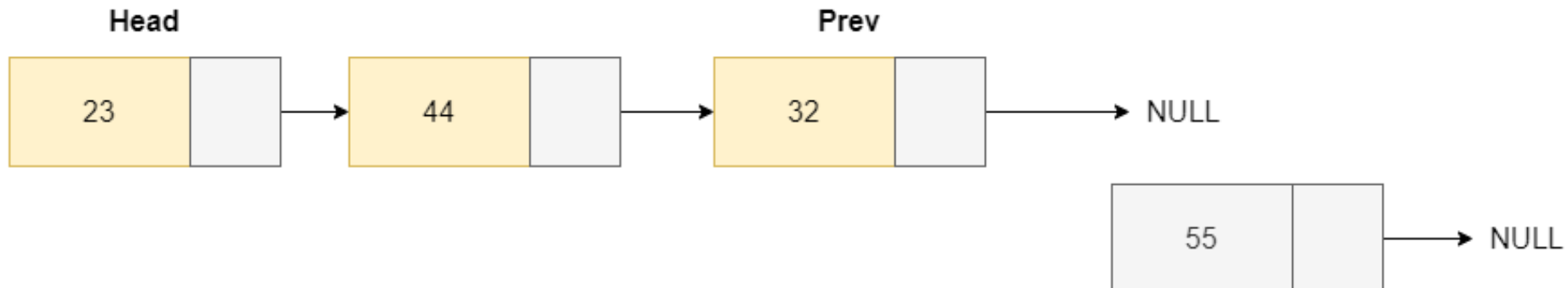
Current = Head



O(n)



```
while(current.Next!=null)
{
    prev = current;
    current = current.Next;
}
prev.Next=null;
```



Singly Linked Lists

Ara Düğümü Silme (Deleting an intermediate node in the singly linked list)

- Silinmek istenen düğümden önceki düğüm (**Prev**) bulunur. İlgili düğümün işaretçisinin gösterdiği alan, silinecek düğümün işaretçisi ile güncellenir. Silinecek eleman yok edilir.

Referans Düğüm

Head



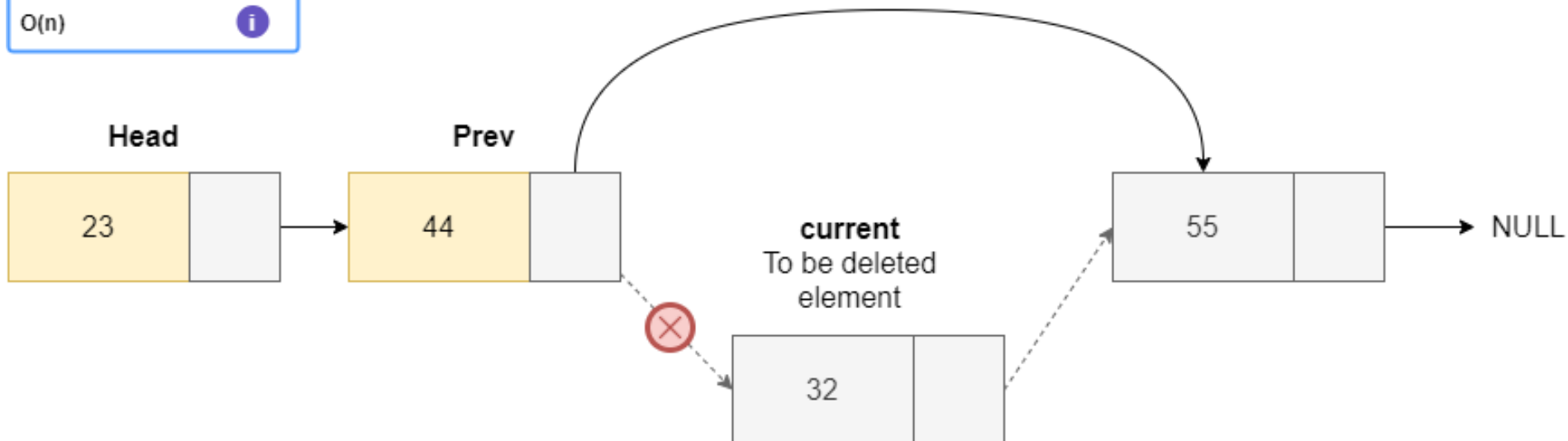
Current = Head



O(n)



```
Prev.Next = current.Next;
```

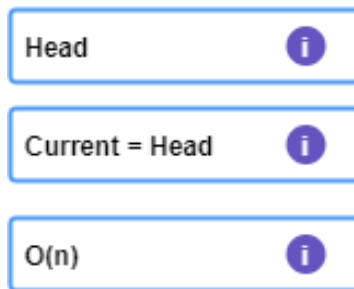


Singly Linked Lists

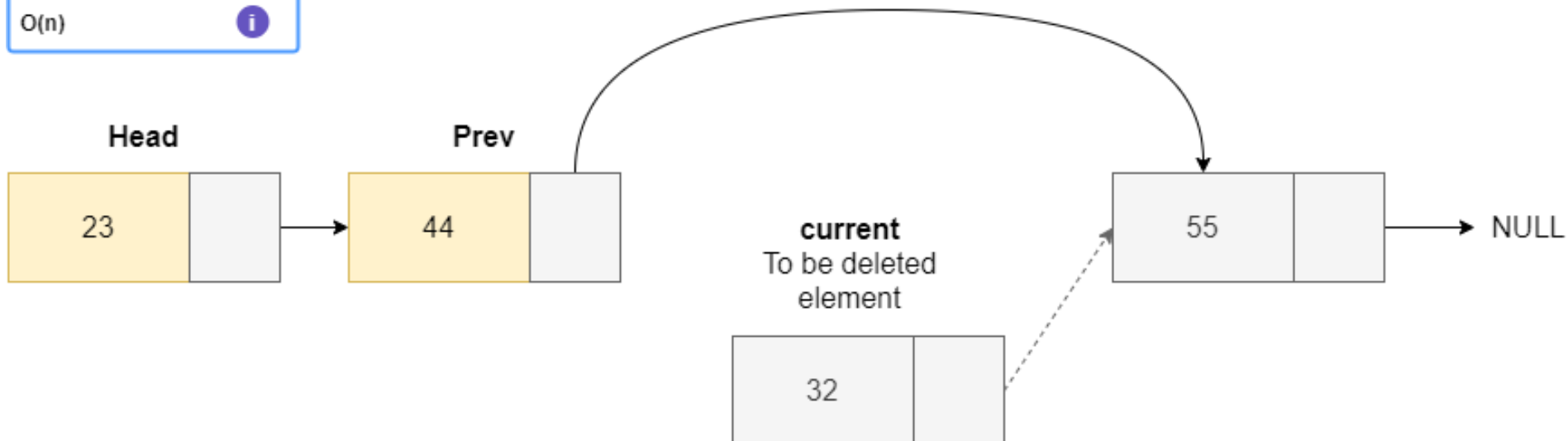
Ara Düğümü Silme (Deleting an intermediate node in the singly linked list)

- Silinmek istenen düğümden önceki düğüm (**Prev**) bulunur. İlgili düğümün işaretçisinin gösterdiği alan, silinecek düğümün işaretçisi ile güncellenir. Silinecek eleman yok edilir.

Referans Düğüm



```
Prev.Next = current.Next;
```

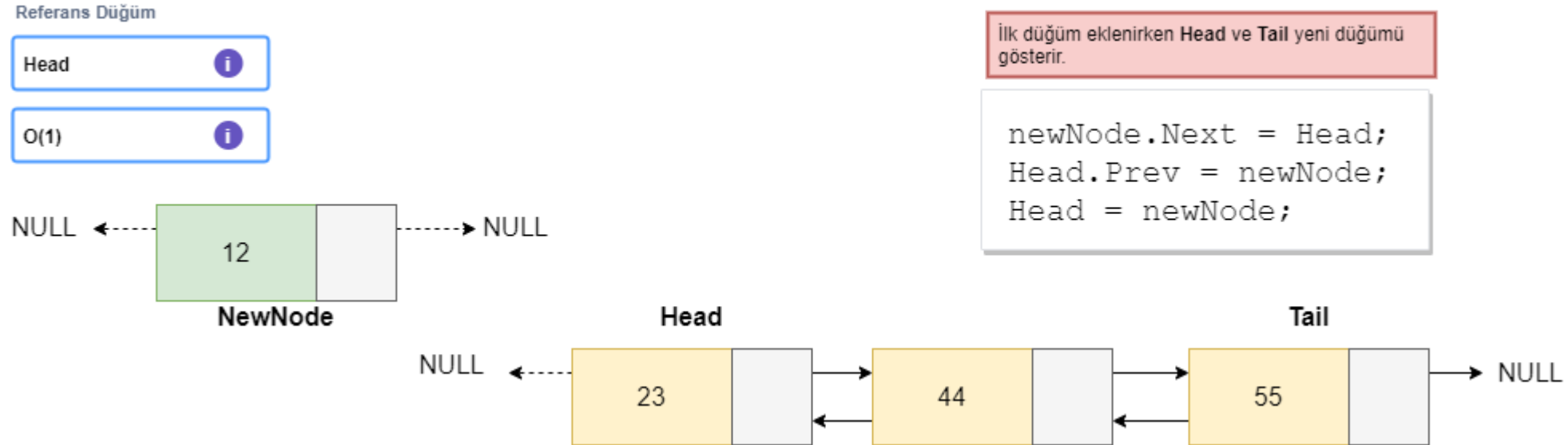


Doubly Linked Lists

Çift Bağlantılı Listeler

Doubly Linked Lists

Liste Başına Ekleme (Inserting at the beginning)





- Sol ve sağ işaretçileri **NULL** olan yeni düğüm oluşturulur.
- Yeni düğümün sağ işaretçisi (**Next**) mevcut kök düğümü işaret edecek şekilde güncellenir.
- Mevcut kök düğümün sol işaretçisi (**Prev**) yeni düğümü işaret edecek şekilde güncellenir.

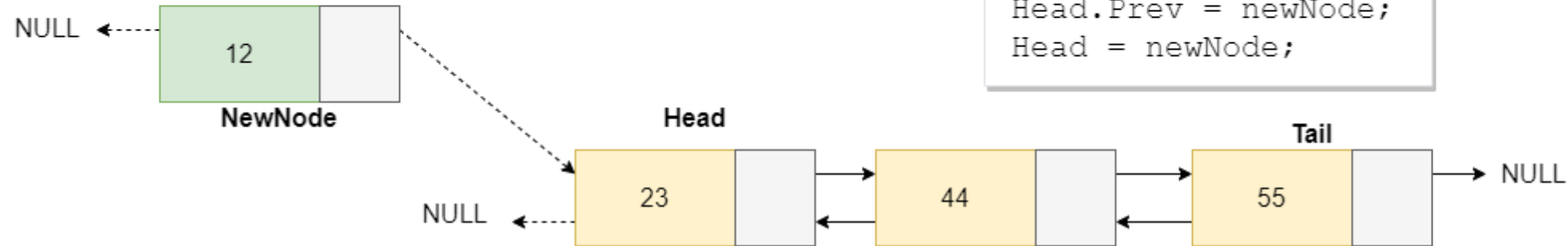
Doubly Linked Lists

Liste Başına Ekleme (Inserting at the beginning)

Referans Düğüm

Head 

O(1) 





- Sol ve sağ işaretçileri **NULL** olan yeni düğüm oluşturulur.
- Yeni düğümün sağ işaretçisi (**Next**) mevcut kök düğümü işaret edecek şekilde güncellenir.
- Mevcut kök düğümün sol işaretçisi (**Prev**) yeni düğümü işaret edecek şekilde güncellenir.

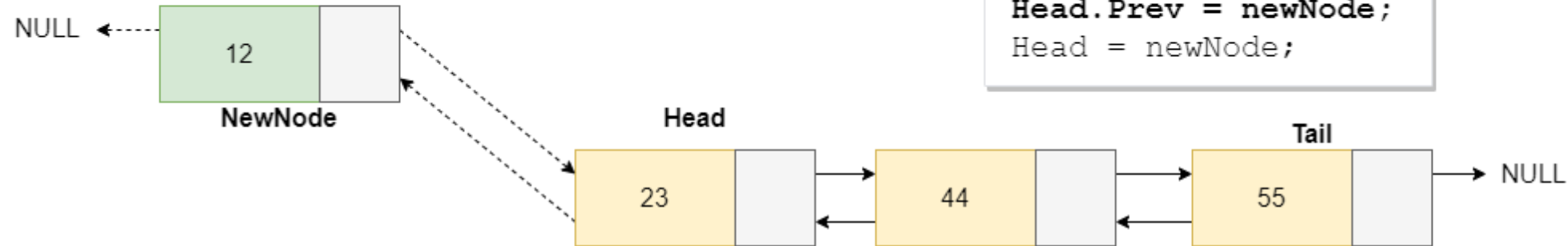
Doubly Linked Lists

Liste Başına Ekleme (Inserting at the beginning)

Referans Düğüm

Head 

O(1) 




- Sol ve sağ işaretçileri **NULL** olan yeni düğüm oluşturulur.
- Yeni düğümün sağ işaretçisi (**Next**) mevcut kök düğümü işaret edecek şekilde güncellenir.
- Mevcut kök düğümün sol işaretçisi (**Prev**) yeni düğümü işaret edecek şekilde güncellenir.

Doubly Linked Lists

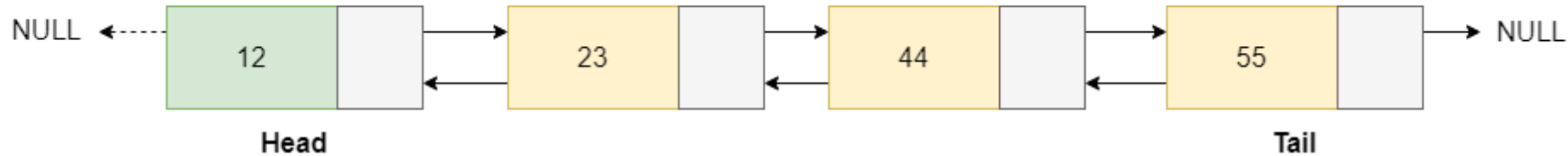
Liste Başına Ekleme (Inserting at the beginning)

Referans Düğüm

Head 

O(1) 

```
newNode.Next = Head;  
Head.Prev = newNode;  
Head = newNode;
```



- Son olarak liste başını işaret eden **Head** düğümü güncellenir.

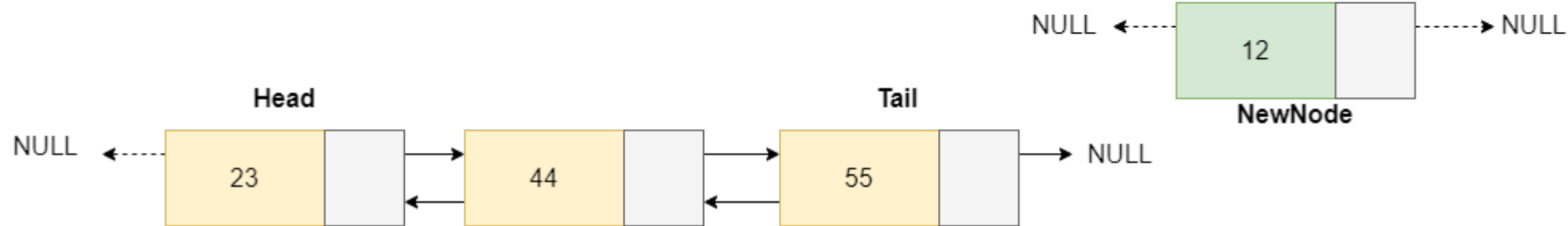
Doubly Linked Lists

Liste Sonuna Ekleme (Inserting at the ending)

Referans Düğüm



```
Tail.Next = newNode;  
newNode.Prev = Tail;  
Tail = newNode;
```

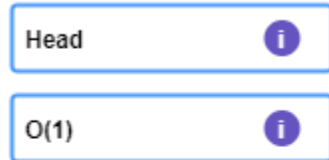


- Sol ve sağ işaretçileri **NULL** olan yeni düğüm oluşturulur.
- Kuyruğun ileri işaretçisi (**Tail.Next = newNode**) yeni düğümü işaret eder.
- Yeni düğümün önceki işaretçisi (**newNode.Prev = Tail**) kuyruğu işaret eder.
- Kuyruk yeni düğümü işaret (**Tail = newNode**) eder.

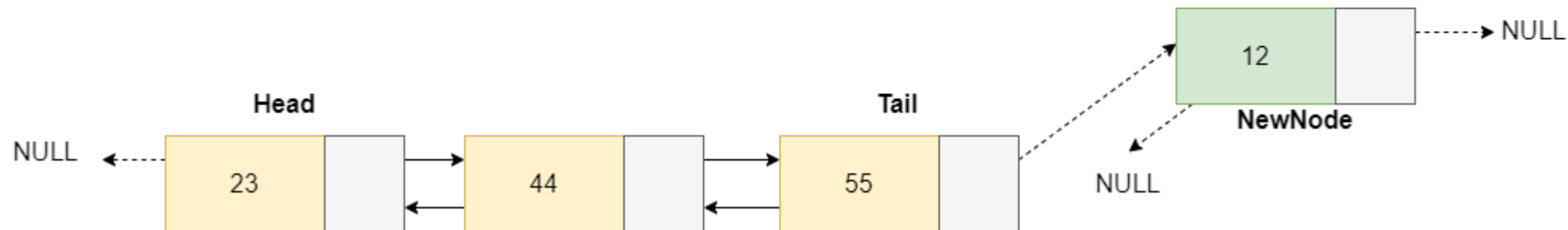
Doubly Linked Lists

Liste Sonuna Ekleme (Inserting at the ending)

Referans Düğüm



```
Tail.Next = newNode;  
newNode.Prev = Tail;  
Tail = newNode;
```



- Sol ve sağ işaretçileri NULL olan yeni düğüm oluşturulur.
- Kuyruğun ileri işaretçisi (**Tail.Next = newNode**) yeni düğümü işaret eder.
- Yeni düğümün önceki işaretçisi (**newNode.Prev = Tail**) kuyruğu işaret eder.
- Kuyruk yeni düğümü işaret (**Tail = newNode**) eder.

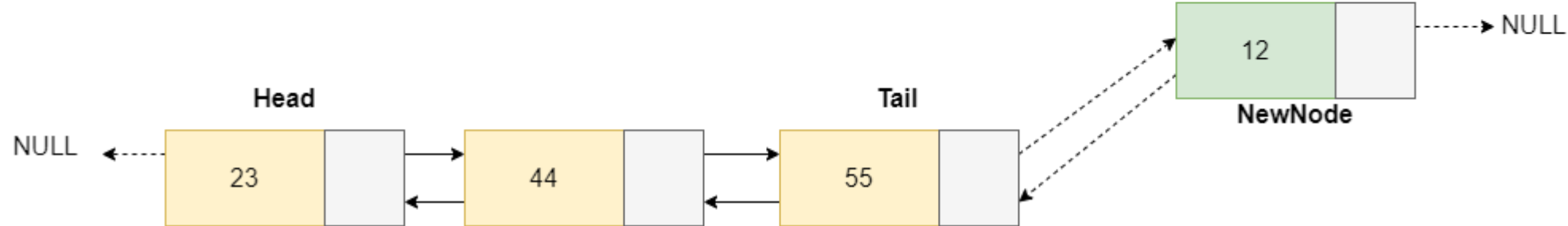
Doubly Linked Lists

Liste Sonuna Ekleme (Inserting at the ending)

Referans Düğüm



```
Tail.Next = newNode;  
newNode.Prev = Tail;  
Tail = newNode;
```



- Sol ve sağ işaretçileri NULL olan yeni düğüm oluşturulur.
- Kuyruğun ileri işaretçisi (**Tail.Next = newNode**) yeni düğümü işaret eder.
- Yeni düğümün önceki işaretçisi (**newNode.Prev = Tail**) kuyruğu işaret eder.
- Kuyruk yeni düğümü işaret (**Tail = newNode**) eder.

Doubly Linked Lists

Liste Sonuna Ekleme (Inserting at the ending)

Referans Düğüm

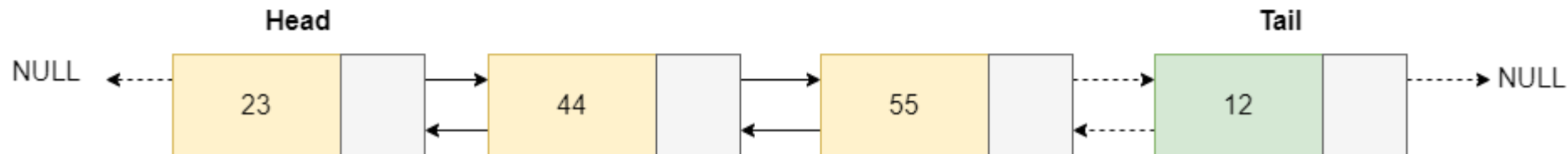
Head



O(1)



```
Tail.Next = newNode;  
newNode.Prev = Tail;  
Tail = newNode;
```



- Sol ve sağ işaretçileri NULL olan yeni düğüm oluşturulur.
- Kuyruğun ileri işaretçisi (**Tail.Next = newNode**) yeni düğümü işaret eder.
- Yeni düğümün önceki işaretçisi (**newNode.Prev = Tail**) kuyruğu işaret eder.
- Kuyruk yeni düğümü işaret (**Tail = newNode**) eder.

Doubly Linked Lists

Araya Ekleme (Inserting at the given position)

Referans Düğüm

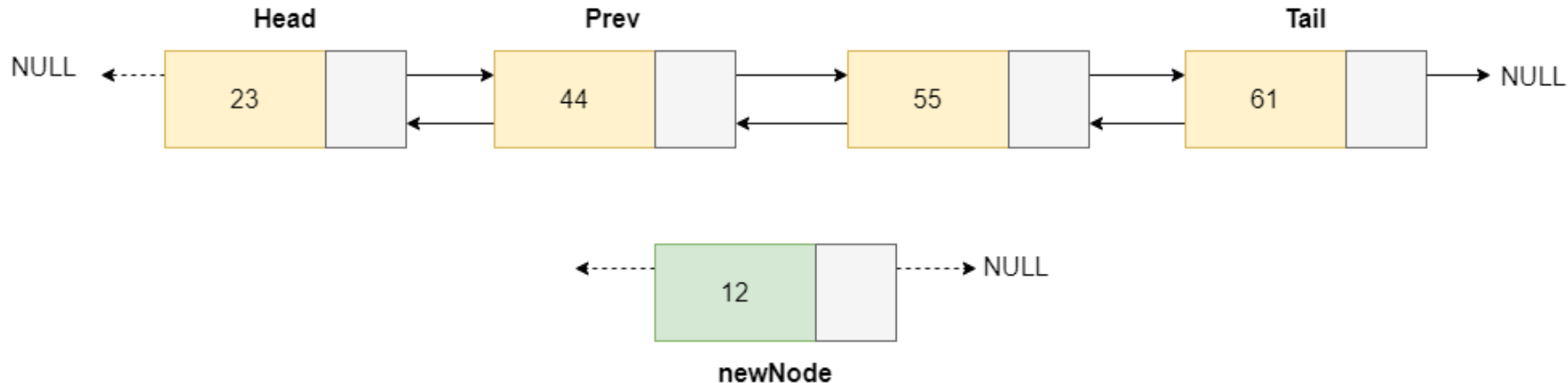
Head



O(n)



```
newNode.Next = prev.Next;  
newNode.Prev = prev;  
prev.Next = newNode;  
newNode.Next.Prev = newNode;
```



- Öncesi (**Prev**) ve Sonrası (**Next**) işaretçileri **NULL** olan bir düğüm (**newNode**) oluşturulur.

Doubly Linked Lists

Araya Ekleme (Inserting at the given position)

Referans Düğüm

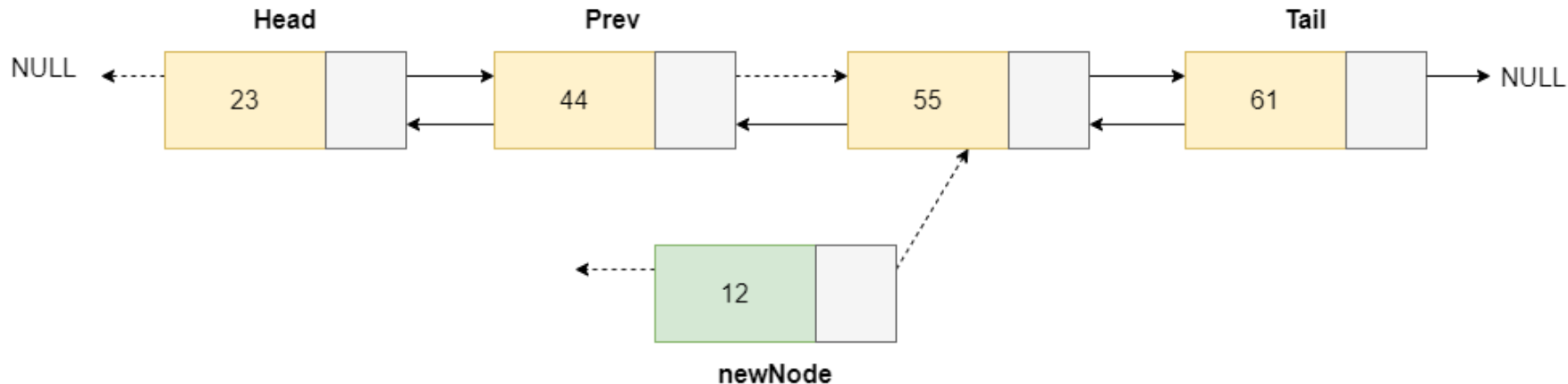
Head



O(n)



```
newNode.Next = prev.Next;  
newNode.Prev = prev;  
prev.Next = newNode;  
newNode.Next.Prev = newNode;
```



- Eklenmek istenen pozisyondan önceki düğüm (Prev) bulunur.
- Yeni düğümün sonraki işaretçisi (**newNode.Next = prev.Next**) önceki düğümün ileri işaretçisi ile güncellenir.

Doubly Linked Lists

Araya Ekleme (Inserting at the given position)

Referans Düğüm

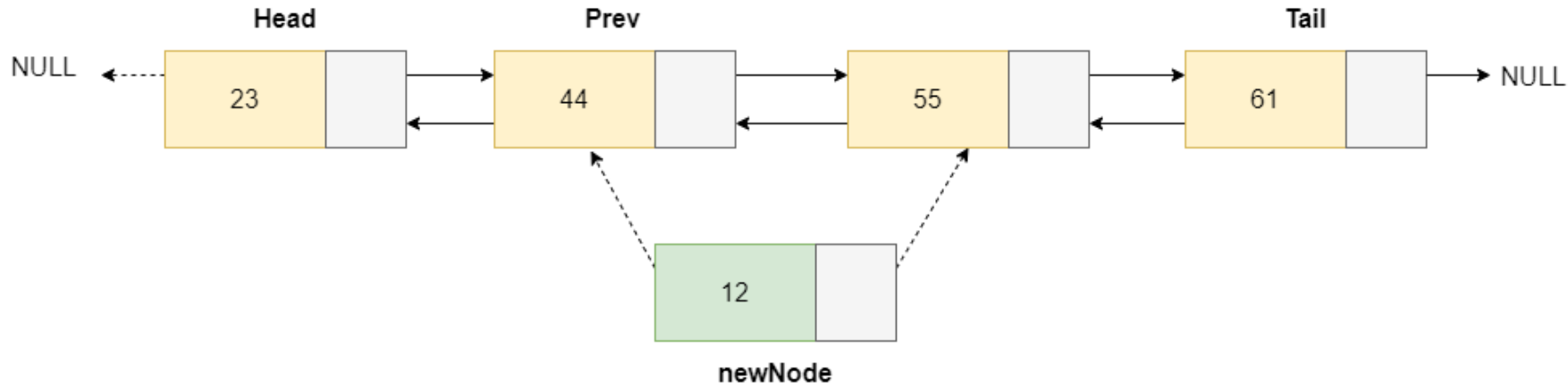
Head



O(n)



```
newNode.Next = prev.Next;  
newNode.Prev = prev;  
prev.Next = newNode;  
newNode.Next.Prev = newNode;
```



- Yeni düğümün önceki işaretçisi; önceki olarak işaretlenen düğümü gösterecek şekilde (**newNode.Prev = prev**) güncellenir.

Doubly Linked Lists

Araya Ekleme (Inserting at the given position)

Referans Düğüm

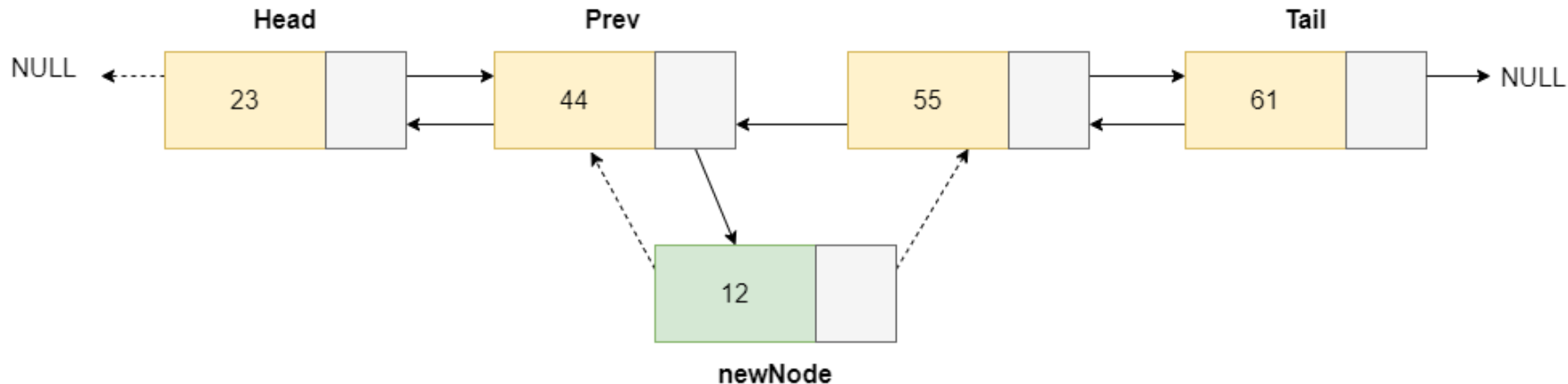
Head



O(n)



```
newNode.Next = prev.Next;  
newNode.Prev = prev;  
prev.Next = newNode;  
newNode.Next.Prev = newNode;
```



- Önceki düğümün ileri işaretçisi yeni düğümü işaret edecek şekilde güncellenir (**prev.Next = newNode**).

Doubly Linked Lists

Araya Ekleme (Inserting at the given position)

Referans Düğüm

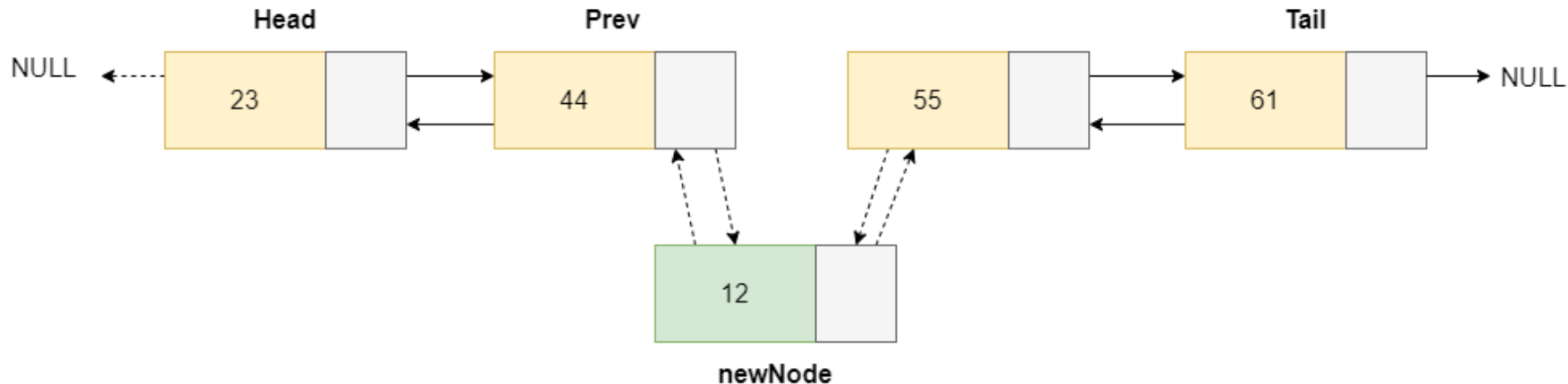
Head



O(n)



```
newNode.Next = prev.Next;  
newNode.Prev = prev;  
prev.Next = newNode;  
newNode.Next.Prev = newNode;
```



- Yeni düğümden sonra gelen düğümün önceki işaretçisi yeni düğümü işaret edecek şekilde güncellenir (**newNode.Next.Prev = newNode**).

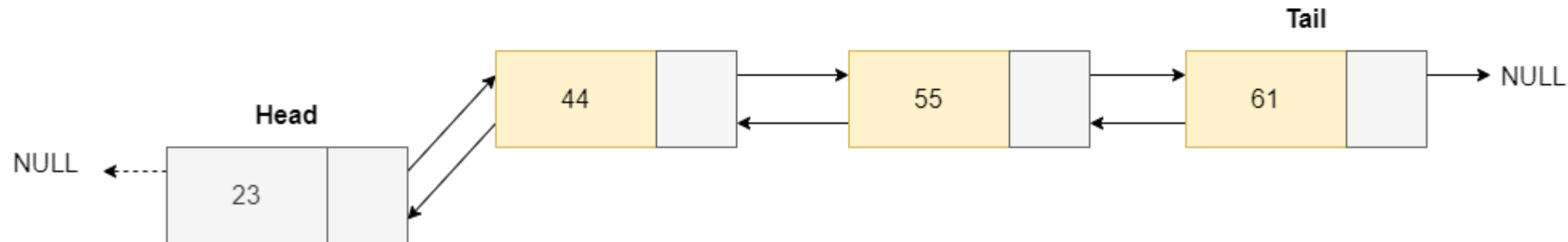
Doubly Linked Lists

İlk Düğümü Silme (Deleting the first node)

Referans Düğüm



```
Head = Head.Next;  
Head.Prev = null;
```



- Referans düğüm güncellenir (**Head = Head.Next**).

Doubly Linked Lists

İlk Düğümü Silme (Deleting the first node)

Referans Düğüm

Head



O(1)



```
Head = Head.Next;  
Head.Prev = null;
```



- Yeni referans düğümün önceki işaretçisi güncellenir (**Head.Prev = null**).

Doubly Linked Lists

Son Düğümü Silme (Deleting at the last node)

Referans Düğüm

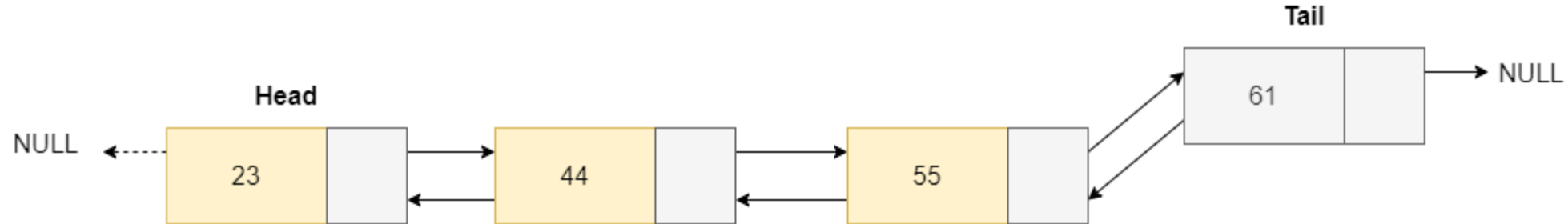
Head



O(1)

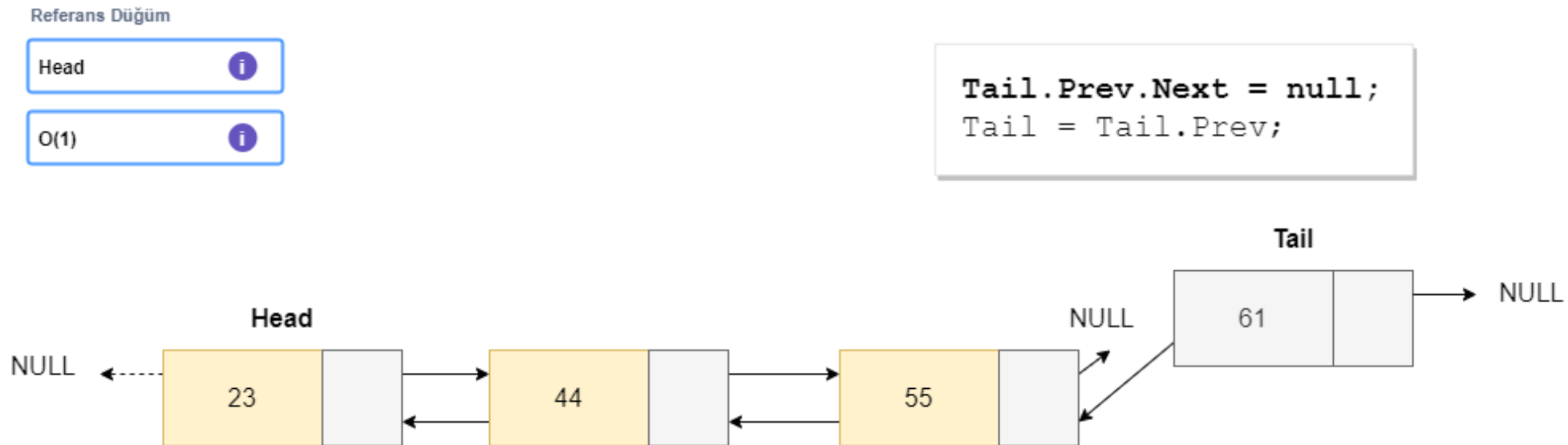


```
Tail.Prev.Next = null;  
Tail = Tail.Prev;
```



Doubly Linked Lists

Son Düğümü Silme (Deleting at the last node)



- Kuyruğun öncesindeki düğümün sonraki işaretçisi **NULL** olarak güncellenir.

Doubly Linked Lists

Son Düğümü Silme (Deleting at the last node)

Referans Düğüm



```
Tail.Prev.Next = null;  
Tail = Tail.Prev;
```



- Kuyruk güncellenir.

Doubly Linked Lists

Ara Düğümü Silme (Deleting an intermediate node in the doubly linked list)

Referans Düğüm

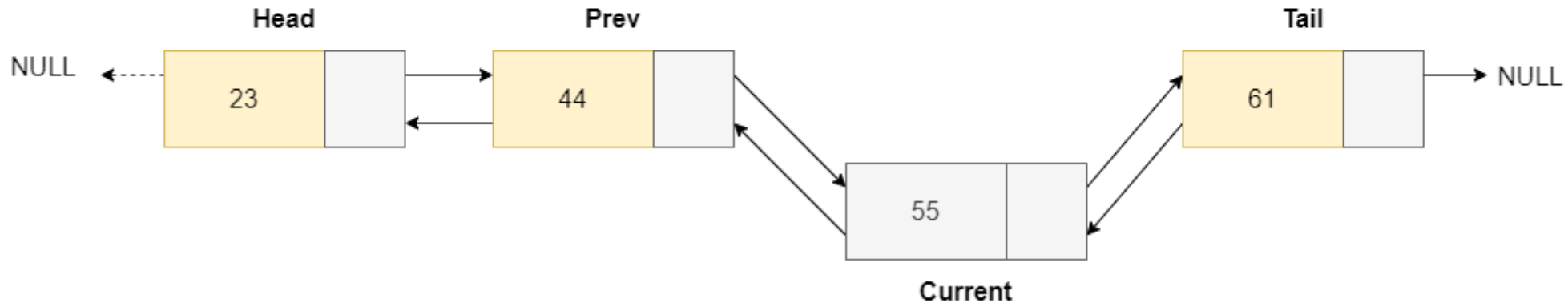
Head



O(n)



```
prev.Next = current.Next;  
current.Next.Prev = current.Prev;  
current = null;
```



- Silinecek düğüme kadar gidilir.

Doubly Linked Lists

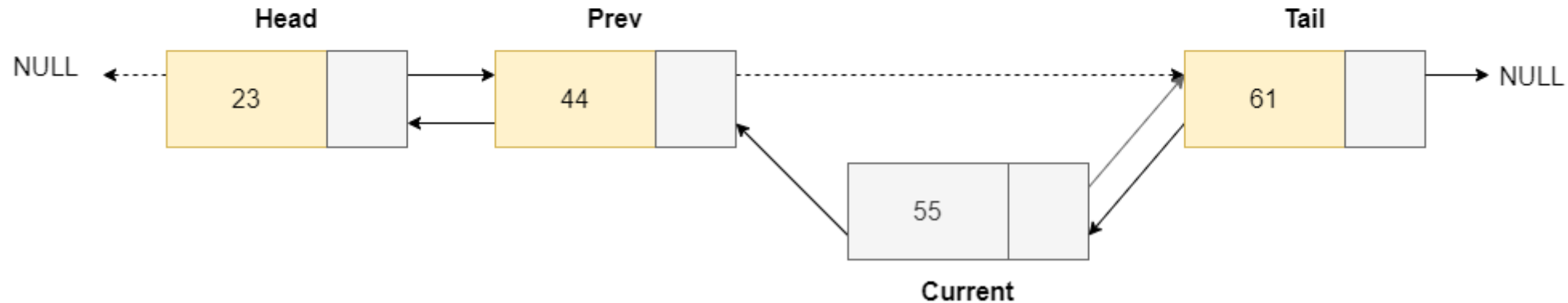
Ara Düğümü Silme (Deleting an intermediate node in the doubly linked list)

Referans Düğüm

Head 

O(n) 

```
prev.Next = current.Next;  
current.Next.Prev = current.Prev;  
current = null;
```



- Silinecek düğümün ileri işaretçisinin gösterdiği adres; önceki düğümün ileri işaretçisine atanır.

Doubly Linked Lists

Ara Düğümü Silme

(Deleting an intermediate node in the doubly linked list)

Referans Düğüm

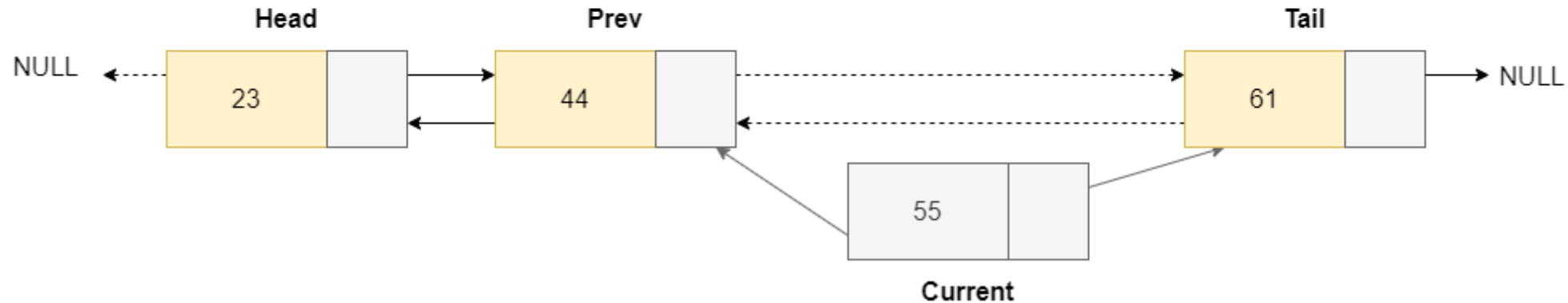
Head



O(n)



```
prev.Next = current.Next;  
current.Next.Prev = current.Prev;  
current = null;
```



- Silinecek düğümden sonra gelen düğümün önceki işaretçisi; silinecek düğümün önceki işaretçisi ile güncellenir.

Doubly Linked Lists

Ara Düğümü Silme

(Deleting an intermediate node in the doubly linked list)

Referans Düğüm

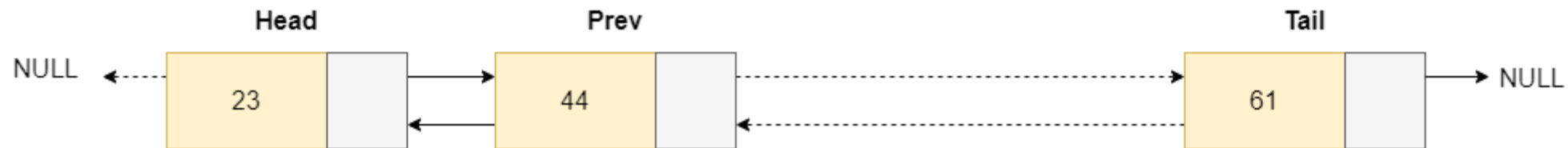
Head



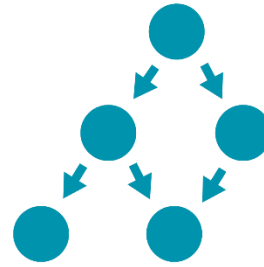
O(n)



```
prev.Next = current.Next;  
current.Next.Prev = current.Prev;  
current = null;
```



- Silinecek düğüm kaldırılır.



Veri Yapıları ve Algoritmalar

ZAFER CÖMERT

Öğretim Üyesi