CS 2124: DATA STRUCTURES
Spring 2024

6th Lecture

Topics: Advanced Linked Lists and Priority Queues

Midterm Exams

- Data Structure (Midterm Exam – In person) – Thursday, 29th Feb
  - Exam will be on Canvas (You can bring your own system or you can use lab systems)
  - Physical Attendance will be taken in Lab
  - Location: NPB 1.226
  - Lectures 1 to 6
  - Timing:

Sections		Time/ NPB 1.226	Students
CS 2124-0CA	36734	10:00 – 10:30	30
CS 2124-0CB	36736	10:40 – 11:10	29
CS 2124-0DA	36738	11:30 – 12:00	30
CS 2124-0DB	36739	12:10 – 12:40	30
CS 2124-0EA	42879	01:30 – 02:00	30
CS 2124-0EB	42880	02:10 – 02:40	30

Midterm Exams

- Points: 20
- Time will be marked (by canvas) when students access the Exam
- Number of MCQ: 18 (Each MCQ Points vary based on difficulty)
- Once the Quiz starts students will have 30 Min to complete it.
- The quiz cannot be paused or stopped. It must be attempted in one sitting
- Kindly do not refresh or go back to the previous question (press back on the browser) as that is not allowed.
- One question will be visible at one time.
- Once you answer the question (submit) it cannot be changed

❖ Students with SDS approval only need to attempt the first 9 questions that they receive on Canvas.

❖ After completion do email for grade scaling

Topics

- Circular LLL (Linear Linked List)
- Singly LinkedList (L.L) as Circular L.L
  - Algorithm
  - Implementation
  - Operation - Insertion at Front
  - Operation - Insertion at Last
  - Operation - Delete First Element
  - Operation – Searching
  - Applications
- Dual LinkedList (DLL)
  - Memory Representation and Operations on a DLL
  - Insertion At Beginning Of DLL
  - Insertion At End Of DLL
  - Deletion At Beginning Of DLL
  - Deletion After A Specified Node
- Circular DLL
  - Implementation
- Priority Queues
  - Priority Queues – Characteristics
  - Priority Queues – Implementation

LinkedList (Dual LinkedList)

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer).

Node

Doubly Linked List

```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
}
```

```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
};
struct node *head;
```

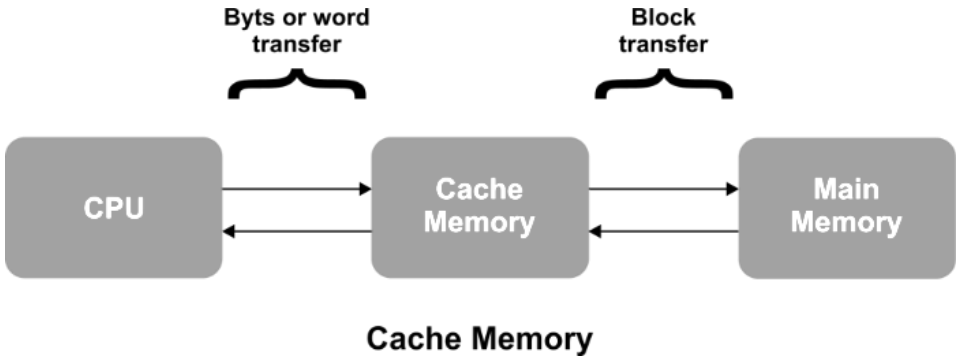
```
struct DLLNode {
int info;
struct node *left, *right;
};
```

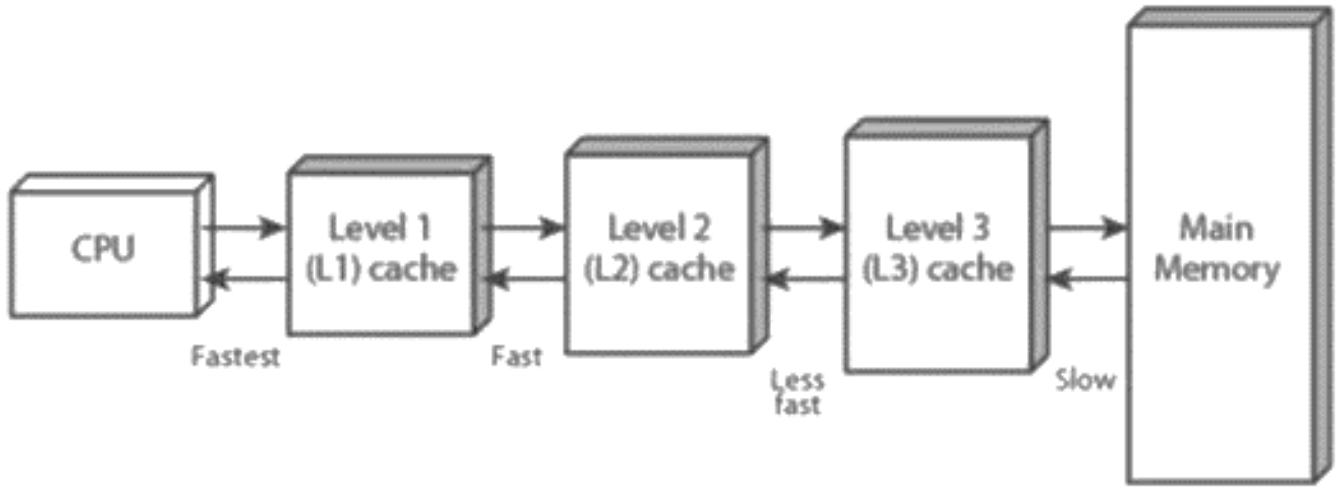
The prev part of the first node and the next part of the last node will always contain null indicating end in each direction.

Dual LinkedList (DLL)

Applications:

- It is used by web browsers for backward and forward navigation of web pages
- LRU (Least Recently Used) / MRU (Most Recently Used) Cache are constructed using Doubly Linked Lists.
- Used by various applications to maintain undo and redo functionalities.
- In Operating Systems, a doubly linked list is maintained by thread scheduler to keep track of processes that are being executed at that time.



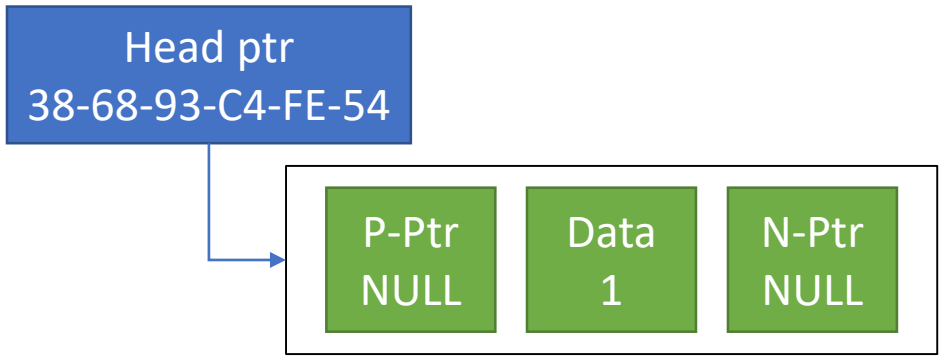


Memory Representation and Operations on a DLL

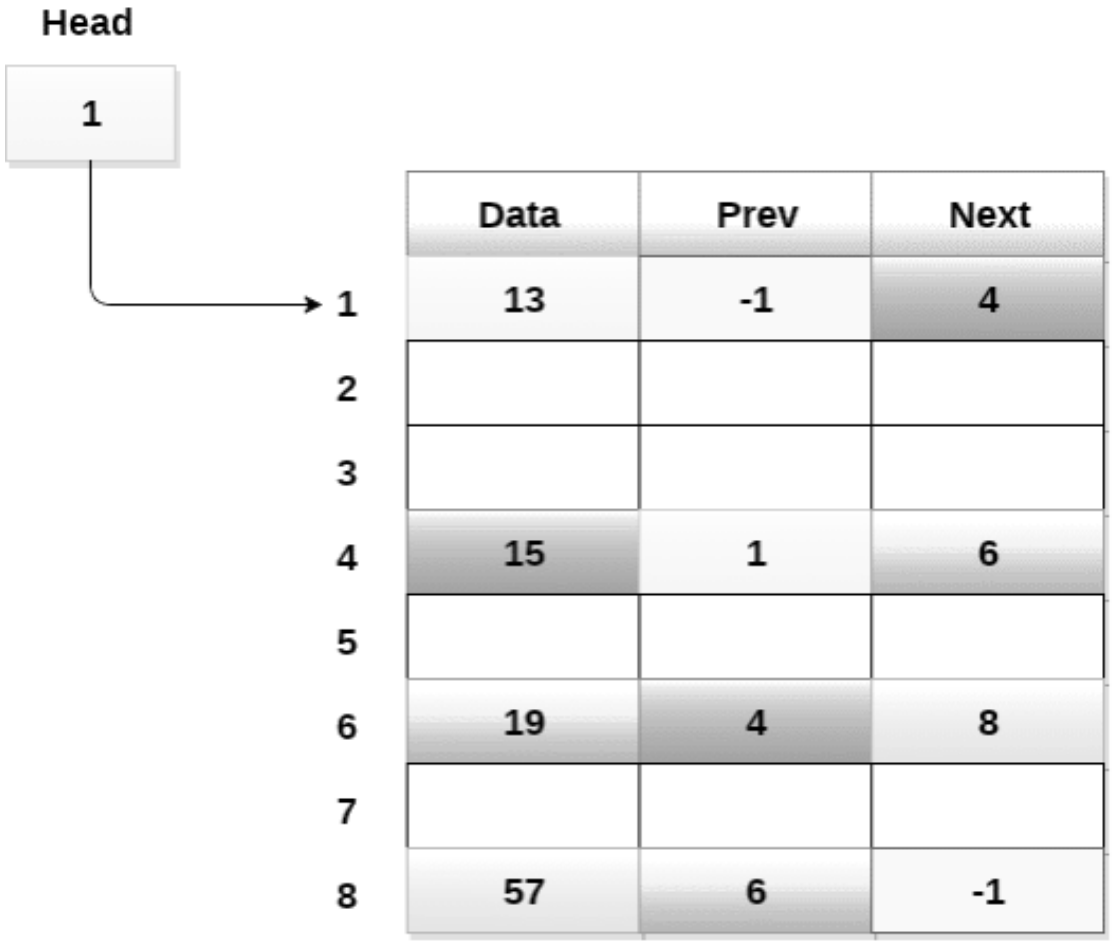
- Generally, doubly linked list consumes more space for every node and therefore, causes more expansive basic operations such as insertion and deletion. But, can easily be manipulated (forward and backward).

Insertion At Beginning	Adding the node into the linked list at beginning.
Insertion At End	Adding the node into the linked list to the end.
Insertion After Specified Node	Adding the node into the linked list after the specified node.
Deletion At Beginning	Removing the node from beginning of the list
Deletion At The End	Removing the node from end of the list.
Deletion Of The Node Having Given Data	Removing the node which is present just after the node containing the given data.
Searching	Comparing each node data with the item to be searched and return the location of the item in the list if the item found else return null.
Traversing	Visiting each node of the list at least once in order to perform some specific operation like searching, sorting, display, etc.

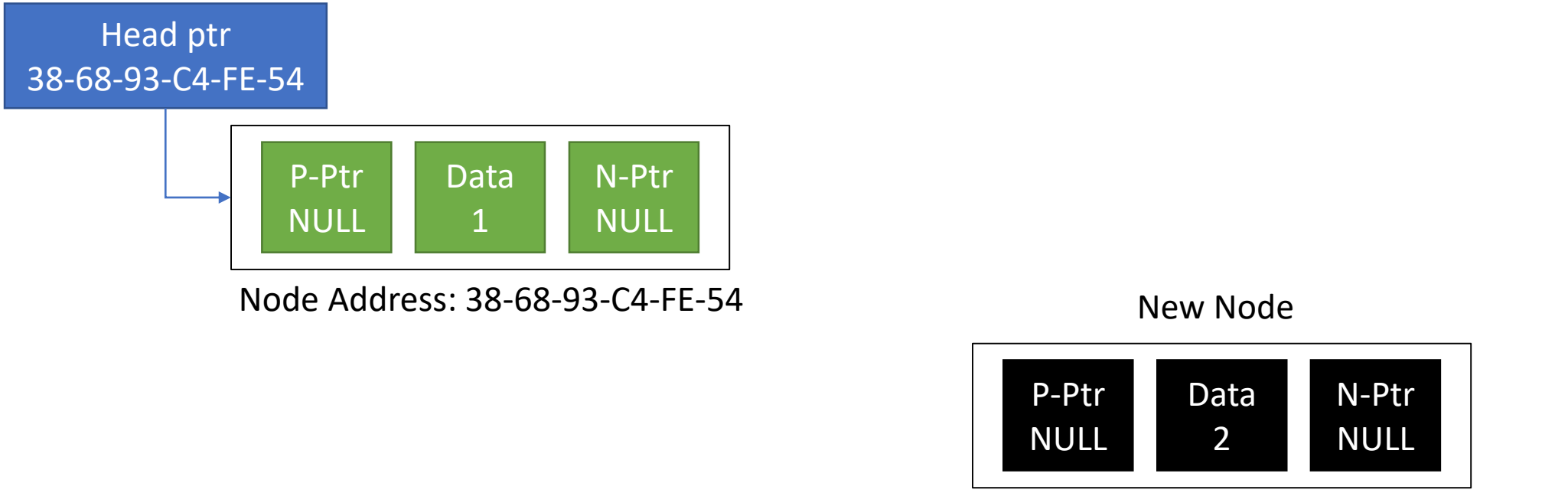
Insertion At Beginning Of DLL



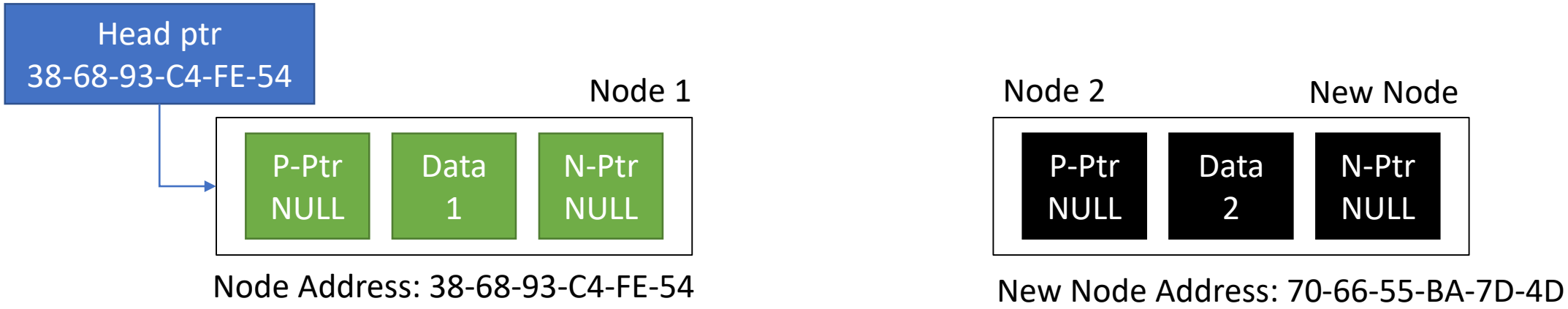
Node Address: 38-68-93-C4-FE-54



Insertion At Beginning Of DLL



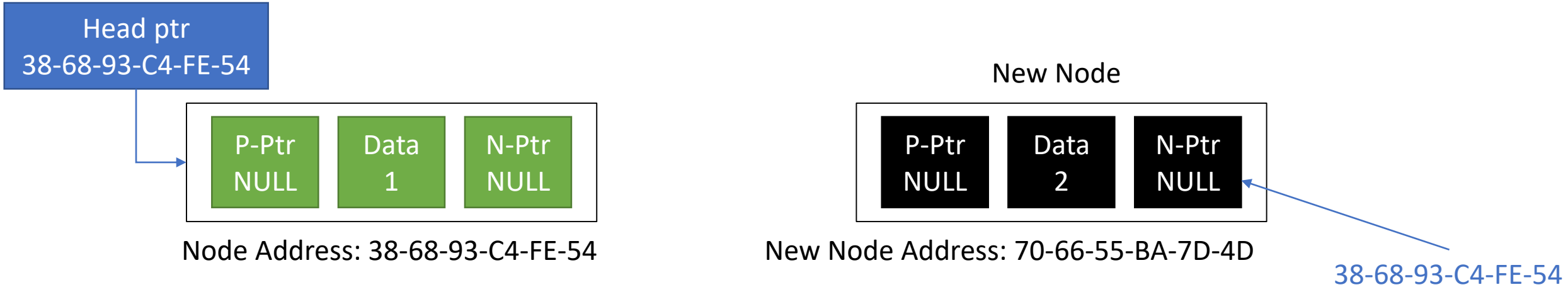
Insertion At Beginning Of DLL



Target is to move the new node in front of the existing node

1. Head pointer should point to node 2
2. Next Pointer of node 2 should point to node 1
3. Node 1 Previous pointer should point to node 2

Insertion At Beginning Of DLL



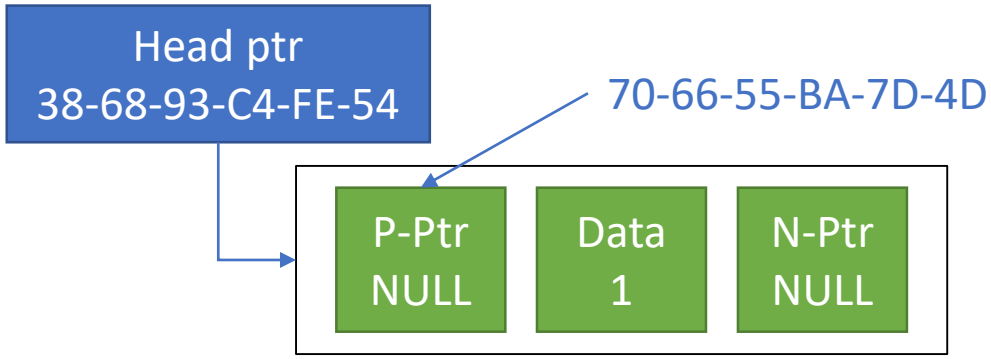
Target is to move the new node in front of the existing node

1. Head pointer should point to node 2
2. Next Pointer of node 2 should point to node 1
3. Node 1 Previous pointer should point to node 2

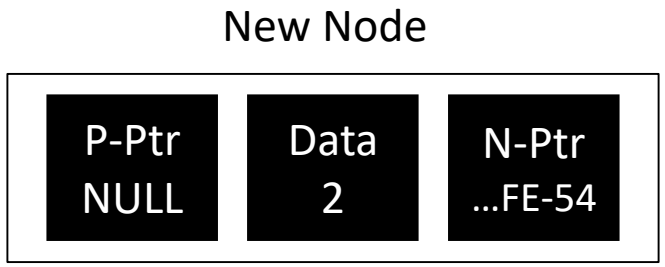
temp -> next = head

We are using temp pointer to exchange addresses

Insertion At Beginning Of DLL



Node Address: 38-68-93-C4-FE-54



New Node Address: 70-66-55-BA-7D-4D

Target is to move the new node in front of the existing node

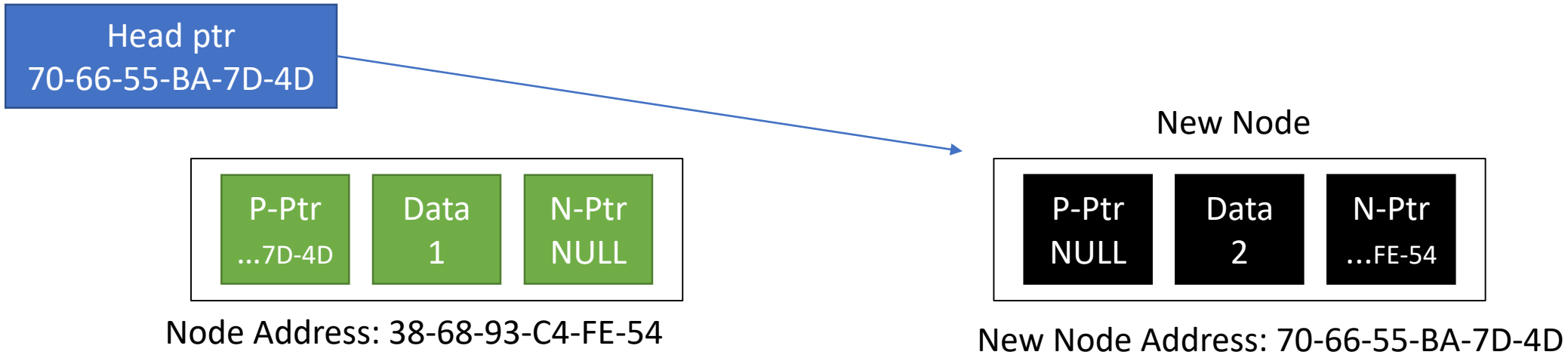
1. Head pointer should point to node 2
2. Next Pointer of node 2 should point to node 1
3. Node 1 Previous pointer should point to node 2

temp -> next = head

head -> prev = temp

(i.e. 70-66-55-BA-7D-4D)

Insertion At Beginning Of DLL

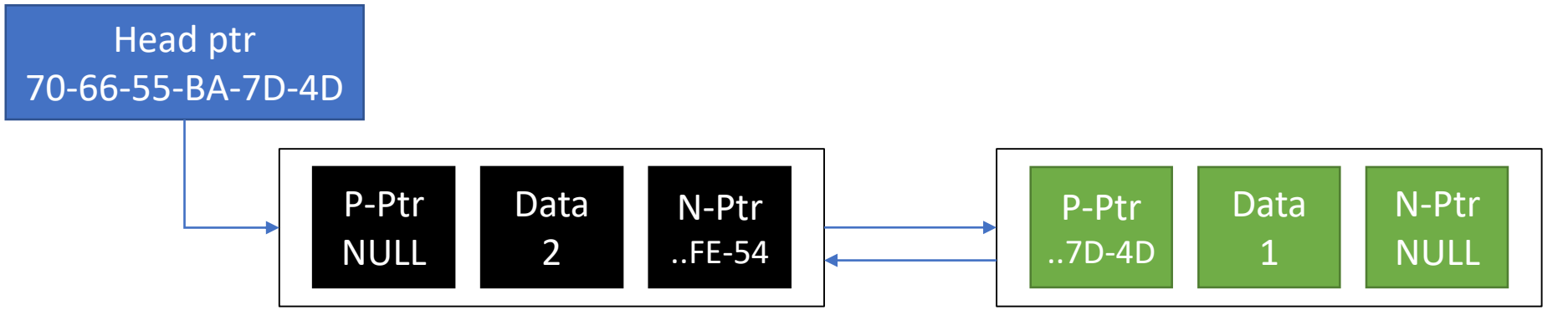


Target is to move the new node in front of the existing node

1. Head pointer should point to node 2
2. Next Pointer of node 2 should point to node 1
3. Node 1 Previous pointer should point to node 2

1. temp -> next = head
2. head -> prev = temp
3. head = temp

Insertion At Beginning Of DLL



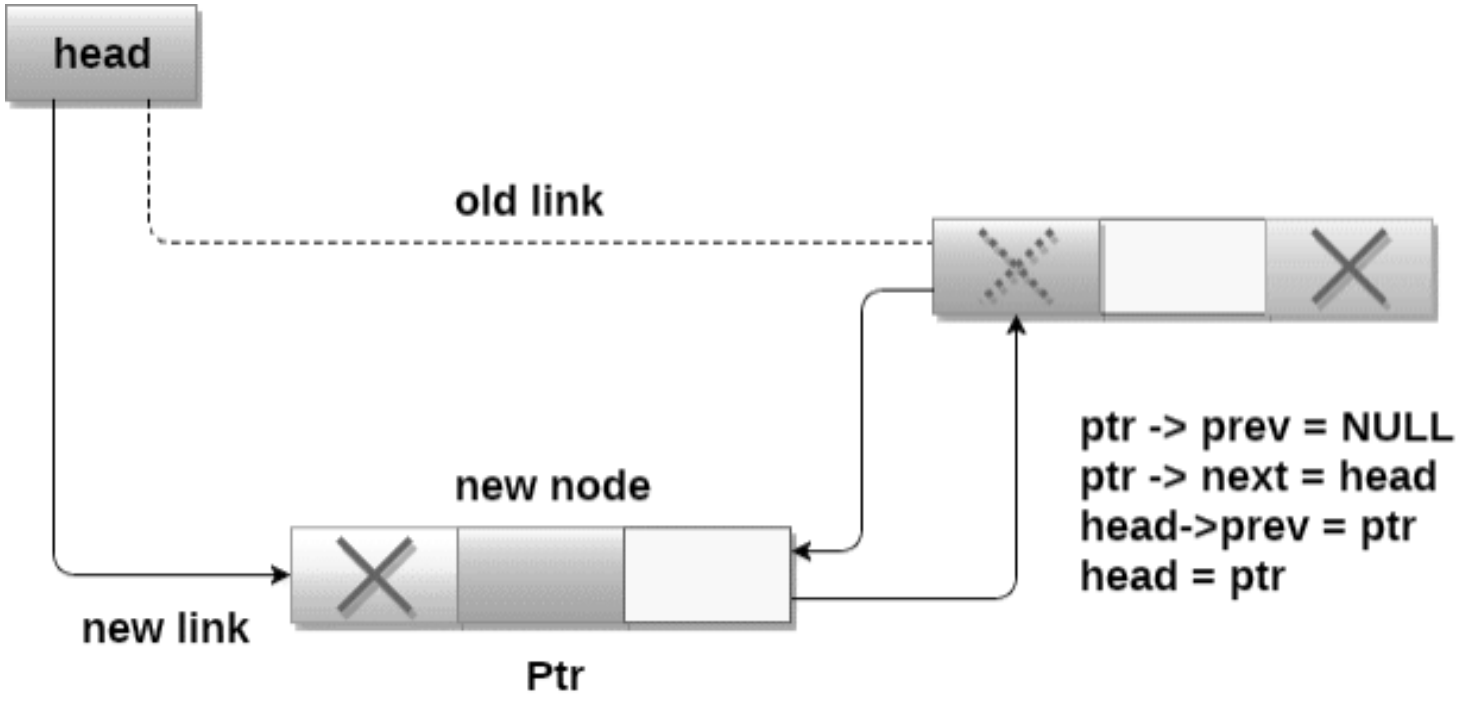
Target is to move the new node in front of the existing node

1. Head pointer should point to node 2
2. Next Pointer of node 2 should point to node 1
3. Node 1 Previous pointer should point to node 2

1. temp -> next = head
2. head -> prev = temp
3. Head = temp

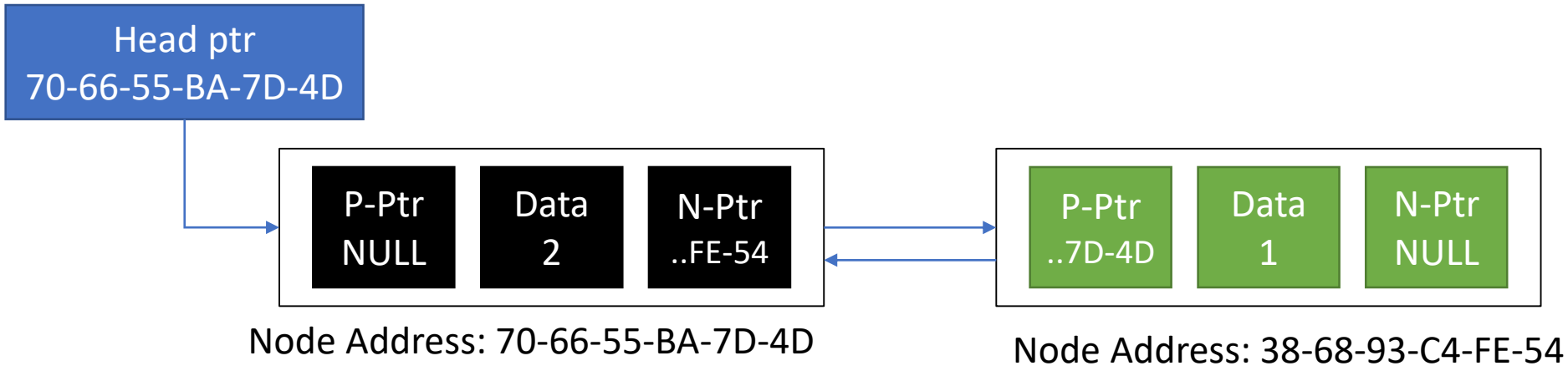
Insertion At Beginning Of DLL

- **Step 1:** IF ptr = NULL
  - Write OVERFLOW
    Go to Step 9
    [END OF IF]
- **Step 2:** SET NEW_NODE = ptr
- **Step 3:** SET ptr = ptr -> NEXT
- **Step 4:** SET NEW_NODE -> DATA = VAL
- **Step 5:** SET NEW_NODE -> PREV = NULL
- **Step 6:** SET NEW_NODE -> NEXT = START
- **Step 7:** SET head -> PREV = NEW_NODE
- **Step 8:** SET head = NEW_NODE
- **Step 9:** EXIT

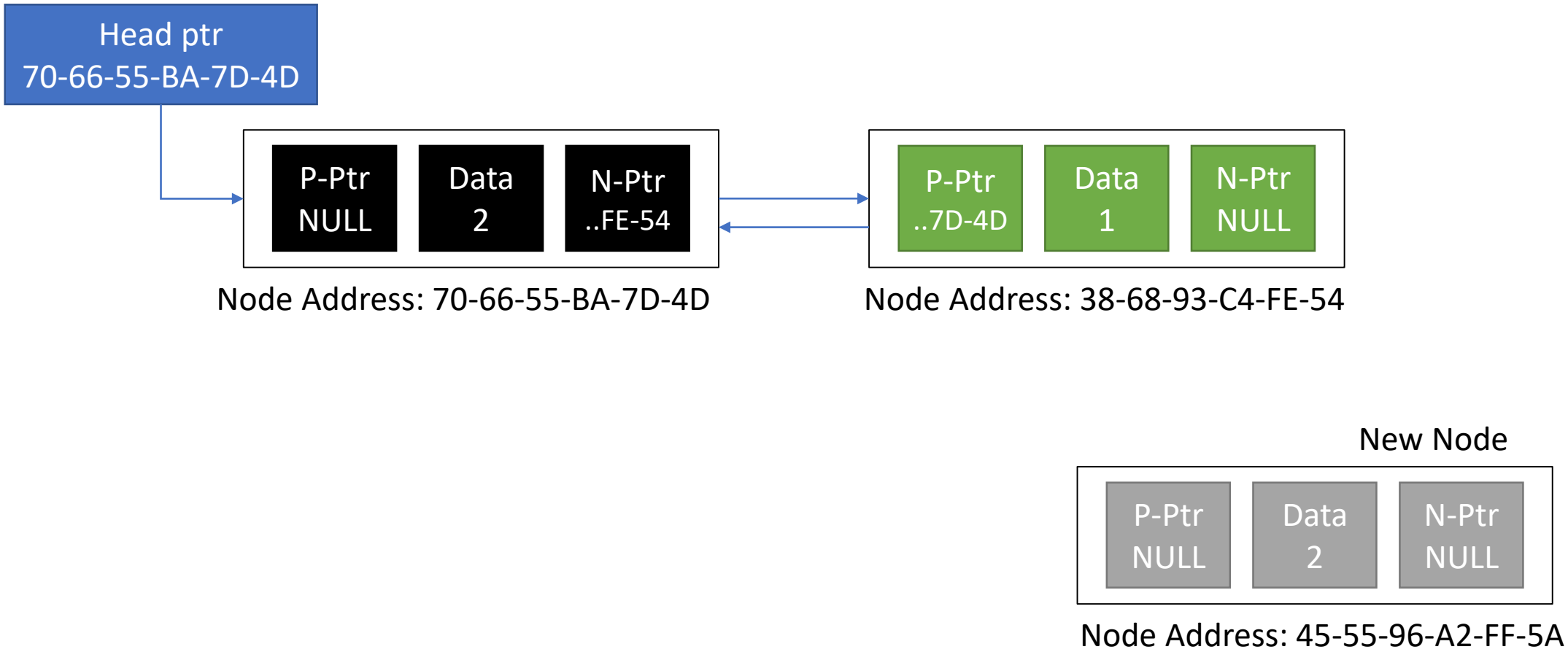


Implementation at the end of lecture

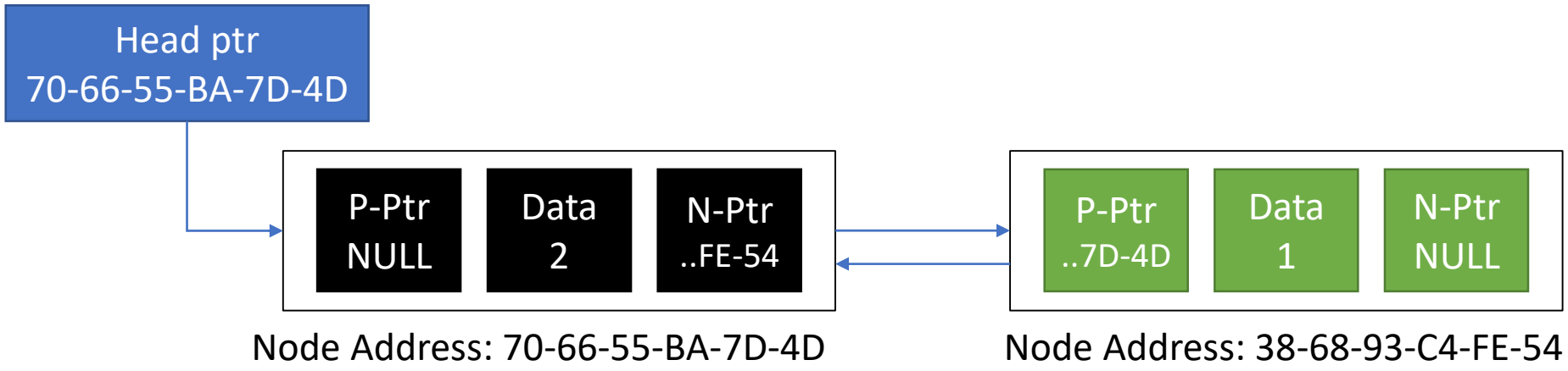
Insertion At End Of DLL



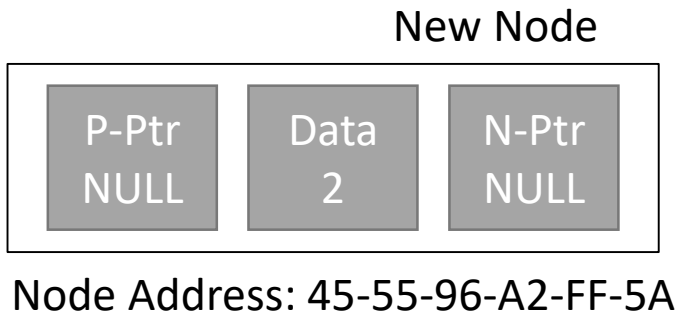
Insertion At End Of DLL



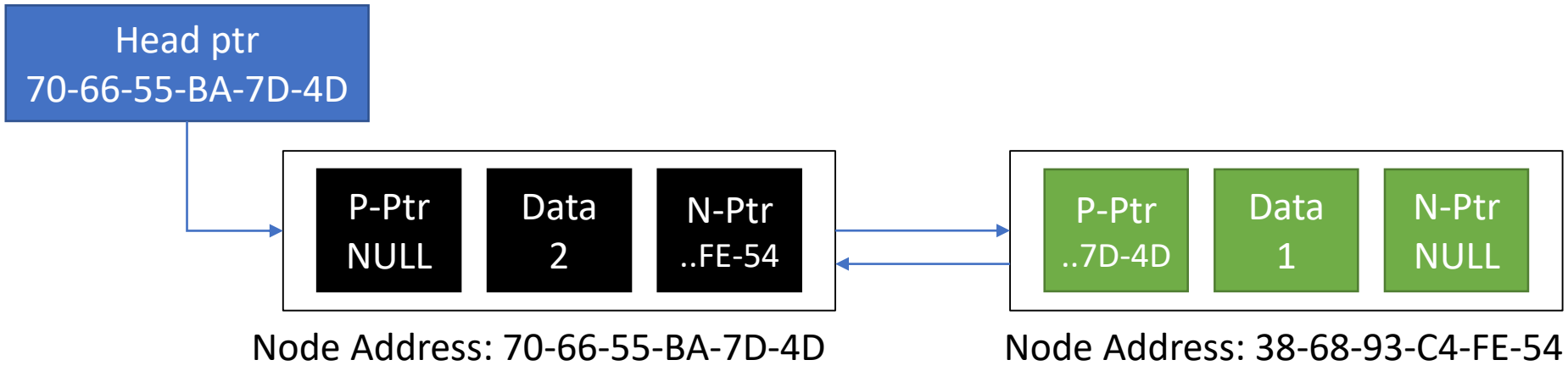
Insertion At End Of DLL



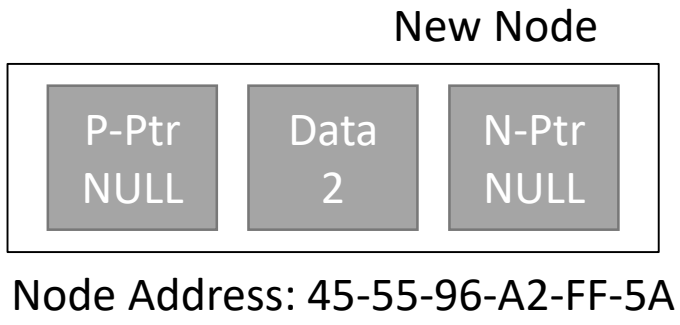
1. Traverse to the end of the Linked List



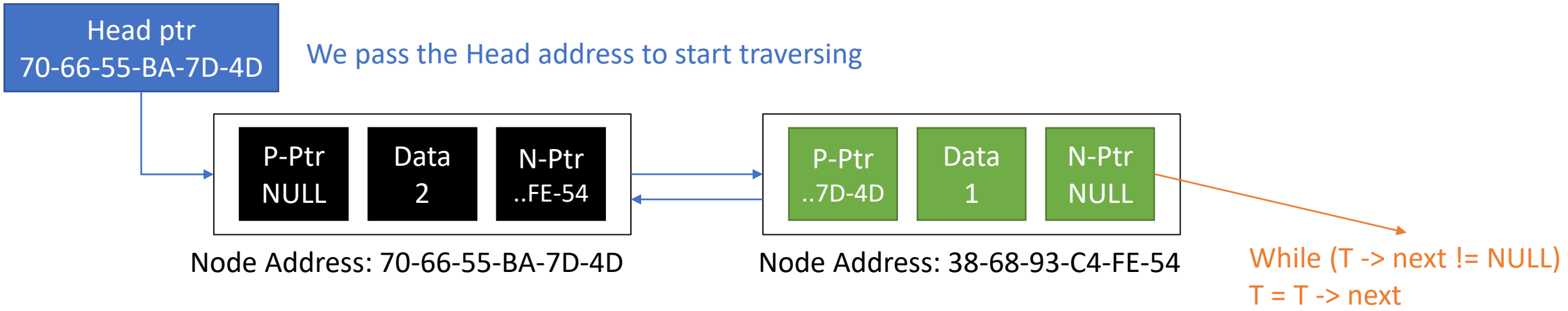
Insertion At End Of DLL



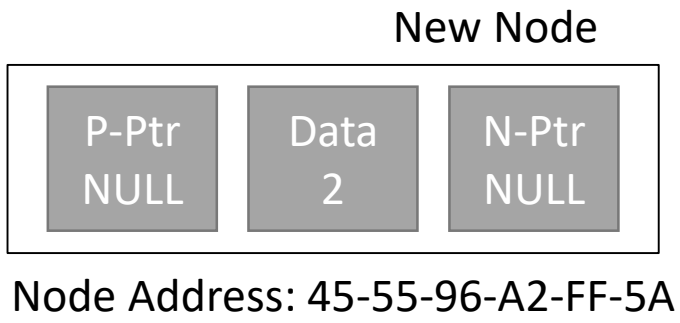
1. Traverse to the end of the Linked List



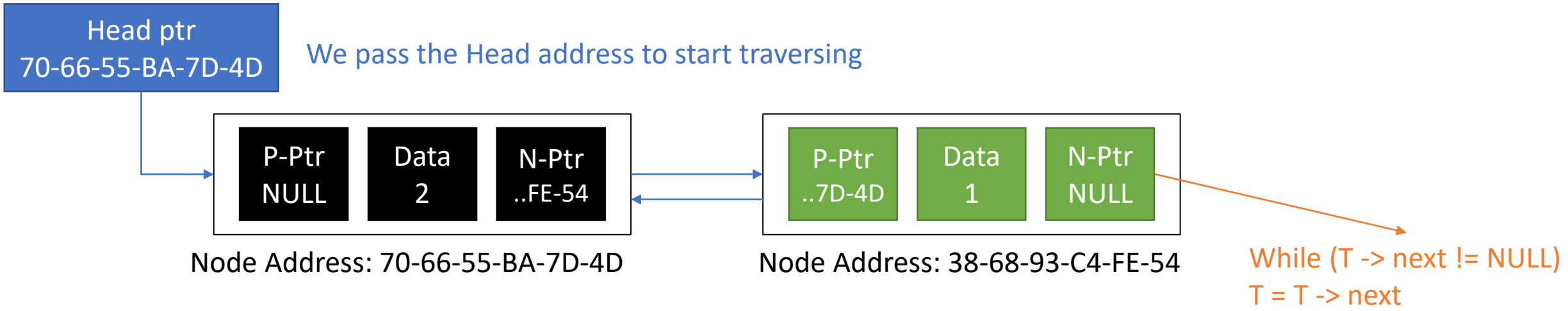
Insertion At End Of DLL



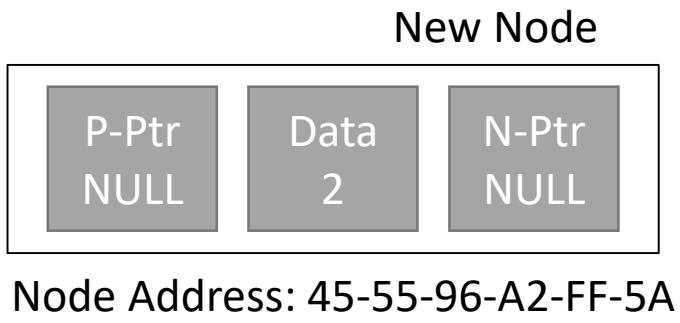
1. Traverse to the end of the Linked List



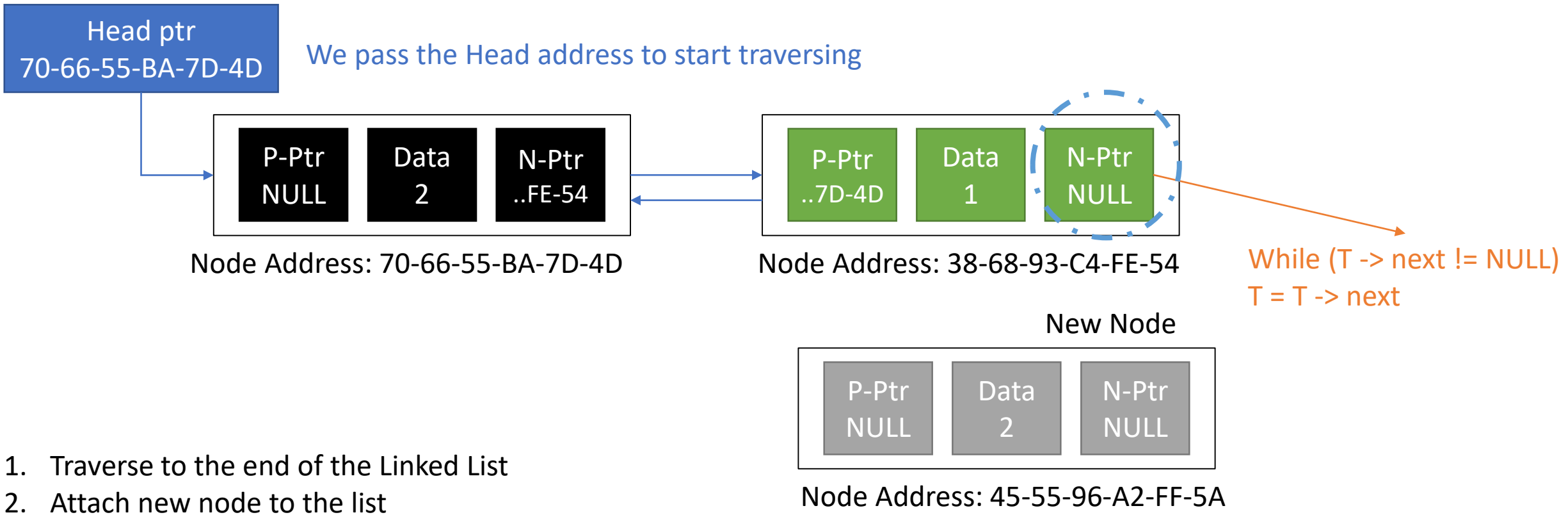
Insertion At End Of DLL



1. Traverse to the end of the Linked List
2. Attach new node to the list



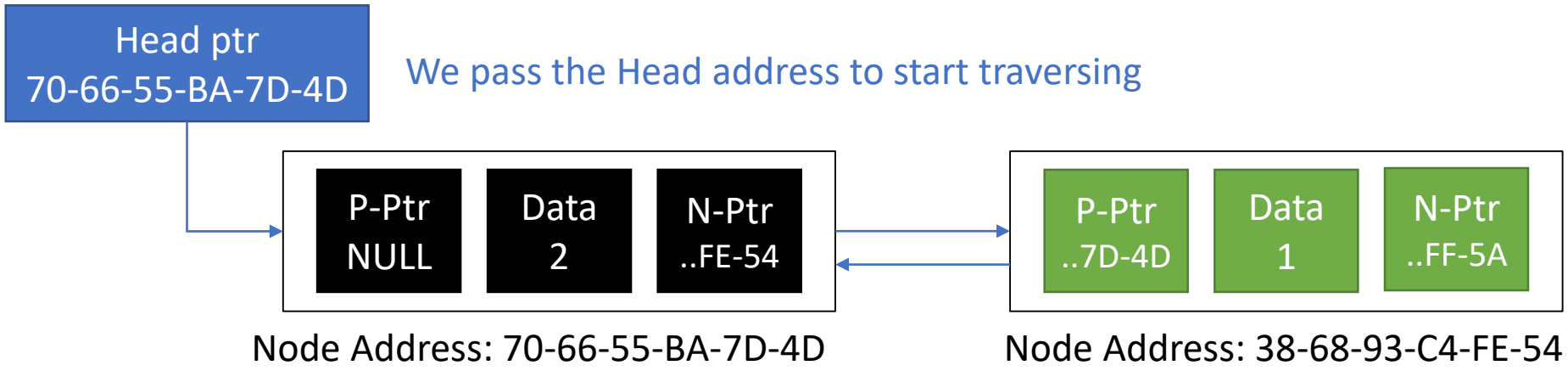
Insertion At End Of DLL

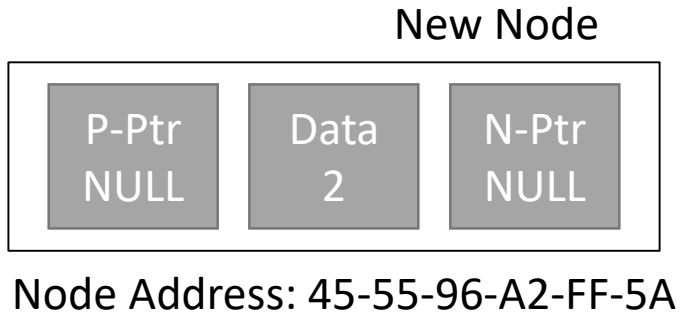


1. Traverse to the end of the Linked List
2. Attach new node to the list

T -> next = temp (We need to update N-ptr of node 2 to point to new node)

Insertion At End Of DLL

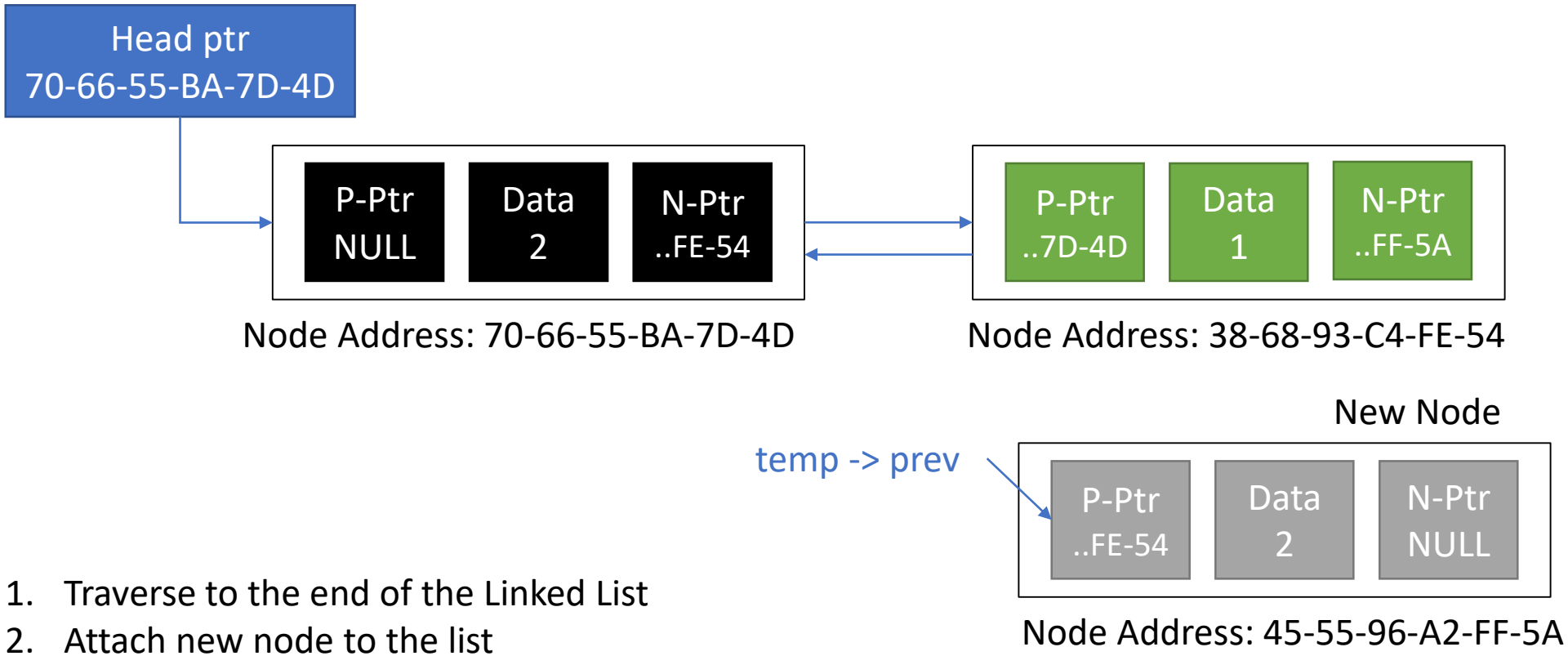




1. Traverse to the end of the Linked List
2. Attach new node to the list

T -> next = temp (We need to update N-ptr of node 2 to point to new node)

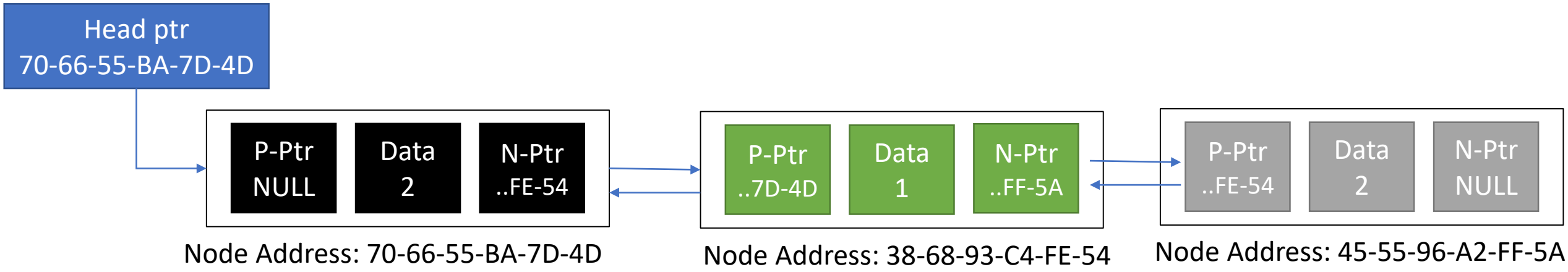
Insertion At End Of DLL



1. Traverse to the end of the Linked List
2. Attach new node to the list

T -> next = temp (We need to update N-ptr of node 2 to point to new node)
temp -> prev = T (We update the new node prev-pointer to node 2)

Insertion At End Of DLL

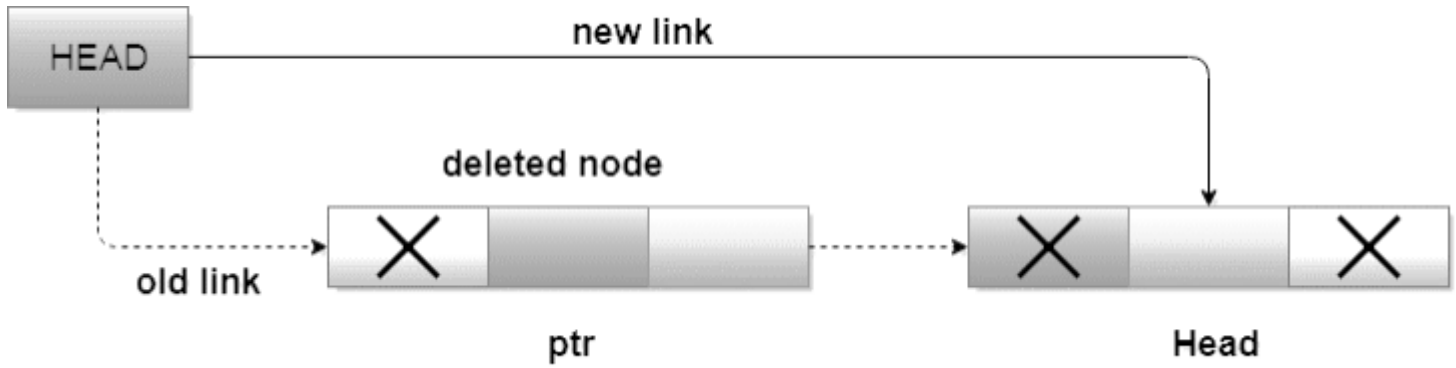


1. Traverse to the end of the Linked List
2. Attach new node to the list

T -> next = temp (We need to update N-ptr of node 2 to point to new node)
temp -> prev = T (We update the new node prev-pointer to node 2)

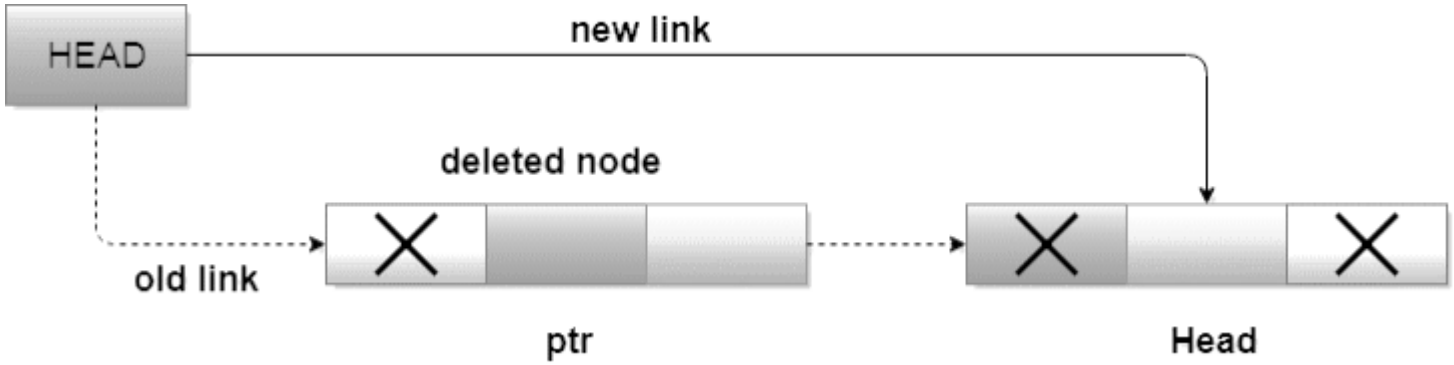
Deletion At Beginning Of DLL

- **Step 1**: If Head = Null
  - Write Underflow
  - Goto Step 6
- **Step 6**: Exit



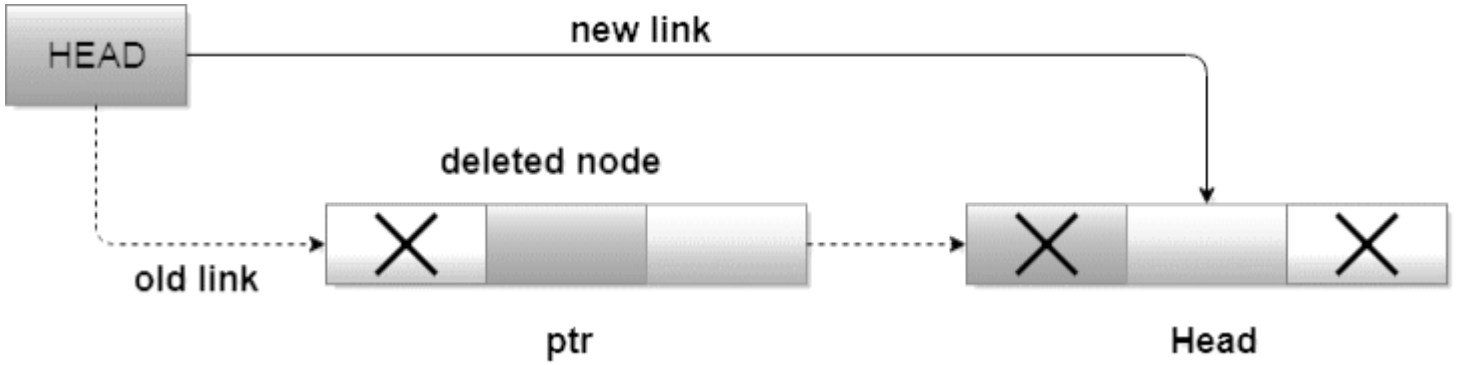
Deletion At Beginning Of DLL

- **Step 1**: If Head = Null
  - Write Underflow
  - Goto Step 6
- **Step 2**: Set Ptr = Head
- **Step 6**: Exit



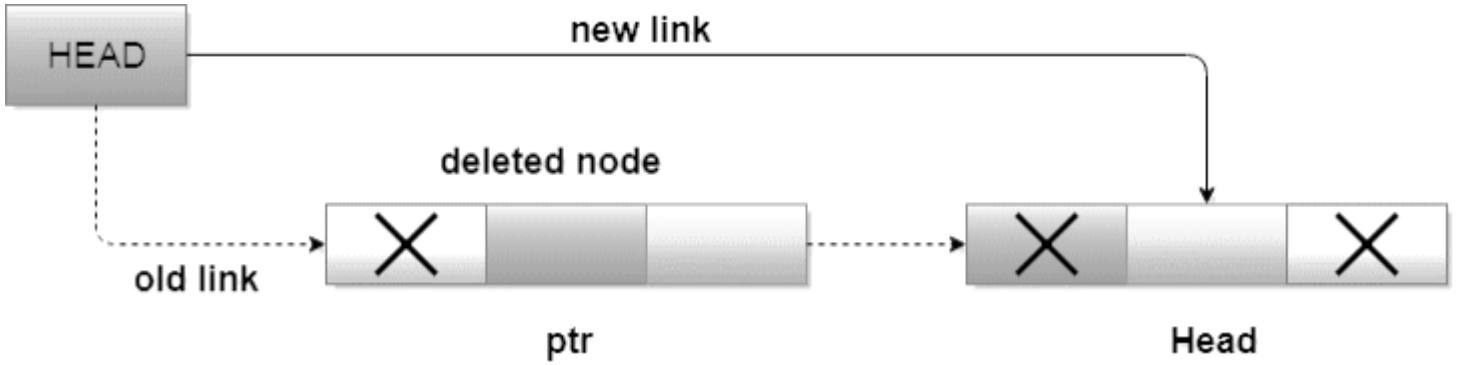
Deletion At Beginning Of DLL

- **Step 1**: If Head = Null
  - Write Underflow
  - Goto Step 6
- **Step 2**: Set Ptr = Head
- **Step 3**: Set Head = Head → Next
- **Step 6**: Exit



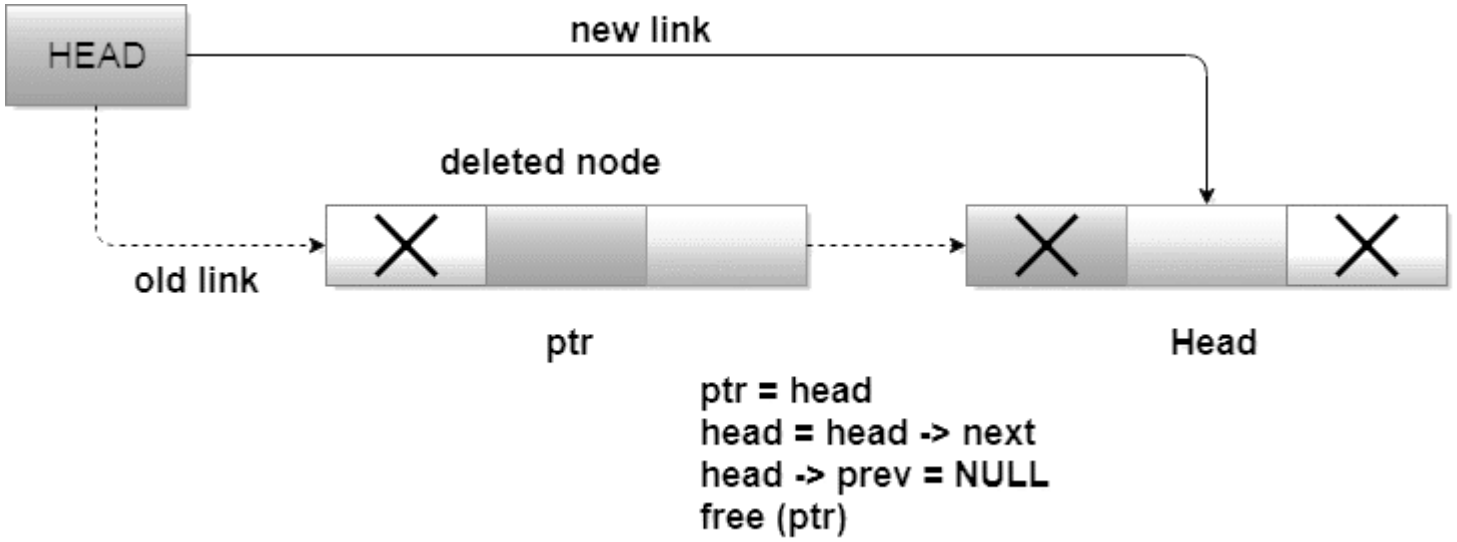
Deletion At Beginning Of DLL

- **Step 1**: If Head = Null
  - Write Underflow
  - Goto Step 6
- **Step 2**: Set Ptr = Head
- **Step 3**: Set Head = Head → Next
- **Step 4**: Set Head → Prev = Null
- **Step 6**: Exit



Deletion At Beginning Of DLL

- **Step 1**: If Head = Null
  - Write Underflow
  - Goto Step 6
- **Step 2**: Set Ptr = Head
- **Step 3**: Set Head = Head → Next
- **Step 4**: Set Head → Prev = Null
- **Step 5**: Free Ptr
- **Step 6**: Exit



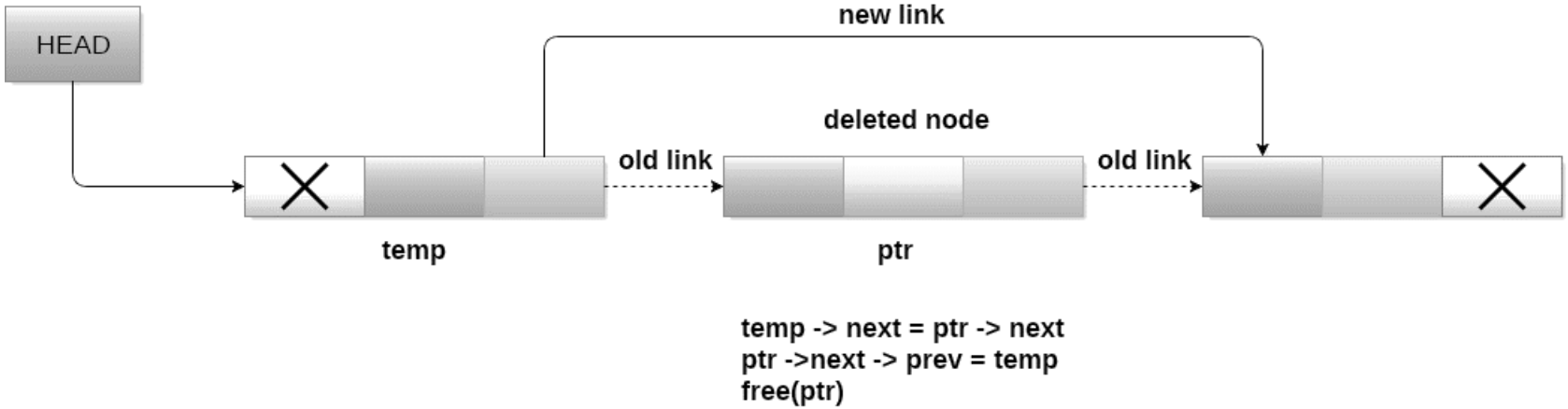
Deletion In DLL After The Specified Node

(Searching Node based on data)

- **Step 1:** IF HEAD = NULL

  Write UNDERFLOW
  Go to Step 9
  [END OF IF]
- **Step 2:** SET TEMP = HEAD
- **Step 3:** Repeat Step 4 while TEMP -> DATA != ITEM
- **Step 4:** SET TEMP = TEMP -> NEXT

  [END OF LOOP]
- **Step 5:** SET PTR = TEMP -> NEXT
- **Step 6:** SET TEMP -> NEXT = PTR -> NEXT
- **Step 7:** SET PTR -> NEXT -> PREV = TEMP
- **Step 8:** FREE PTR
- **Step 9:** EXIT



Dual LinkedList
(Node creation & Insertion in front) – Part 1/5

```c
#include <stdio.h>
#include <stdlib.h>
// node creation
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};
// insert node at the front
void insertFront(struct Node** head, int data) {
  // allocate memory for newNode
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  // assign data to newNode
  newNode->data = data;
  // make newNode as a head
  newNode->next = (*head);
  // assign null to prev
  newNode->prev = NULL;
  // previous of head (now head is the second node) is newNode
  if ((*head) != NULL)
    (*head)->prev = newNode;
  // head points to newNode
  (*head) = newNode;  }
```

Dual LinkedList
(Insert node at specific location) – Part 2/5

```
25 // insert a node after a specific node
26 void insertAfter(struct Node* prev_node, int data) {
27   // check if previous node is null
28   if (prev_node == NULL) {
29     printf("previous node cannot be null");
30     return;     }
31   // allocate memory for newNode
32 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
33   // assign data to newNode
34   newNode->data = data;
35   // set next of newNode to next of prev node
36   newNode->next = prev_node->next;
37   // set next of prev node to newNode
38   prev_node->next = newNode;
39   // set prev of newNode to the previous node
40   newNode->prev = prev_node;
41   // set prev of newNode's next to newNode
42   if (newNode->next != NULL)
43     newNode->next->prev = newNode;  }
```

Dual LinkedList
(Insertion node at the end) – Part 3/5

```
45 // insert a newNode at the end of the list
46 void insertEnd(struct Node** head, int data) {
47   // allocate memory for node
48 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
49   // assign data to newNode
50   newNode->data = data;
51   // assign null to next of newNode
52   newNode->next = NULL;
53   // store the head node temporarily (for later use)
54   struct Node* temp = *head;
55   // if the linked list is empty, make the newNode as head node
56   if (*head == NULL) {
57     newNode->prev = NULL;
58     *head = newNode;
59     return;  }
60 // if the linked list is not empty, traverse to the end of the linked list
61   while (temp->next != NULL)
62     temp = temp->next;
63 // now, the last node of the linked list is temp
64 // assign next of the last node (temp) to newNode
65   temp->next = newNode;
66 // assign prev of newNode to temp
67   newNode->prev = temp;
68 }
```

Dual LinkedList
(Delete a node) – Part 4/5

```
70 // delete a node from the doubly linked list
71 void deleteNode(struct Node** head, struct Node* del_node) {
72   // if head or del is null, deletion is not possible
73   if (*head == NULL || del_node == NULL)
74     return;
75   // if del_node is the head node, point the head pointer to the next of del_node
76   if (*head == del_node)
77     *head = del_node->next;
78   // if del_node is not at the last node, point the prev of node next to del_node to the previous of del_node
79   if (del_node->next != NULL)
80     del_node->next->prev = del_node->prev;
81   // if del_node is not the first node, point the next of the previous node to the next node of del_node
82   if (del_node->prev != NULL)
83     del_node->prev->next = del_node->next;
84   // free the memory of del_node
85   free(del_node);
86 }
```

Dual LinkedList

(Display LinkedList & Main function) – Part 5/5

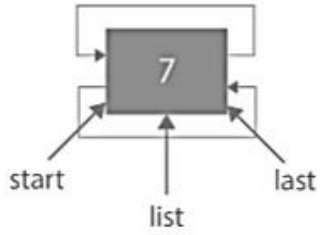
```
88  // print the doubly linked list
89  void displayList(struct Node* node) {
90    struct Node* last;
91    while (node != NULL) {
92      printf("%d->", node->data);
93      last = node;
94      node = node->next;
95    }
96    if (node == NULL)
97      printf("NULL\n");
98  }
99  int main() {
100   // initialize an empty node
101   struct Node* head = NULL;
102   insertEnd(&head, 5);
103   insertFront(&head, 1);
104   insertFront(&head, 6);
105   insertEnd(&head, 9);
106   // insert 11 after head
107   insertAfter(head, 11);
108   // insert 15 after the seond node
109   insertAfter(head->next, 15);
110   displayList(head);
111   // delete the last node
112   deleteNode(&head, head->next->next->next->next->next);
113   displayList(head);
114 }
```

What will be the output sequence ?

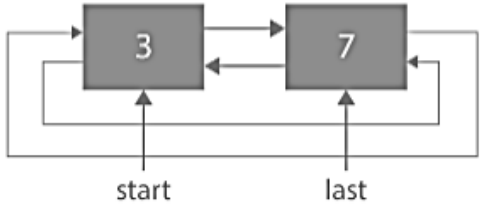
Double Linear LinkedList (Circular)

→ List ⟶ NULL

→ Inserting 7 at end. There is only one element in the list, so the end element is the first element. Our linked list after insertion will be :

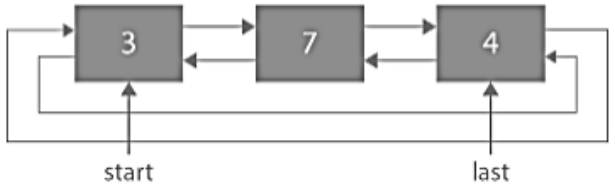


→ Inserting 3 at beginning. After insertion our linked list will be :



(Insertion will be at start)

→ Inserting 4 at end. After insertion our linked list will be :



(Insertion will be at last)

Additional Source: Link

Double Linear LinkedList (Circular)

Will this Program work ?

```c
#include <stdio.h>
#include <stdlib.h>
void display();
struct Node {
int data;
struct Node* prev;
struct Node* next;
};
int main()
{
struct Node* head;
struct Node* first=NULL;
struct Node* second=NULL;
struct Node* third=NULL;
     first = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));
first->data = 10;
second->data = 20;
third->data = 30;
```

Node structure

Pointers

```c
        first->next = second;
        first->prev=third;
        second->next = third;
        second->prev=first;
        third->next = first;
        third->prev=second;
head=first;
display(first);
return 0;
}
void display(struct Node* ptr) {
  struct Node* last;
printf("The doubly linked list elements are:\n");
  while (ptr != NULL) {
    printf("\n %d, ", ptr->data);
    last = ptr;
    printf("\n Last Node: %p, ", last);
    ptr = ptr->next;
    printf("\n Next Node: %p, ", ptr);
  }
}
```

Pointers Assignment

Double Linear LinkedList (Circular)

```c
#include <stdio.h>
#include <stdlib.h>
void display();
struct Node {
int data;
struct Node* prev;
struct Node* next;
};
int main()
{
struct Node* head;
struct Node* first=NULL;
struct Node* second=NULL;
struct Node* third=NULL;
     first = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));
first->data = 10;
second->data = 20;
third->data = 30;
        first->next = second;
        printf("\n first->next: %p, ", first->next);
        first->prev=NULL;
        printf("\n first->prev: %p, ", first->prev);
        second->next = third;
        printf("\n second->next: %p, ", second->next);
        second->prev=first;
        printf("\n second->prev: %p, ", second->prev);
        third->next = NULL;
        printf("\n third->next: %p, ", third->next);
        third->prev=second;
        printf("\n third->prev: %p, ", third->prev);
head=first;
display(first);
return 0;
}
void display(struct Node* ptr) {
  struct Node* last;
printf("\nThe doubly linked list elements are:\n");
  while (ptr != NULL) {
    printf("\n %d [%p] ", ptr->data, &ptr->data);
    last = ptr;
    ptr = ptr->next;
  }
}
```

Lines: 22 & 25

Double Linear LinkedList (Circular)

1. Will this work (line 41 i.e. use a variable to store the length of DLL)?
2. Will it display only 2 nodes or 3 nodes ?

```c
#include <stdio.h>
#include <stdlib.h>
void display();
struct Node {
int data;
struct Node* prev;
struct Node* next;
};
int main()
{
struct Node* head;
struct Node* first=NULL;
struct Node* second=NULL;
struct Node* third=NULL;
    first = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));
first->data = 10;
second->data = 20;
third->data = 30;
        first->next = second;
        printf("\n first->next: %p, ", first->next);
        first->prev=third;
        printf("\n first->prev: %p, ", first->prev);
        second->next = third;
        printf("\n second->next: %p, ", second->next);
        second->prev=first;
        printf("\n second->prev: %p, ", second->prev);
        third->next = first;
        printf("\n third->next: %p, ", third->next);
        third->prev=second;
        printf("\n third->prev: %p,\n ", third->prev);
head=first;
display(first);
return 0;
}
void display(struct Node* ptr) {
  struct Node* last;
printf("\nThe doubly linked list elements are:");
int x=0;
  while (x<3) {
    printf("\n %d [%p] ", ptr->data, &ptr->data);
    last = ptr;
    ptr = ptr->next;
    x=x+1;
  }
}
```

Double Linear LinkedList (Circular)

1. Will this work (line 41 to 46)?
2. Will it display only 2 nodes or 3 nodes ?

```c
#include <stdio.h>
#include <stdlib.h>
void display();
struct Node {
int data;
struct Node* prev;
struct Node* next;
};
int main()
{
struct Node* head;
struct Node* first=NULL;
struct Node* second=NULL;
struct Node* third=NULL;
    first = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));
first->data = 10;
second->data = 20;
third->data = 30;
        first->next = second;
        printf("\n first->next: %p, ", first->next);
        first->prev=third;
        printf("\n first->prev: %p, ", first->prev);
        second->next = third;
        printf("\n second->next: %p, ", second->next);
        second->prev=first;
        printf("\n second->prev: %p, ", second->prev);
        third->next = first;
        printf("\n third->next: %p, ", third->next);
        third->prev=second;
        printf("\n third->prev: %p,\n ", third->prev);
head=first;
display(first);
return 0;
}
void display(struct Node* ptr) {
  struct Node* last;
printf("\nThe doubly linked list elements are:");

  do {
    printf("\n %d [%p] ", ptr->data, &ptr->data);
    last = ptr;
    ptr = ptr->next;
  }
  while(ptr->data != 30);
}
```

Double Linear LinkedList (Circular)

1. Will this work as on line 46 have a different Condition?
2. Will it display only 2 nodes or 3 nodes ?

```c
#include <stdio.h>
#include <stdlib.h>
void display();
struct Node {
int data;
struct Node* prev;
struct Node* next;
};
int main()
{
struct Node* head;
struct Node* first=NULL;
struct Node* second=NULL;
struct Node* third=NULL;
    first = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));
first->data = 10;
second->data = 20;
third->data = 30;
        first->next = second;
        printf("\n first->next: %p, ", first->next);
        first->prev=third;
        printf("\n first->prev: %p, ", first->prev);
        second->next = third;
        printf("\n second->next: %p, ", second->next);
        second->prev=first;
        printf("\n second->prev: %p, ", second->prev);
        third->next = first;
        printf("\n third->next: %p, ", third->next);
        third->prev=second;
        printf("\n third->prev: %p,\n ", third->prev);
head=first;
display(first);
return 0;
}
void display(struct Node* ptr) {
  struct Node* last;
printf("\nThe doubly linked list elements are:");

  do {
    printf("\n %d [%p] ", ptr->data, &ptr->data);
    last = ptr;
    ptr = ptr->next;
  }
  while(ptr->data != 10);
}
```

Errors in Software

(Why we need to be careful about errors/warnings/defects)

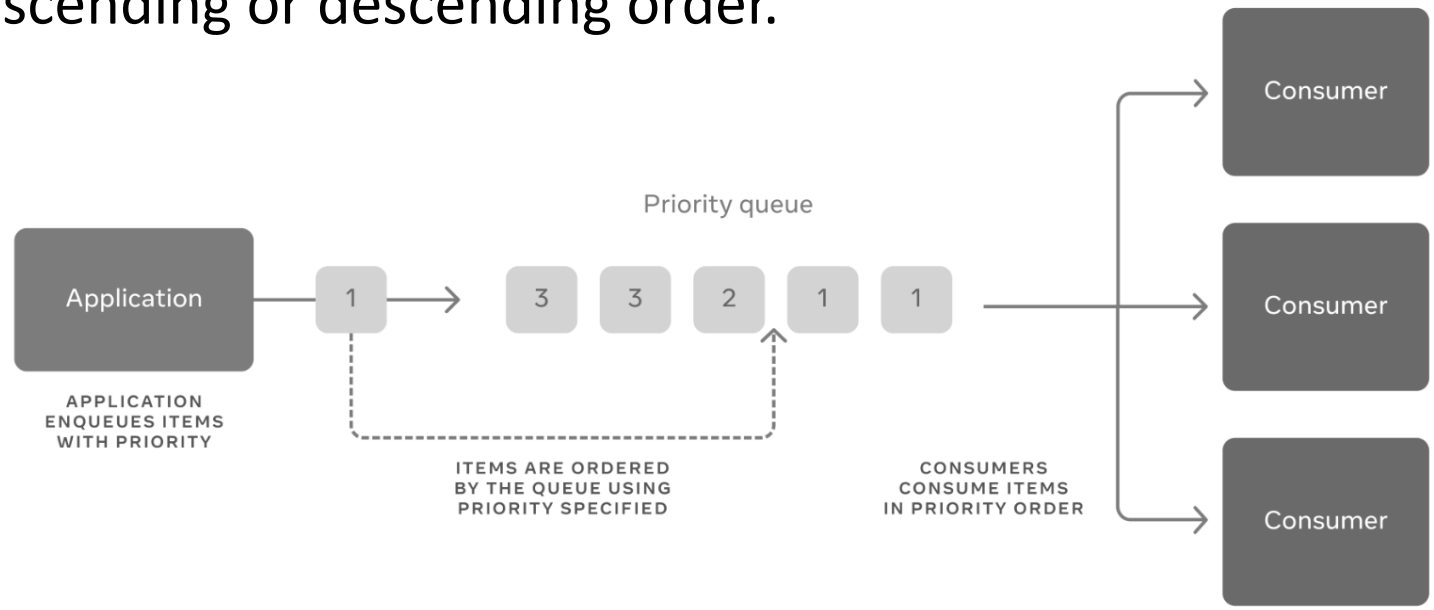
- It is difficult to find accurate numbers for errors per KLOC *(1000 lines of code)* since companies generally don't discuss things that could reflect negatively on them
- The book *Code Complete* published by Steve McDonnell in 2004 provides a range of answers on this topic:
  - ***a)*** ***Industry Average:*** *"**about 15 - 50 errors per 1000 lines** of delivered code." He further says this is usually representative of code that has some level of structured programming behind it, but probably includes a mix of coding techniques*
  - ***b)*** ***Microsoft Applications:*** *"**about 10 - 20 defects per 1000 lines** of code during in-house testing, and 0.5 defect per KLOC in released product (Moore 1992)." He attributes this to a combination of code-reading techniques and independent testing (discussed further in another chapter of his book).*

- *"Harlan Mills pioneered 'cleanroom development', a technique that has been able to achieve rates as low as 3 defects per 1000 lines of code during in-house testing and 0.1 defect per 1000 lines of code in released product (Cobb and Mills 1990). A few projects - for example, the space-shuttle software - have achieved a level of 0 defects in 500,000 lines of code using a system of format development methods, peer reviews, and statistical testing."*

- ***Book reference***: Steve McDonnell, Code Complete, 2nd Edition. Redmond, Wa.: Microsoft Press, 2004.

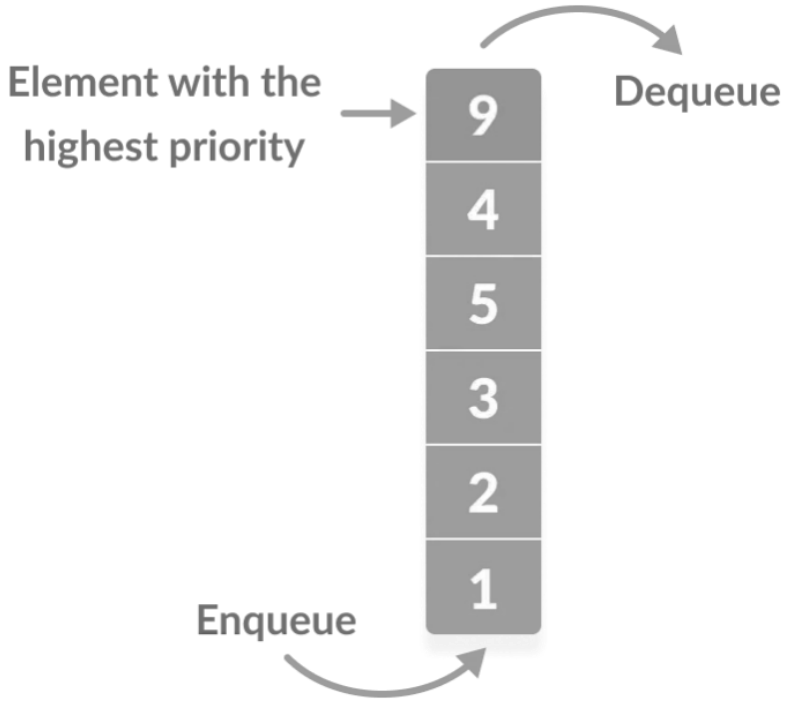
Priority Queues

- A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority,
  - i.e., the element with the highest priority would come first in a priority queue.
- The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue.
- The priority queue supports only comparable elements, which means that the elements are either arranged in an ascending or descending order.



Priority Queues (Characteristics)

- Every element in a priority queue has some priority associated with it.
- An element with the higher priority will be deleted before the deletion of the lesser priority.
- If two elements in a priority queue have the same priority, they will be arranged using the FIFO principle (General rule).
- Priority Queues can be implemented using
    1. LinkedList
    2. Binary Heap
    3. Binary Search Tree



```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
void create_queue(); void insert_element(int);
void delete_element(int); void check_priority(int);
void display_priorityqueue();
int pqueue[MAX]; int front, rear;
void main()
{
    int n, choice;
    printf("\nEnter 1 (Insert element) by priority ");
    printf("\nEnter 2 (Delete element) by priority ");
    printf("\nEnter 3 (Display) priority queue ");
    printf("\nEnter 4 to exit");
    create_queue();
    while (1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
        case 1:
            printf("\nEnter element to insert : ");
            scanf("%d",&n); insert_element(n); break;
        case 2:
            printf("\nEnter element to delete : ");
            scanf("%d",&n); delete_element(n); break;
        case 3:
            display_priorityqueue(); break;
        case 4:
            exit(0);
        default:
            printf("\n Please enter valid choice");
        }
    }
}
```

```c
void create_queue()
{
    front = rear = -1;
}
void insert_element(int data)
{
    if (rear >= MAX - 1)
    {
        printf("\nQUEUE OVERFLOW");
        return;
    }
    if ((front == -1) && (rear == -1))
    {
        front++;  rear++;
        pqueue[rear] = data;
        return;
    }
    else
        check_priority(data);
    rear++;
}
void check_priority(int data)
{
    int i,j;
    for (i = 0; i <= rear; i++)
    {
        if (data >= pqueue[i])
        {
            for (j = rear + 1; j > i; j--)
            {
                pqueue[j] = pqueue[j - 1];
            }
            pqueue[i] = data;
            return;
        }
    }
```

Priority Queues

Function on Lines 4 to 6

```
    pqueue[i] = data;
}
void delete_element(int data)
{
    int i;
    if ((front==-1) && (rear==-1))
    {
        printf("\nEmpty Queue");
        return;
    }
    for (i = 0; i <= rear; i++)
    {
        if (data == pqueue[i])
        {
            for (; i < rear; i++)
            {
                pqueue[i] = pqueue[i + 1];
            }
            pqueue[i] = -99;
            rear--;
            if (rear == -1)
               front = -1;
            return;
        }
    }
    printf("\n%d element not found in queue", data);
}
```

```
void display_priorityqueue()
{
    if ((front == -1) && (rear == -1))
    {
        printf("\nEmpty Queue ");
        return;
    }
    for (; front <= rear; front++)
    {
        printf(" %d ", pqueue[front]);
    }
     front = 0;
}
```

Priority Queues (Another approach)

```
 6  // Node
 7  typedef struct node {
 8      int data;
 9  // We can assign priority based on order
10      int priority;
11      struct node* next;
12  } Node;
13
14  // Function to Create A New Node
15  Node* newNode(int d, int p)
16  {
17      Node* temp = (Node*)malloc(sizeof(Node));
18      temp->data = d;
19      temp->priority = p;
20      temp->next = NULL;
21      return temp;
22  }
```

```
79  int main()
80  {
81      // Create a Priority Queue
82      // 7->4->5->6
83      Node* pq = newNode(4, 1);
84      push(&pq, 5, 2);
85      push(&pq, 6, 3);
86      push(&pq, 7, 0);
87  // 7 have 0 (lowest) Priority
88  printf("Start by Lowest Piroty:\n");
89      while (!isEmpty(&pq)) {
90          printf("%d \n", peek(&pq));
91          pop(&pq);
92      }
93  }
```

- Stack based priority queue
- Maybe Sort or create array based on priority. But do be careful about de-queue operation