Data Structures and Object Oriented Programming

Lecture 9

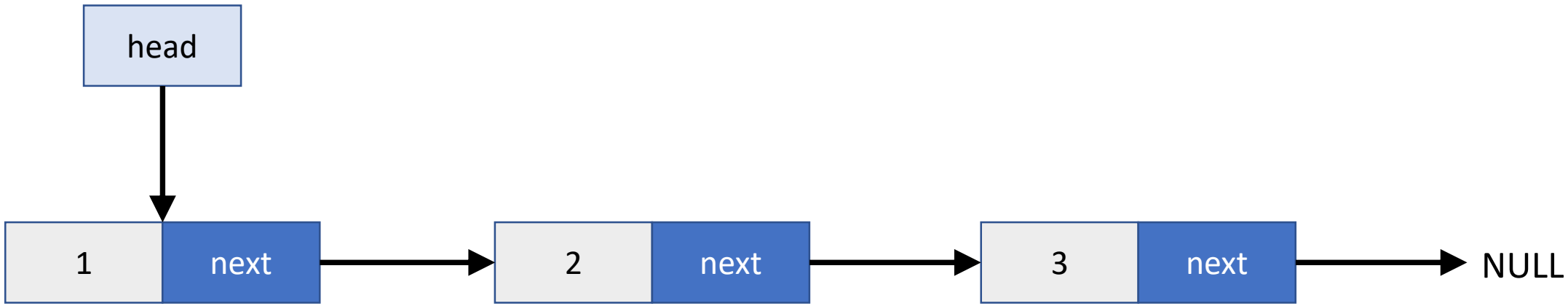
Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

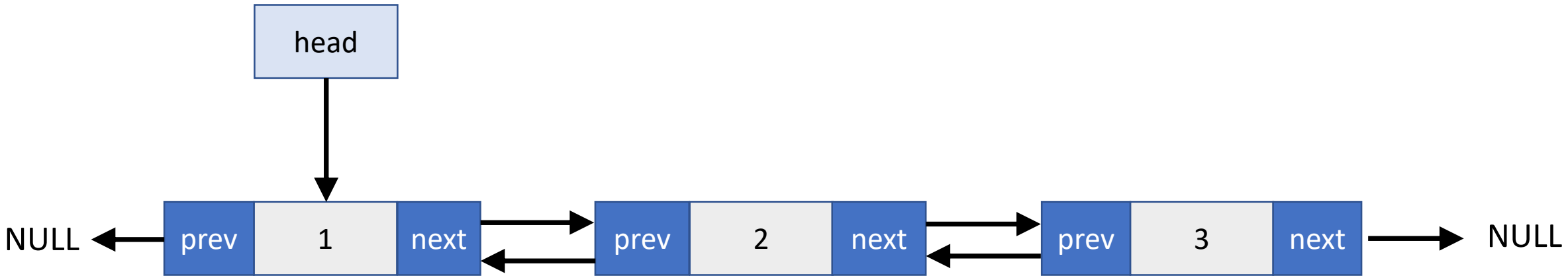
Object-Oriented Programming in C++

Doubly Linked List

Linked List



Doubly Linked List



Doubly Linked List

```
struct Node {
    int data;
    Node* next = NULL;
    Node* previous = NULL;

};
```

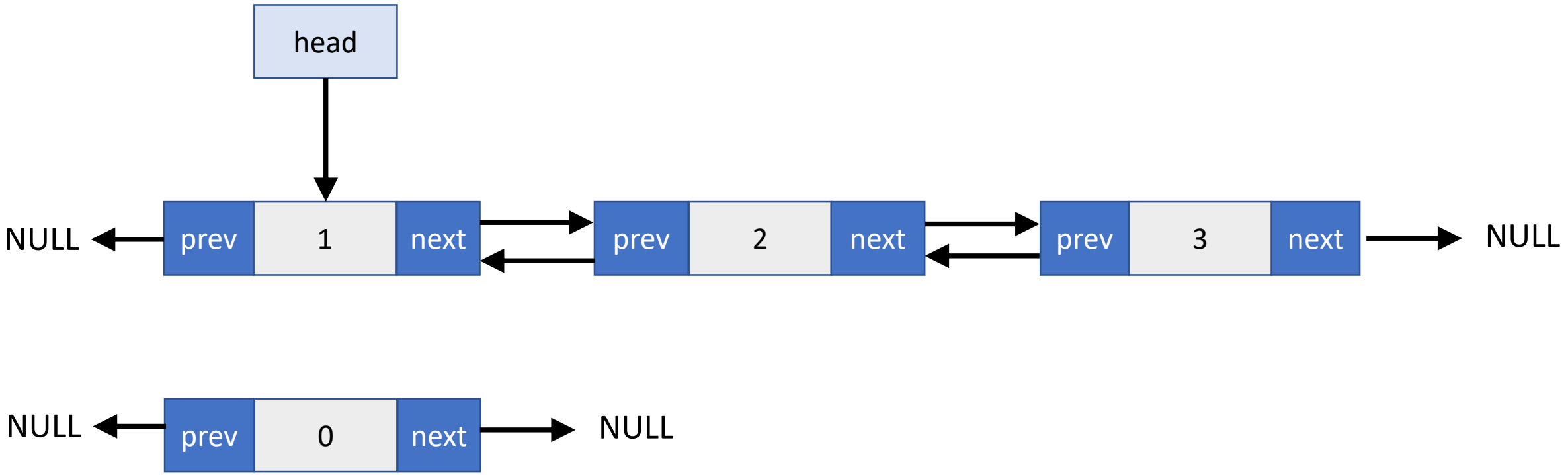
```
int main()
{
    return 0;
}
```

Doubly Linked List

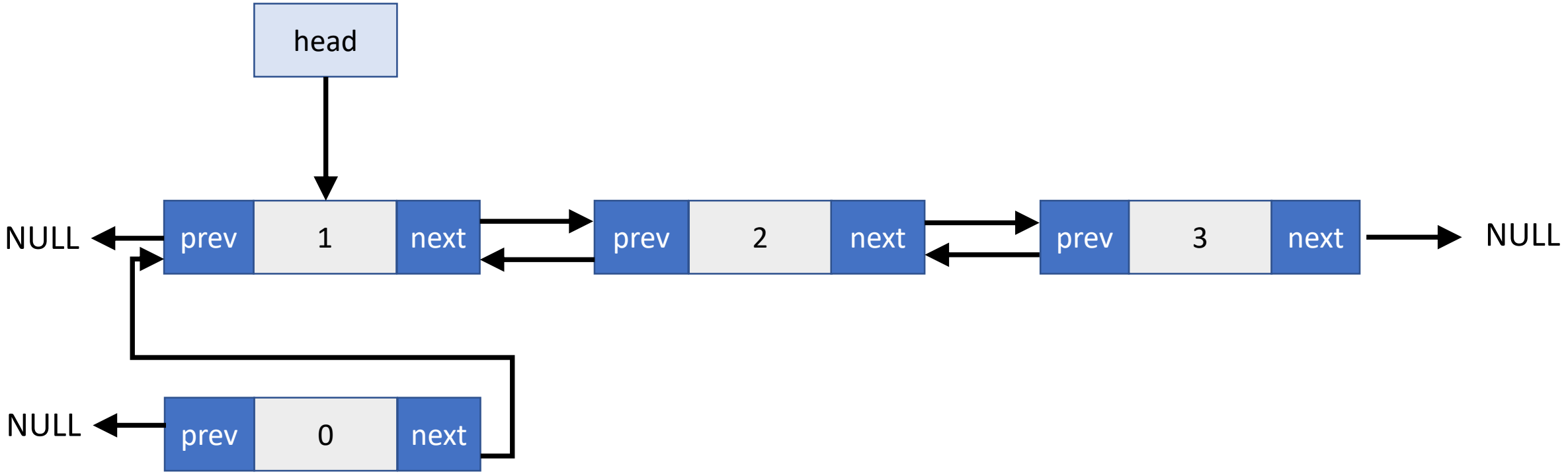
```
struct Node {
    int data;
    Node* next = NULL;
    Node* previous = NULL;

};
class LinkedList {

    Node *head=NULL;

public:
    void printList();
    void insert_start(int value);
    void insert_end(int value);
    void insert_after(int n,int value);
    void delete_start();
    void delete_end();
    void delete_after(int n);

};
```

```
int main()
{
    return 0;
}
```

Doubly Linked List (`insert_start`)

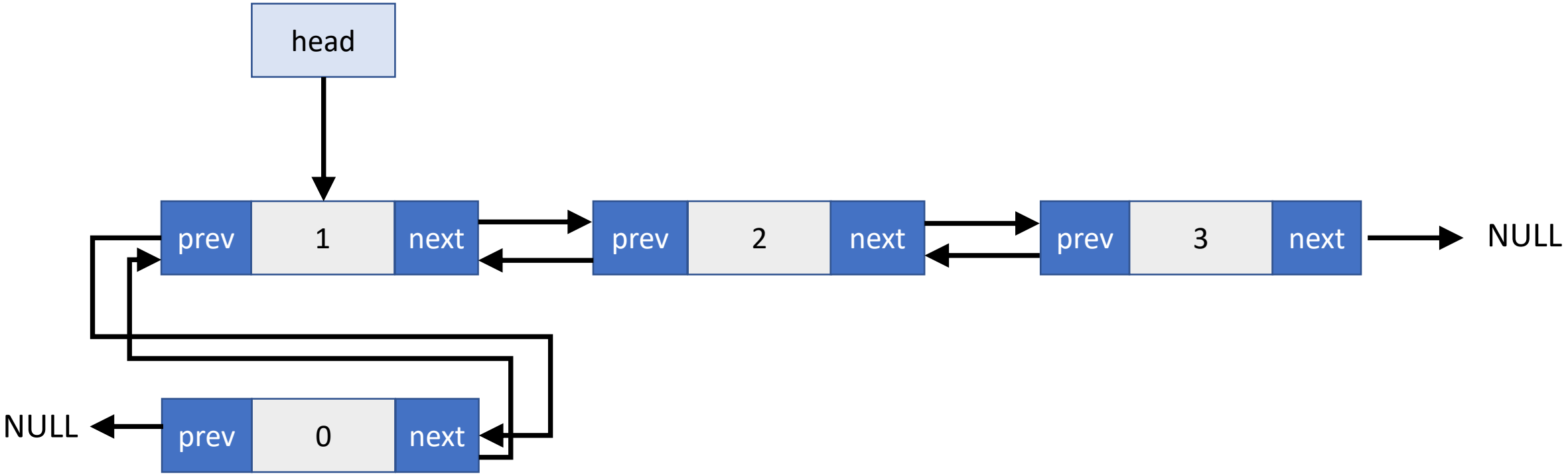


Doubly Linked List (`insert_start`)



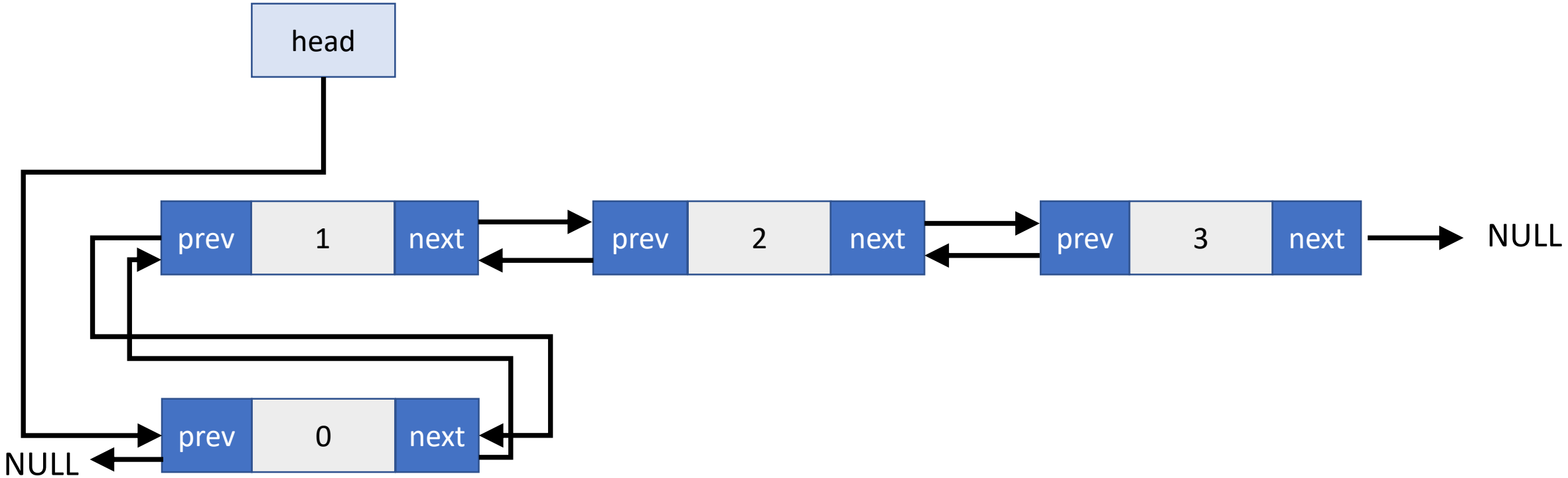
temp->next=head;

Doubly Linked List (`insert_start`)



head->previous=temp;

Doubly Linked List (`insert_start`)



head=temp;

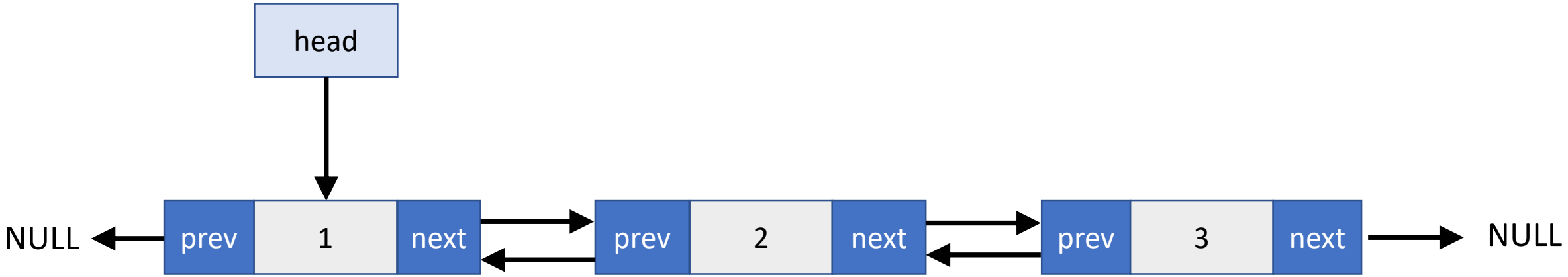
Doubly Linked List (`insert_start`)

```
void LinkedList::insert_start(int value)
{
    Node* temp = new Node;
    temp->data = value;
    temp->next = head;
    head->previous = temp;
    head = temp;

}
```

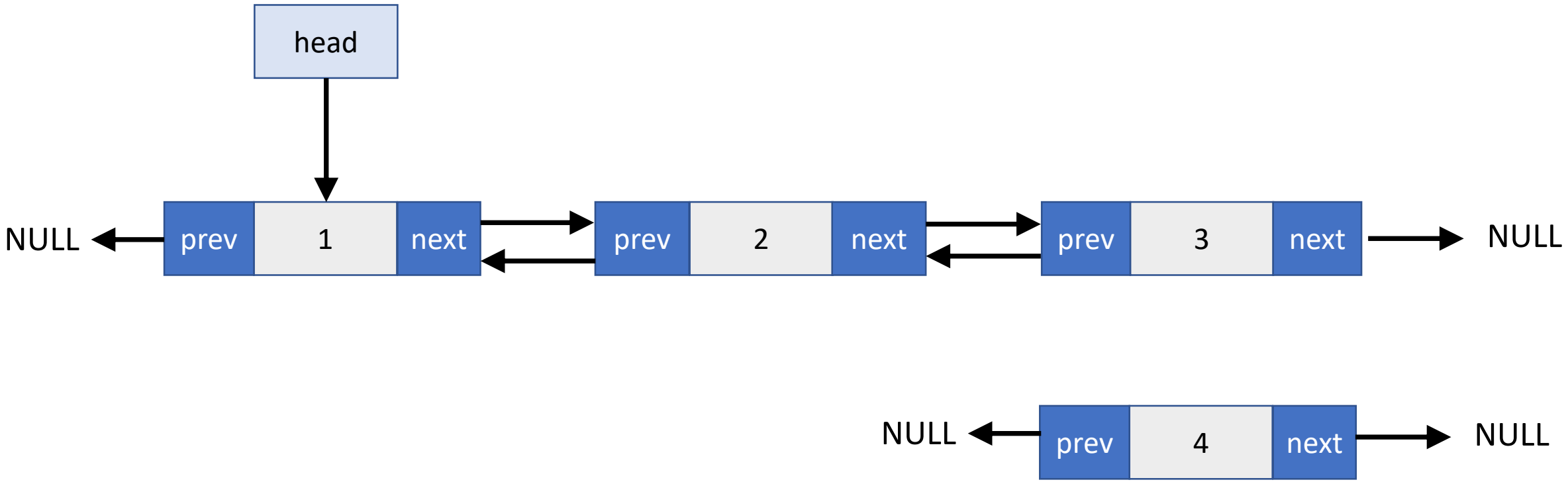
Doubly Linked List (**insert_start**)

```
void LinkedList::insert_start(int value)
{

    Node* temp = new Node;
    temp->data = value;

    if (head == NULL)
    {
        head = temp;
    }

    else
    {
        temp->next = head;
        head->previous = temp;
        head = temp;
    }

}
```

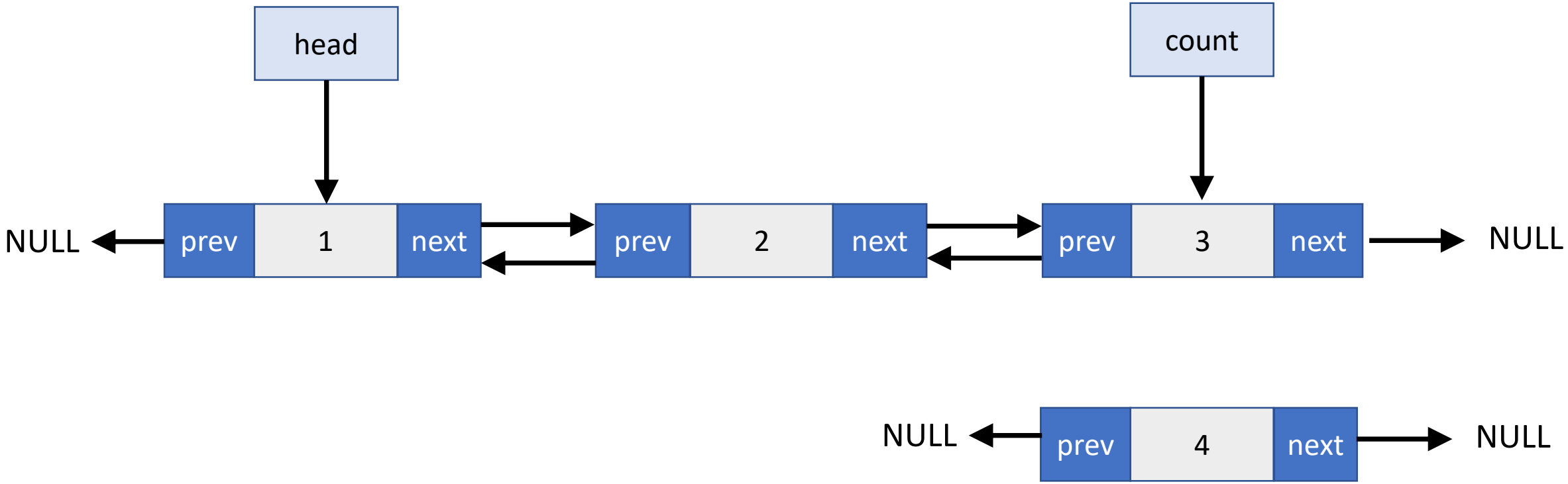
Doubly Linked List (`insert_end`)



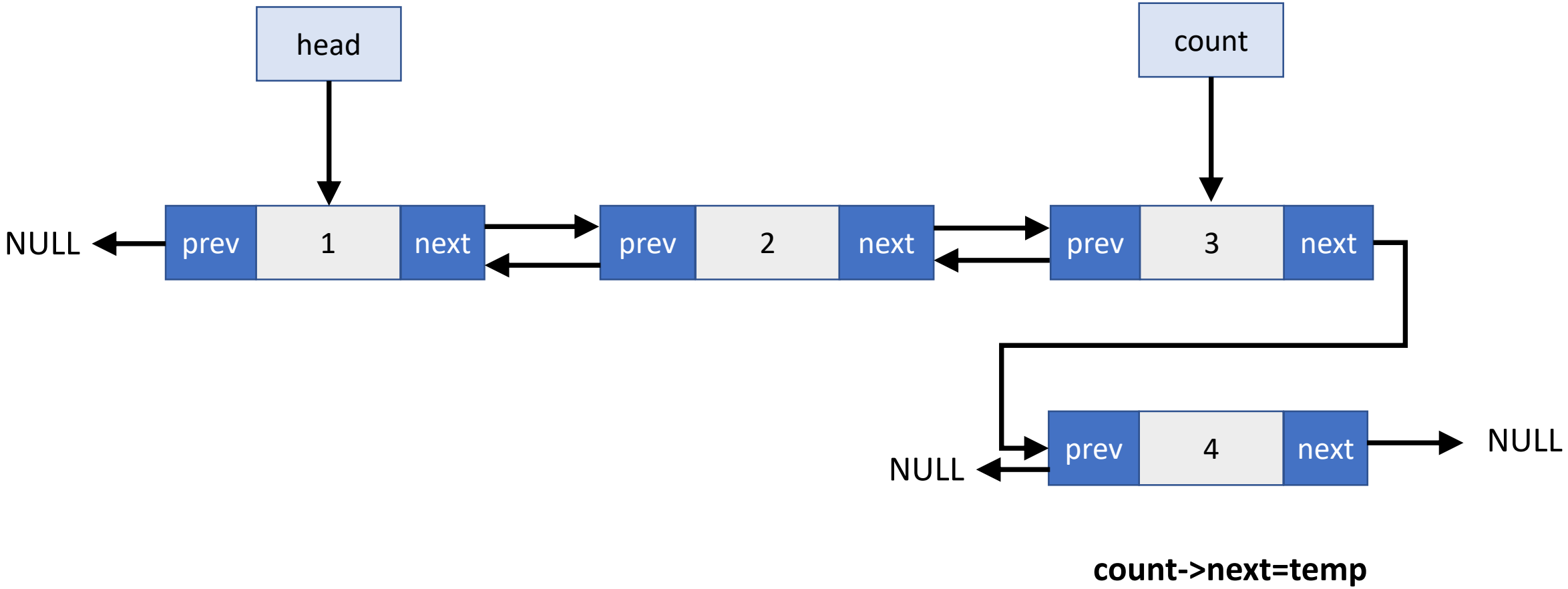
Doubly Linked List (`insert_end`)



Doubly Linked List (`insert_end`)

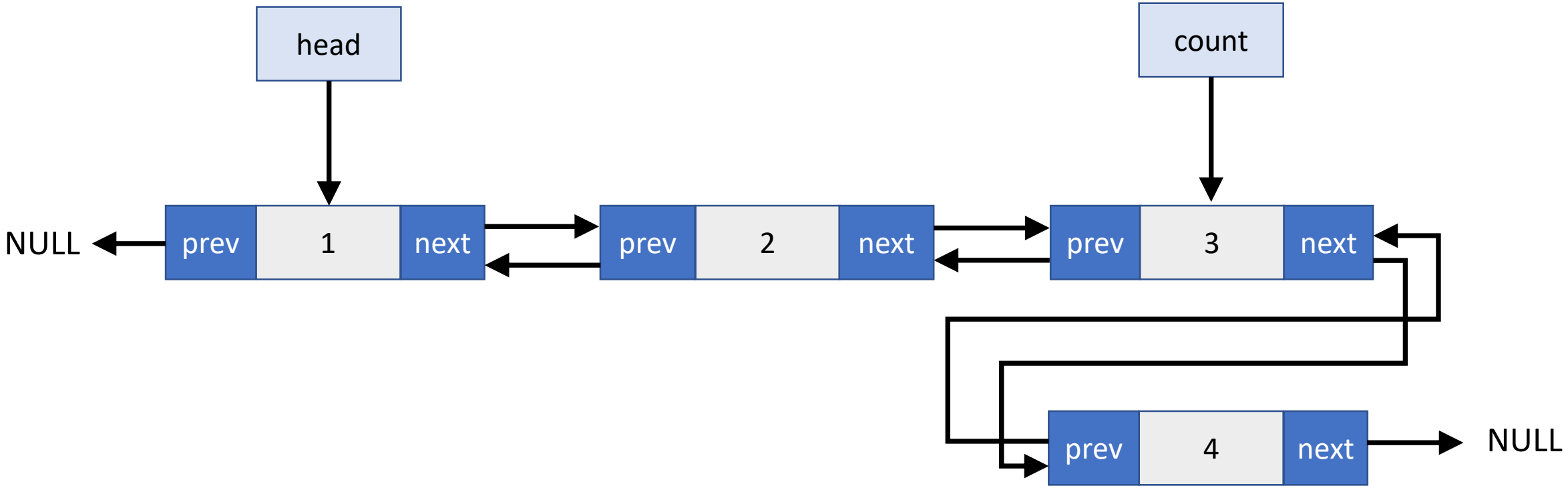


Doubly Linked List (`insert_end`)



count->next=temp

Doubly Linked List (`insert_end`)

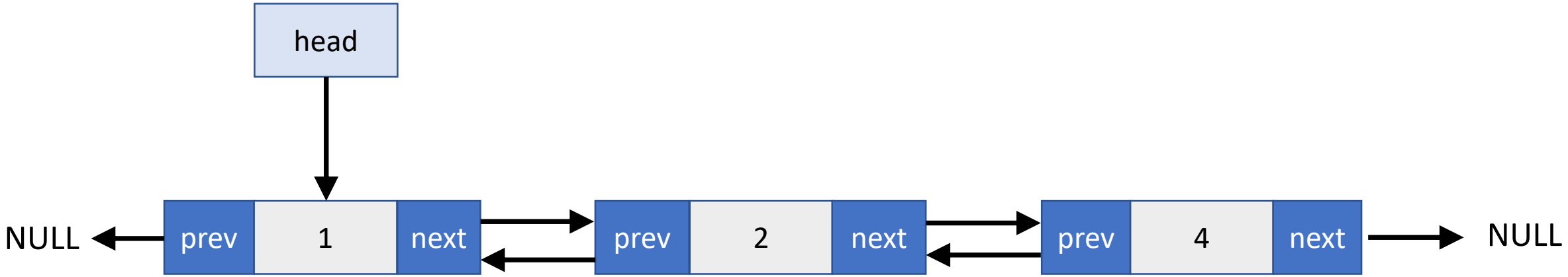


temp->previous=count

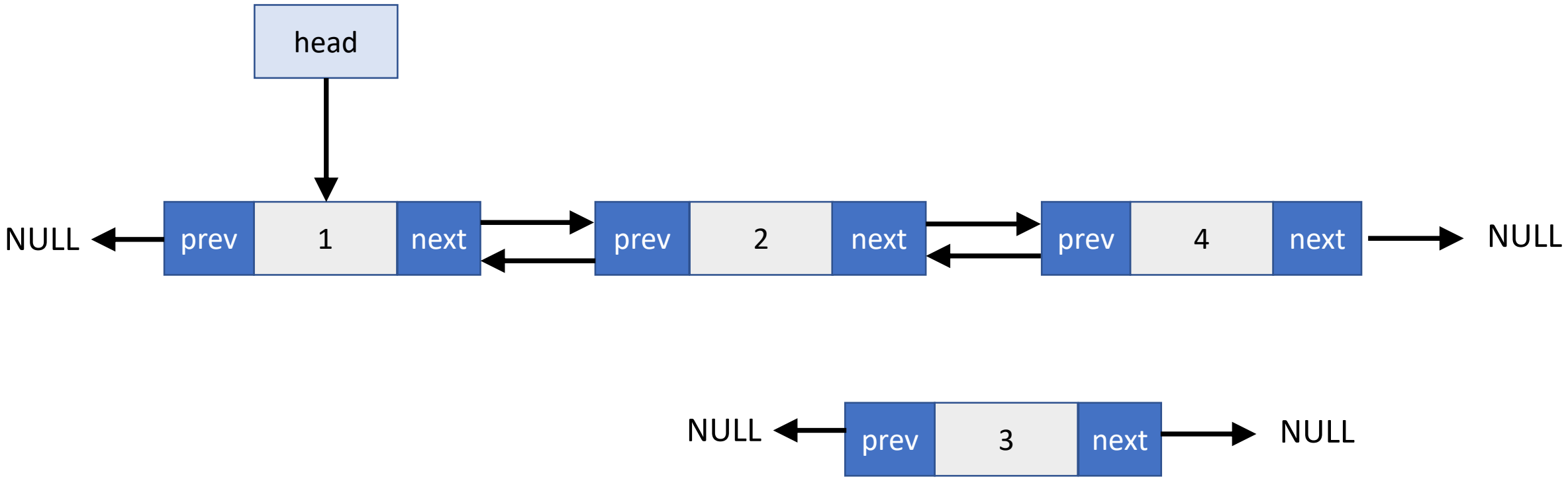
Linked List (insert_end)

```
void LinkedList::insert_end(int value)
{
    Node* temp = new Node;
    temp->data = value;

    if (head == NULL)
    {
        head = temp;
    }

    else
    {
        Node* count = head;
        while (count->next != NULL)
        {
            count = count->next;
        }
        count->next = temp;
        temp->previous = count;
    }
}
```

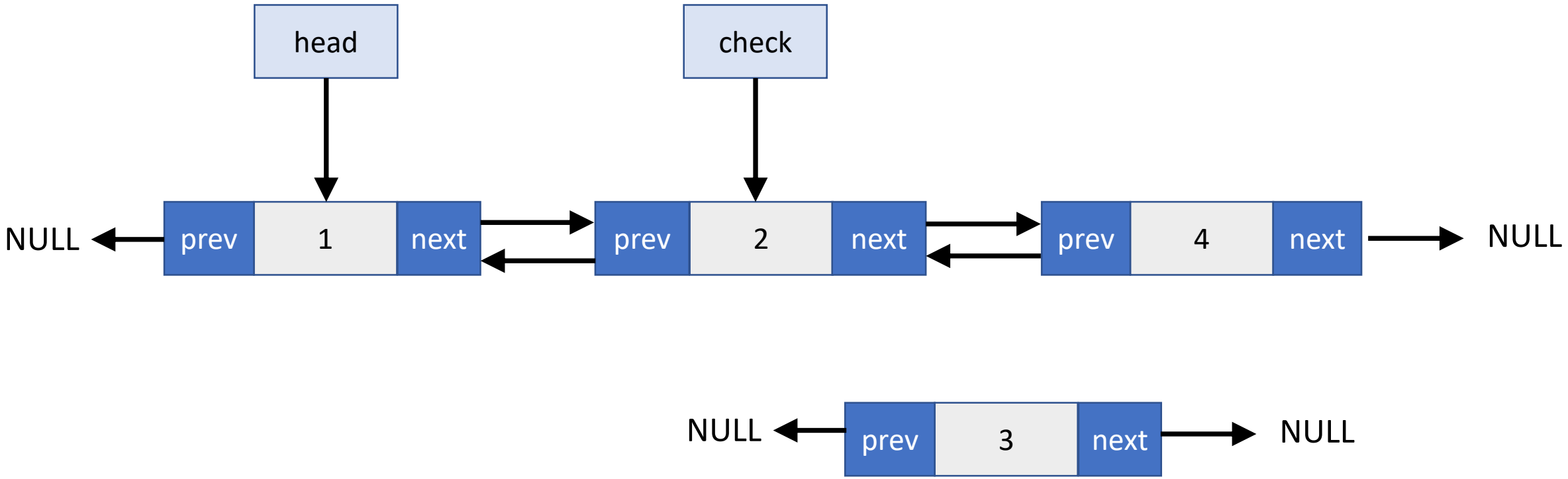
Doubly Linked List (`insert_after`)



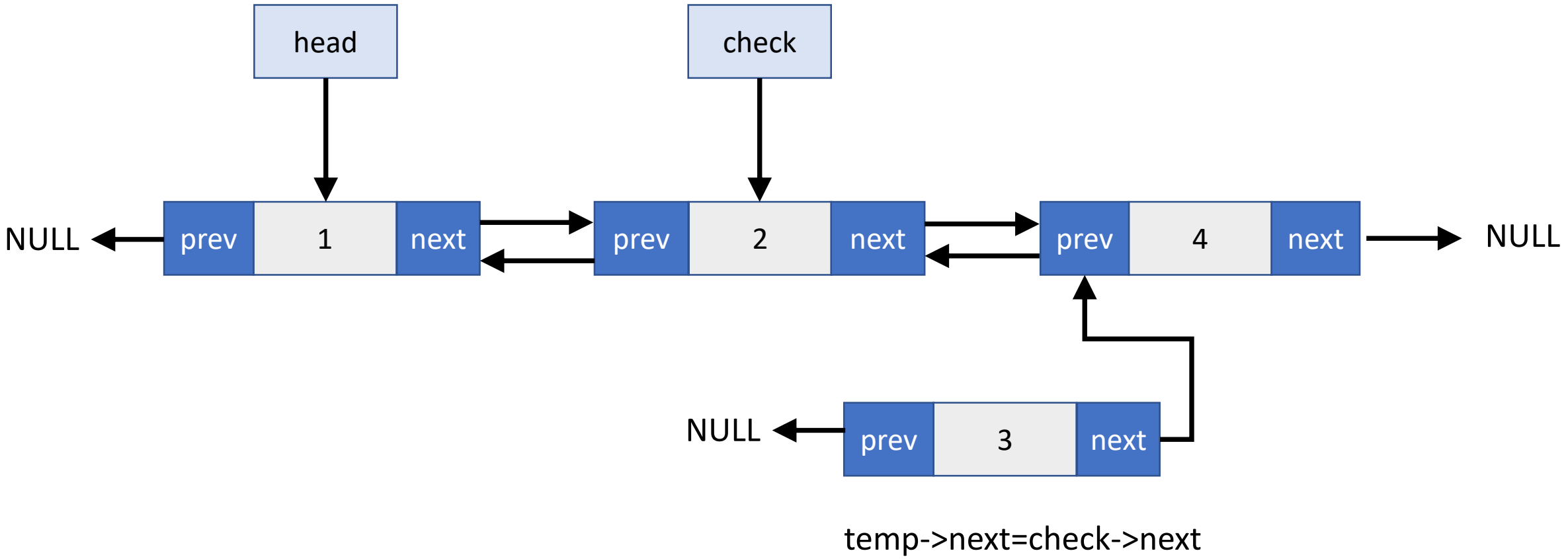
Doubly Linked List (`insert_after`)



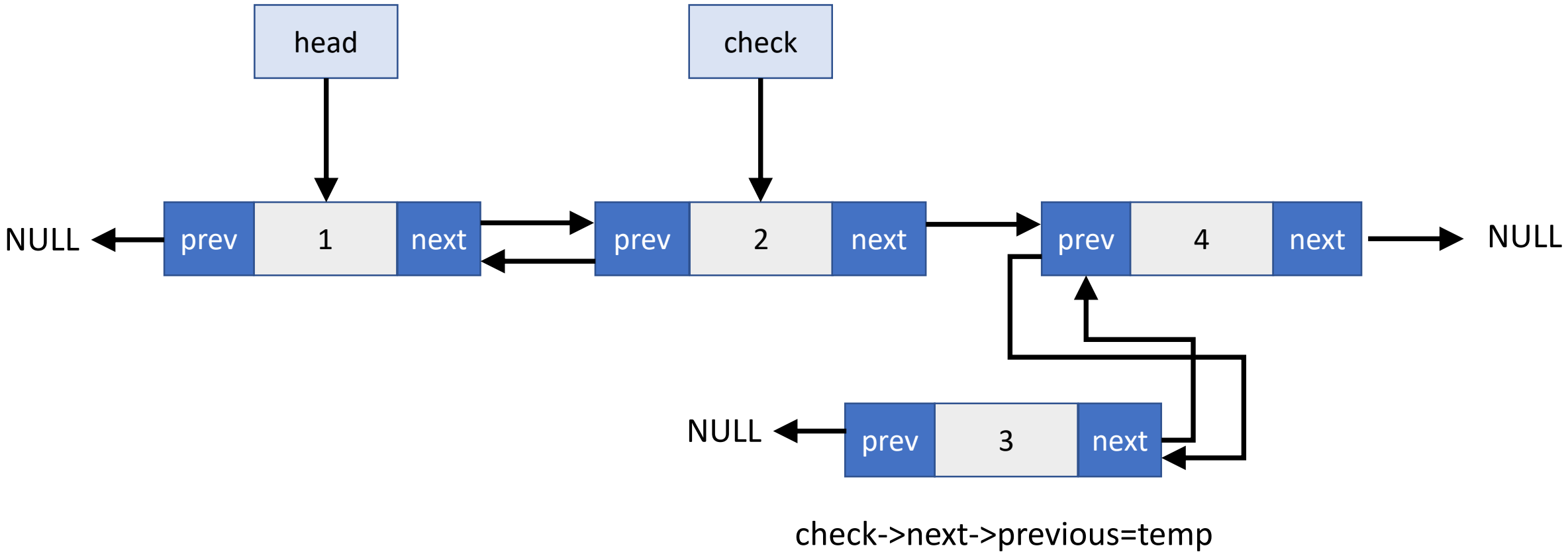
Doubly Linked List (`insert_after`)



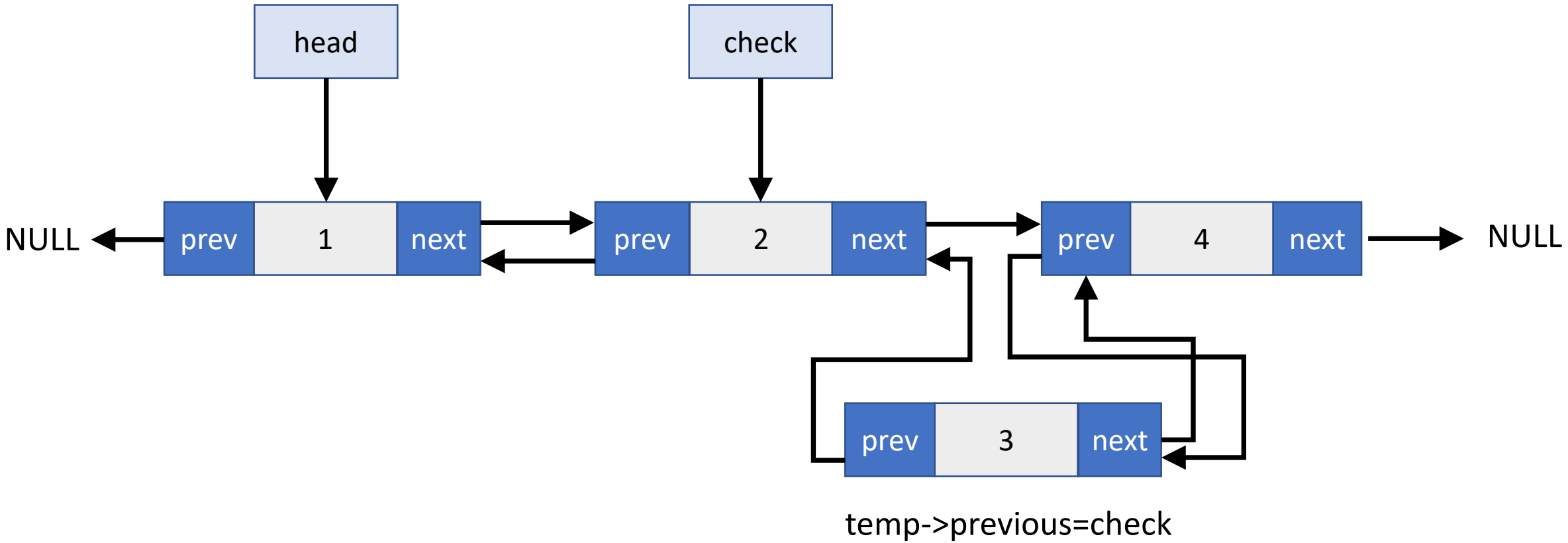
Doubly Linked List (`insert_after`)



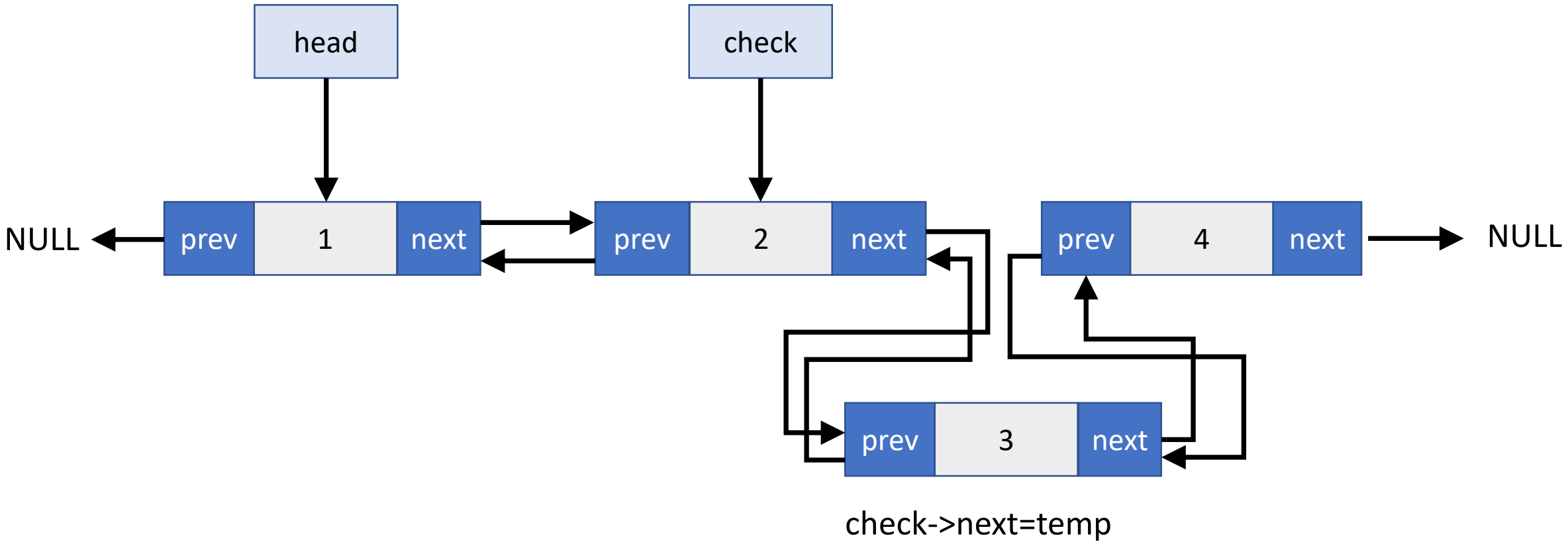
Doubly Linked List (`insert_after`)



Doubly Linked List (`insert_after`)



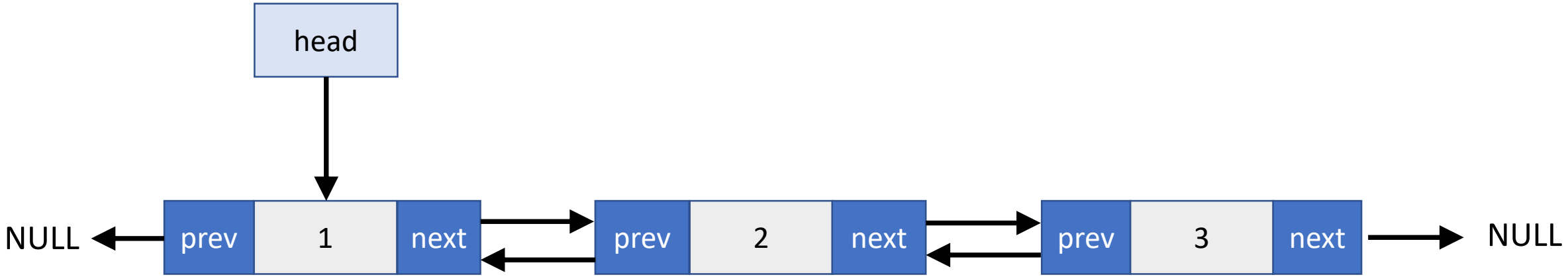
Doubly Linked List (`insert_after`)



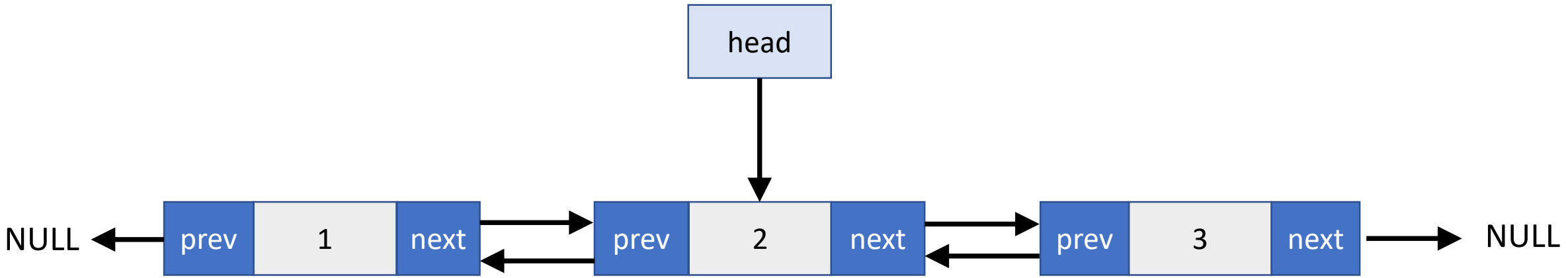
Doubly Linked List (insert_after)

```
void LinkedList::insert_after(int n, int value)
{
    Node* temp = new Node;
    temp->data = value;

    Node* check = head;
    while (check->data != n)
    {
        check = check->next;
        if (check == NULL)
            return;
    }

    temp->next = check->next;
    check->next->previous = temp;
    check->next = temp;
    temp->previous = check;

}
```

Doubly Linked List (`delete_start`)

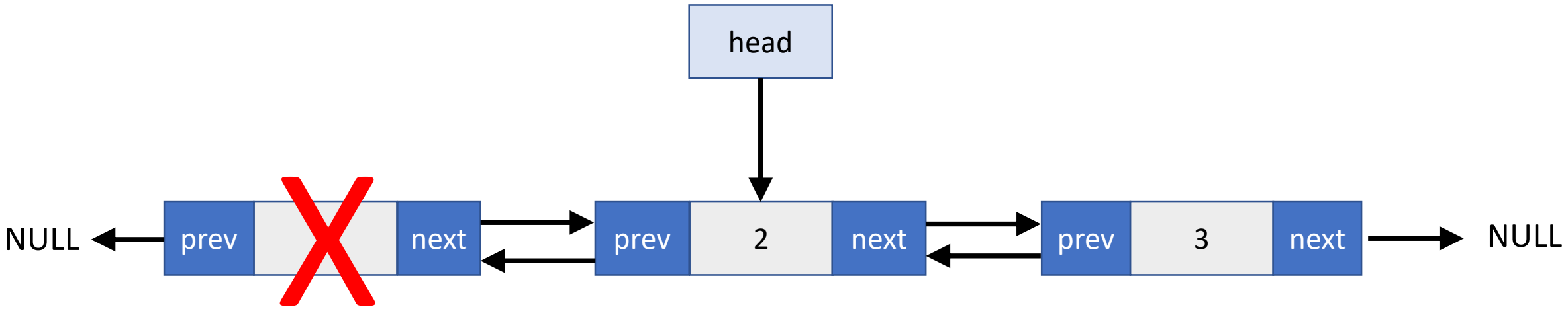


Doubly Linked List (`delete_start`)



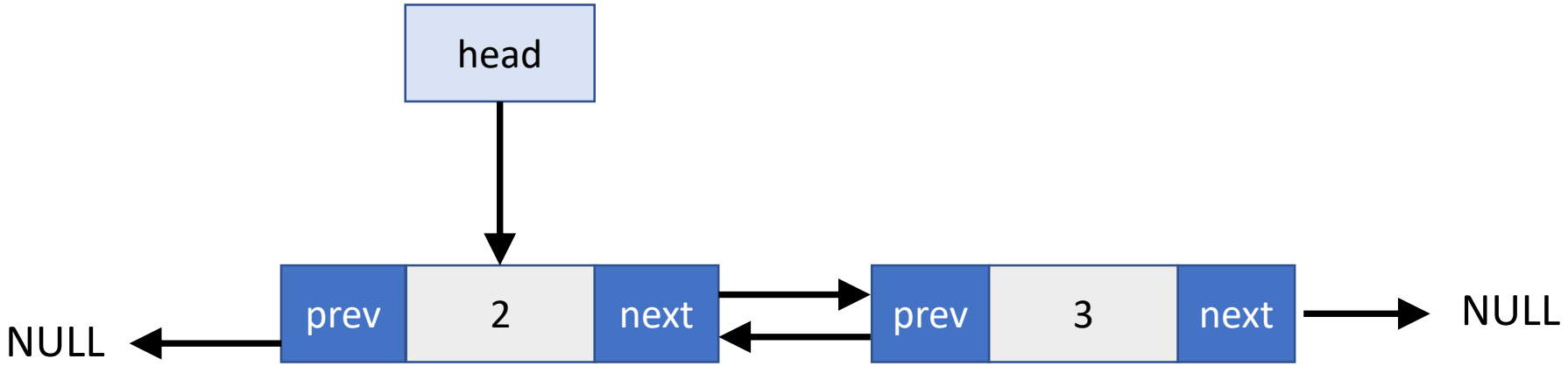
head=head->next

Doubly Linked List (`delete_start`)



delete head->previous

Doubly Linked List (`delete_start`)

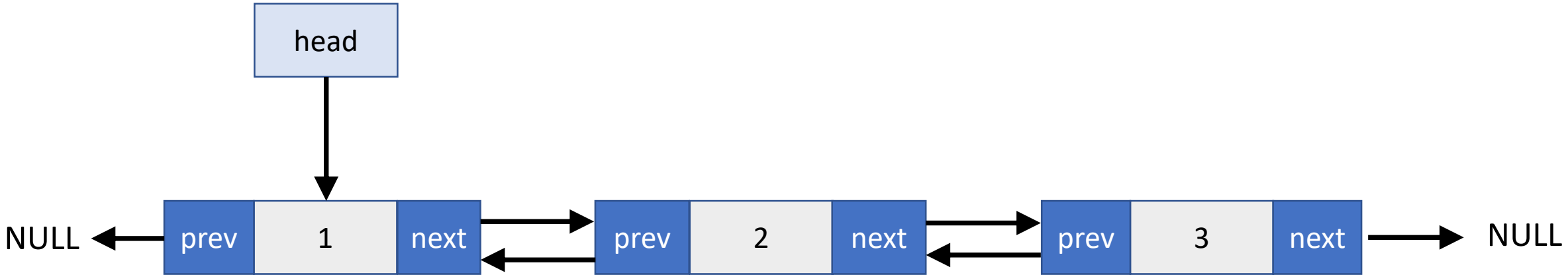


head->previous=NULL;

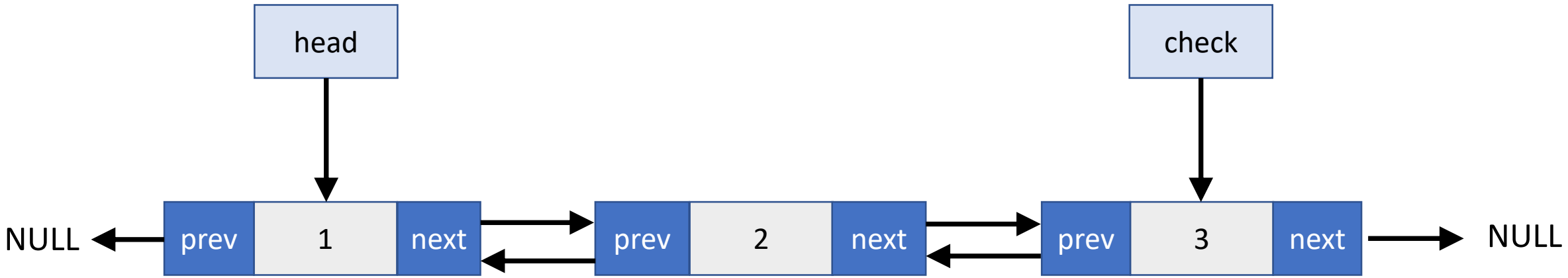
Doubly Linked List (**delete_start**)

```
void LinkedList::delete_start()
{
    if (head == NULL)
    {
        return;
    }

    else
    {
        head=head->next;
        delete head->previous;
        head->previous = NULL;
    }

}
```

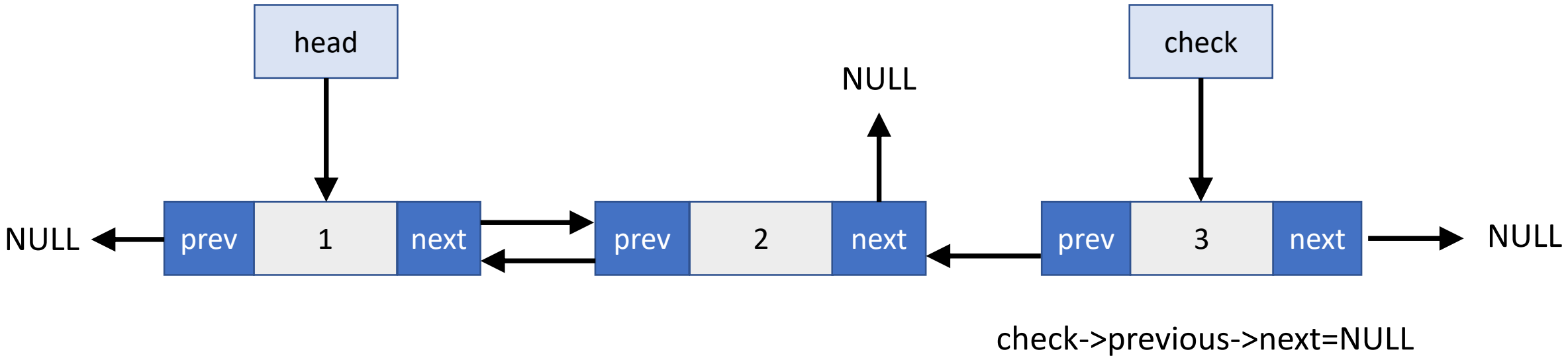
Doubly Linked List (`delete_end`)



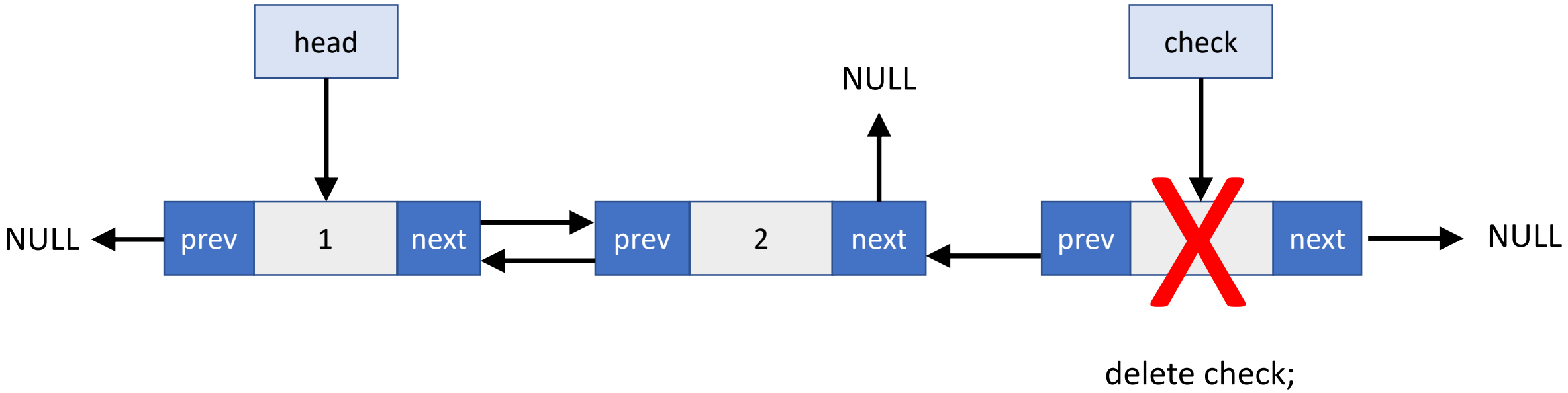
Doubly Linked List (`delete_end`)



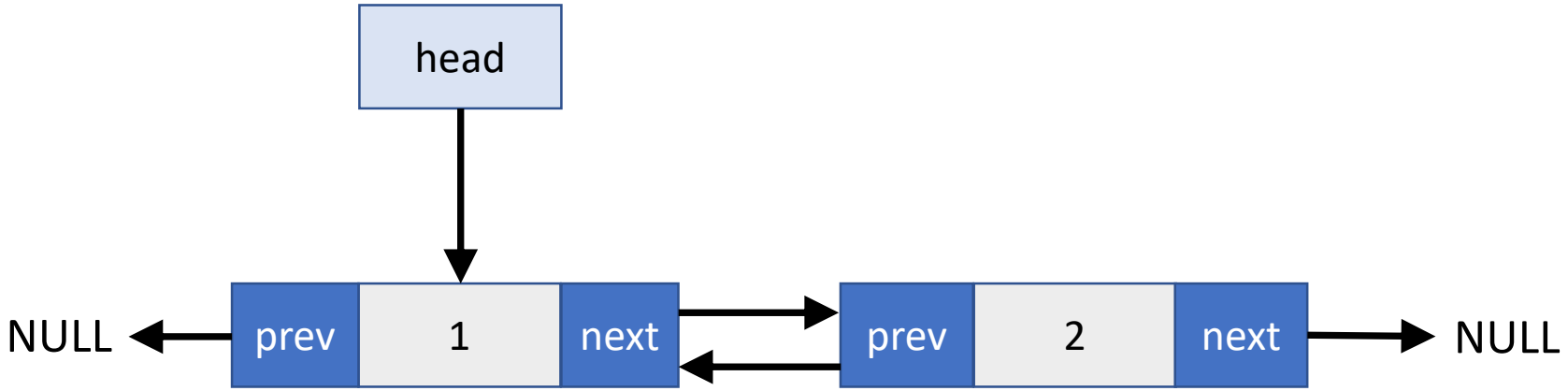
Doubly Linked List (`delete_end`)



Doubly Linked List (`delete_end`)



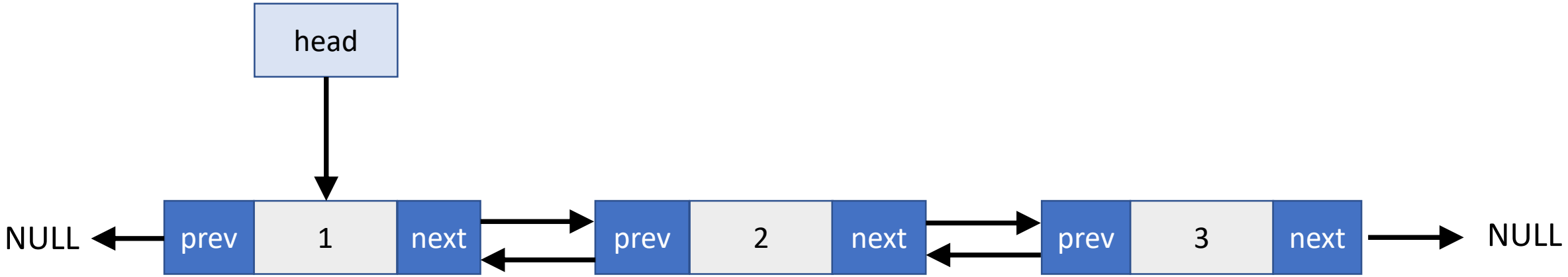
Doubly Linked List (`delete_end`)



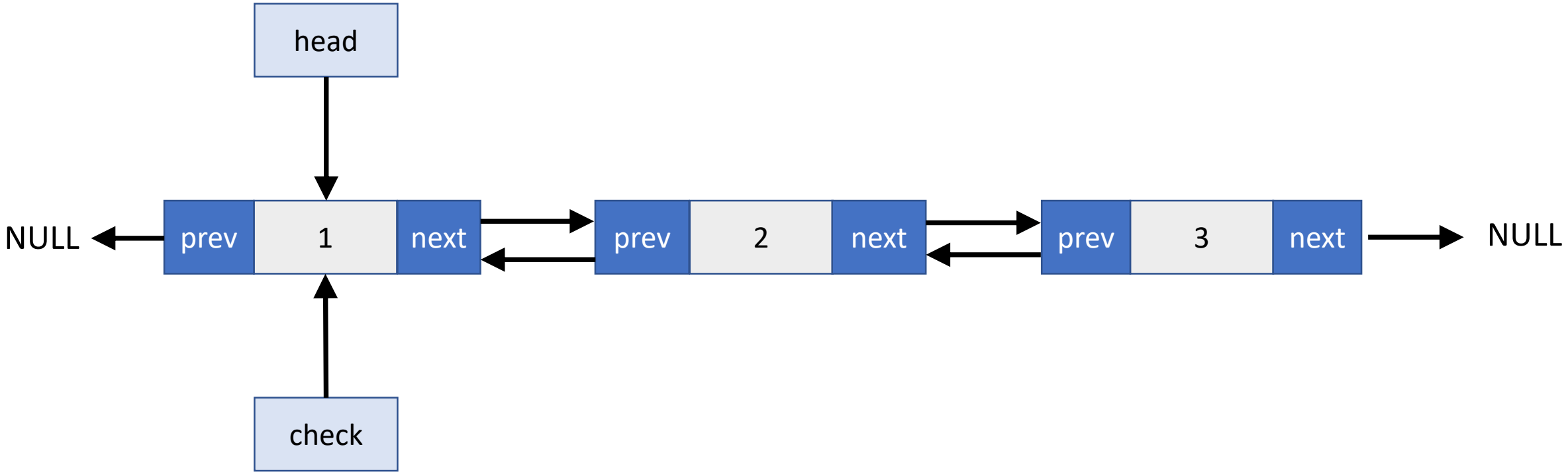
Doubly Linked List (delete_end)

```
void LinkedList::delete_end()
{

    if (head == NULL)
    {
        return;
    }

    else
    {
        Node* check = head->next;
        while (check->next != NULL)
        {
            check = check->next;
        }

        check->previous->next = NULL;
        delete check;
    }
}
```

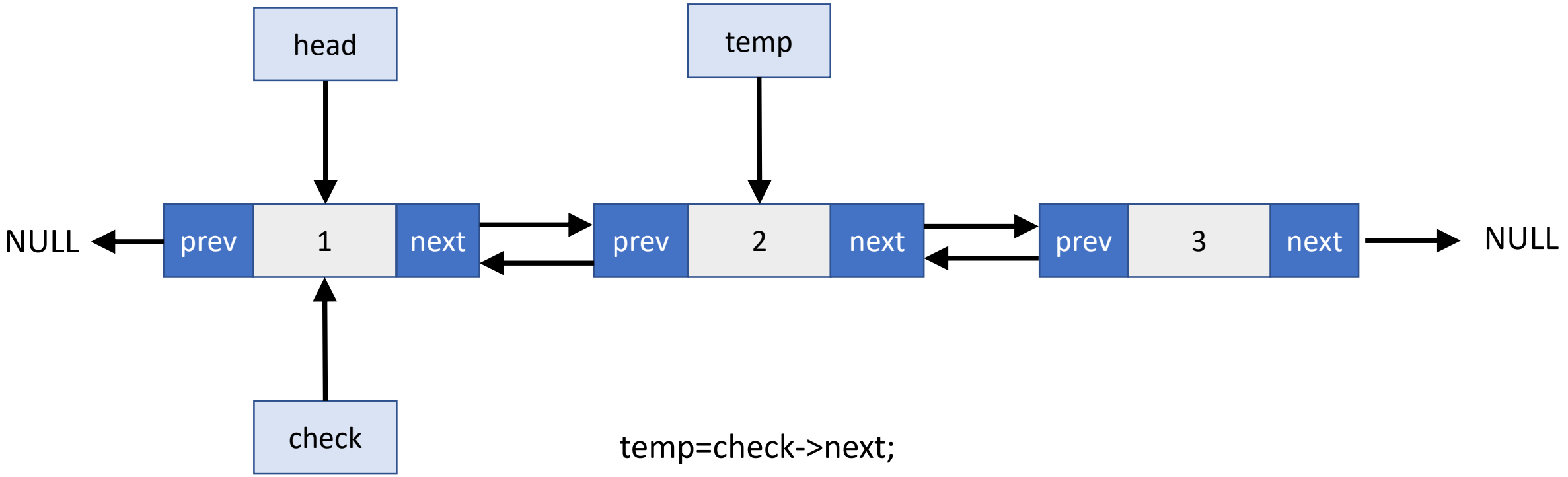
Doubly Linked List (`delete_after`)



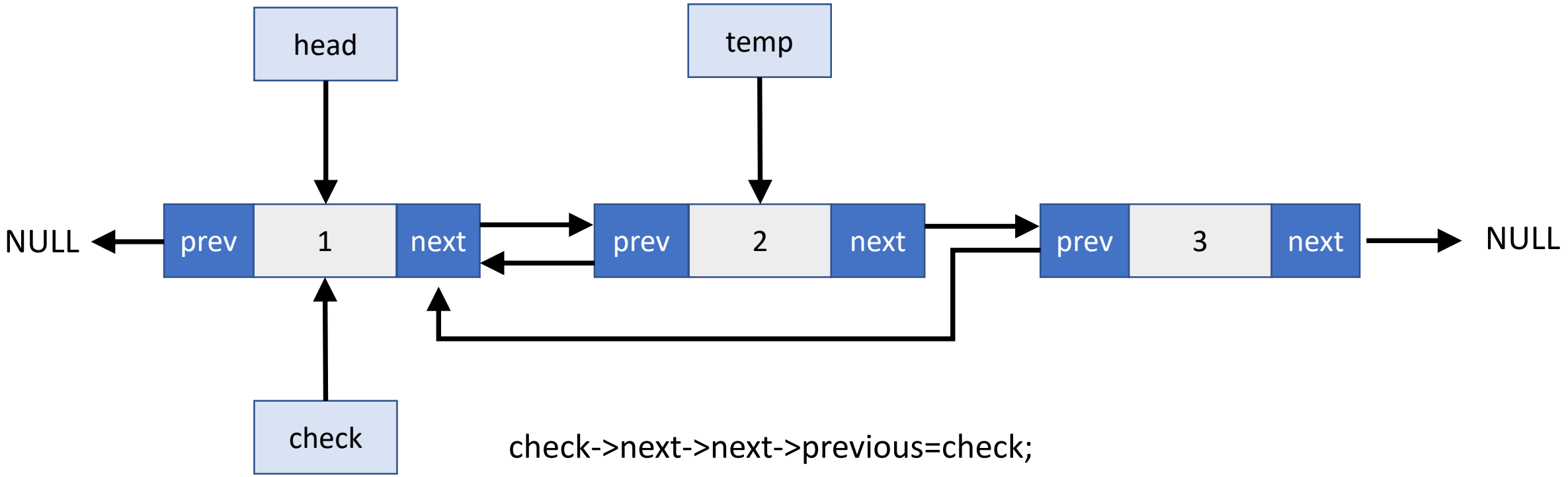
Doubly Linked List (`delete_after`)



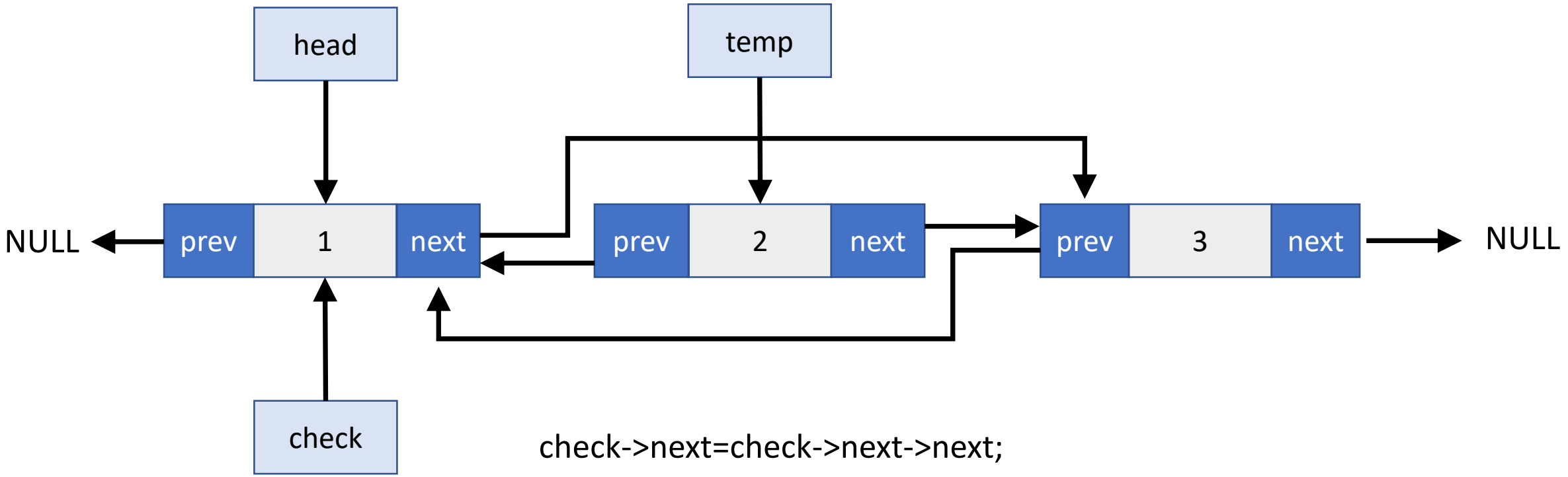
Doubly Linked List (`delete_after`)



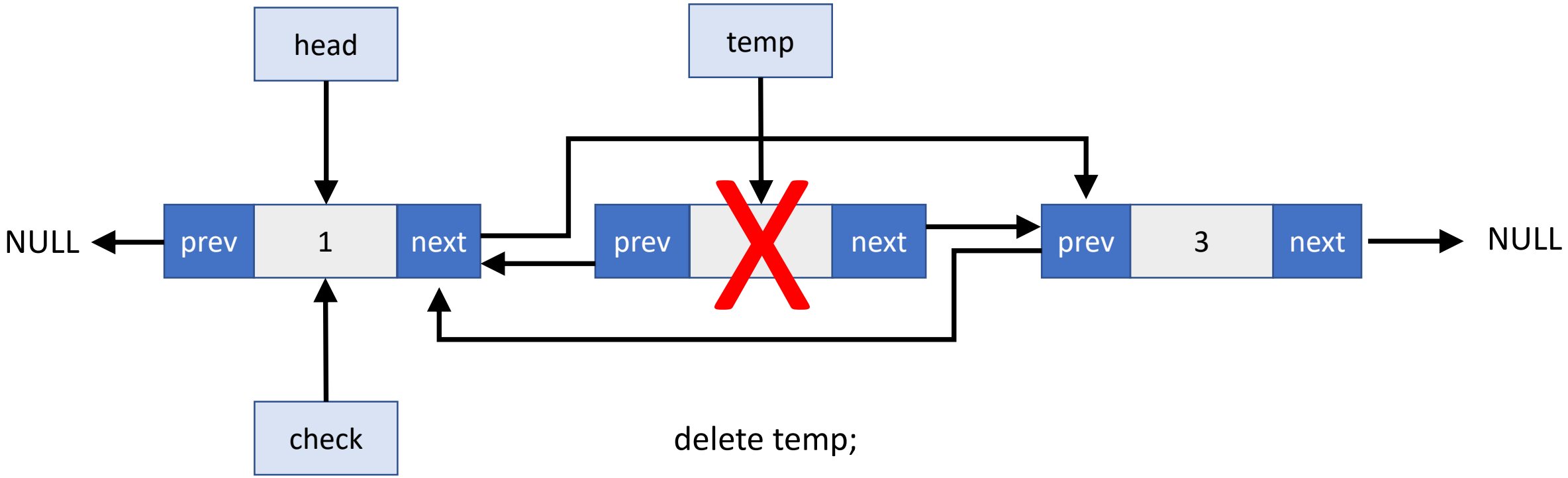
Doubly Linked List (`delete_after`)



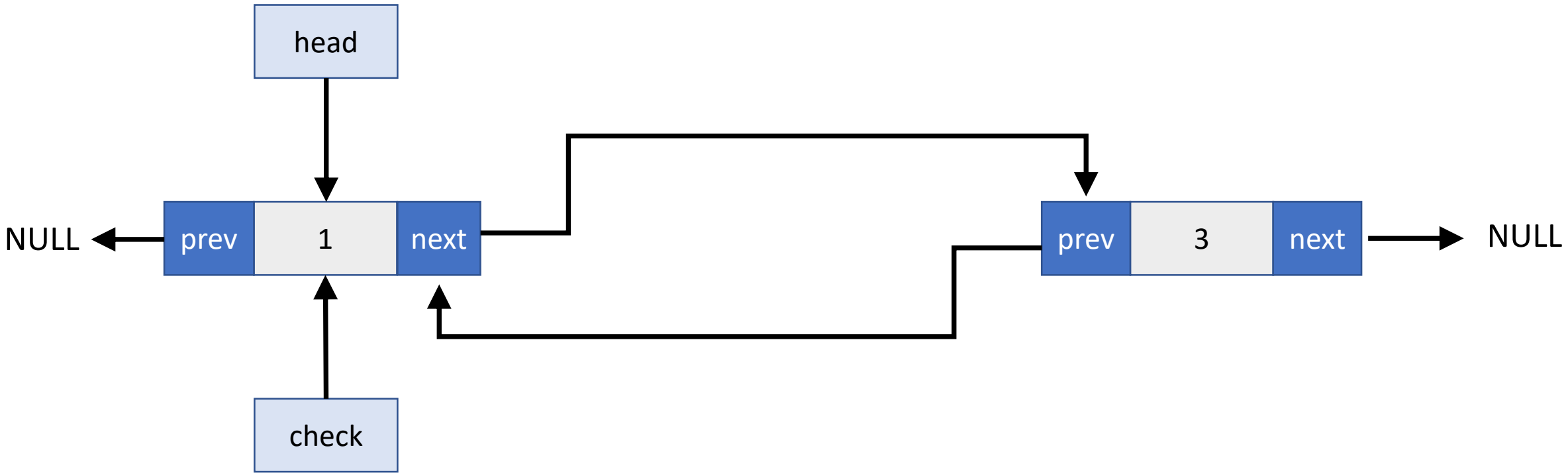
Doubly Linked List (`delete_after`)



Doubly Linked List (`delete_after`)



Doubly Linked List (`delete_after`)



Doubly Linked List (delete_after)

```
void LinkedList::delete_after(int n)
{
    if (head == NULL)
    {
        return;
    }
    else
    {
        Node* check = head;
        while (check->data != n)
        {
            check = check->next;
            if (check == NULL)
                return;
        }
        Node* temp = check->next;
        check->next->next->previous = check;
        check->next = check->next->next;
        delete temp;
    }

}
```

Doubly Linked List (`delete_after`)

Can we do it without using ***temp*** pointer ????

Thanks a lot

If you are taking a Nap, **wake up**.....Lecture Over