Dynamic data structures – Linked lists

Basics of Programming 1

G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

13 November, 2024

Content

Chapter 1

Dynamic data structures

Dynamic data structure – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We are writing a chess program, in which there is undo option for arbitrary number of moves.
- The undo-list is the log of the game, its elements are the moves.
  - Which piece
  - From where
  - Where to
  - Who is captured (removed)
- For logging we use the memory we really need, no more.

Dynamic data structure – motivation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We are writing a chess program, in which there is undo option for arbitrary number of moves.
- The undo-list is the log of the game, its elements are the moves.
  - Which piece
  - From where
  - Where to
  - Who is captured (removed)
- For logging we use the memory we really need, no more.
- The final length of the log will be known only at the end of the game.
- We have to increase the amount of allocated memory with each step (or reduce it, if we undo a move).

Dyanmic data structure – motivation

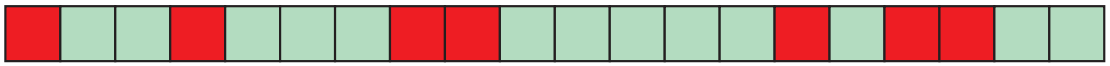
- If we use `realloc` for resizing an array, it may cause many unnecessary copying of data.

Dyanmic data structure – motivation

- If we use `realloc` for resizing an array, it may cause many unnecessary copying of data.

memory: – free, – occupied, – our array

n=0:



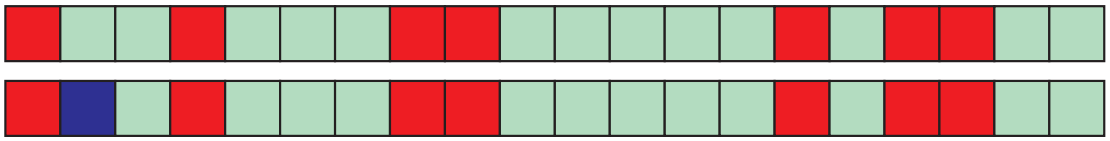
Dyanmic data structure – motivation

- If we use `realloc` for resizing an array, it may cause many unnecessary copying of data.

memory: – free, – occupied, – our array

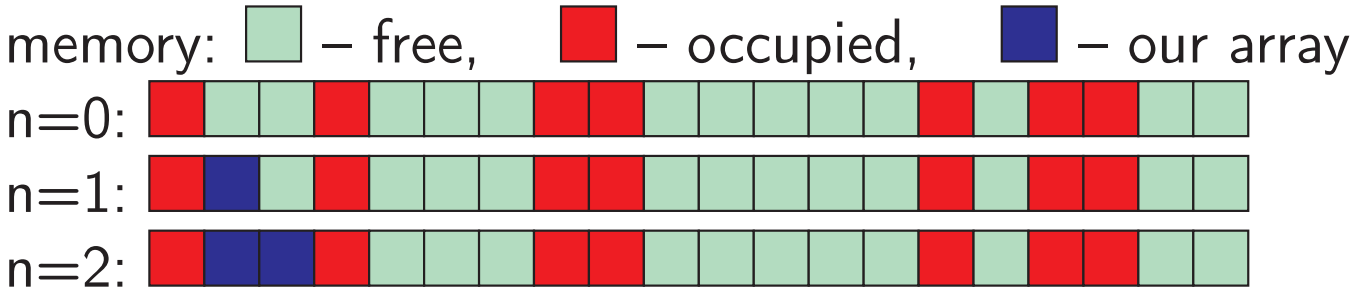
n=0:

n=1:



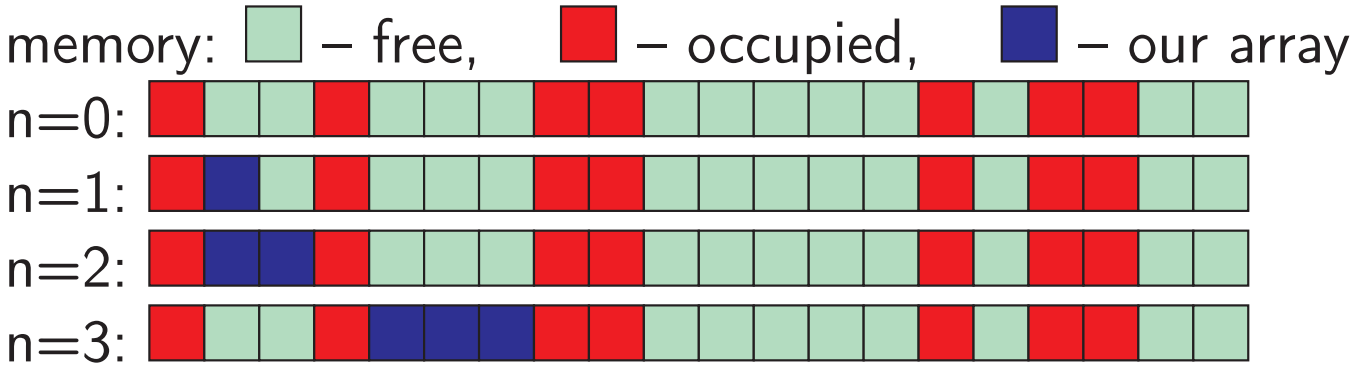
Dyanmic data structure – motivation

- If we use `realloc` for resizing an array, it may cause many unnecessary copying of data.



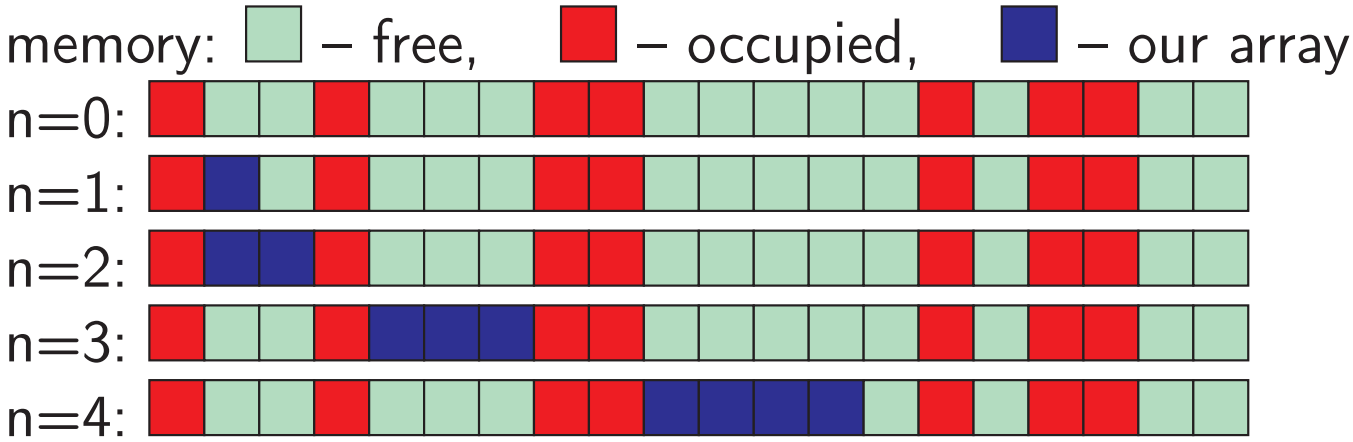
Dyanmic data structure – motivation

- If we use `realloc` for resizing an array, it may cause many unnecessary copying of data.



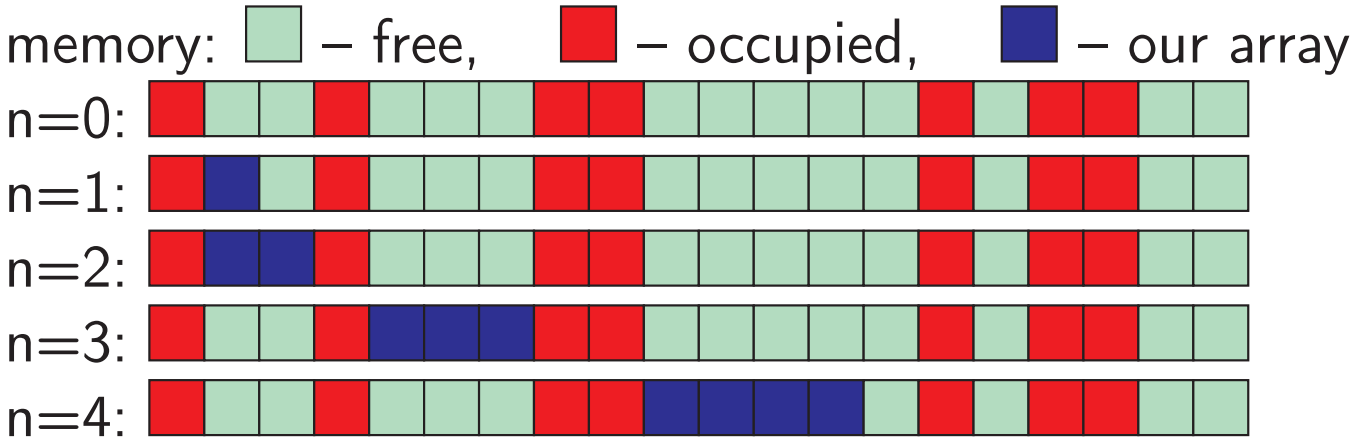
Dyanmic data structure – motivation

- If we use `realloc` for resizing an array, it may cause many unnecessary copying of data.



Dyanmic data structure – motivation

- If we use `realloc` for resizing an array, it may cause many unnecessary copying of data.



- We need a data structure that does not use continuous blocks of memory, and its strucure changes dynamically during the lifecycle of the program.

Dynamic data structure

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Dynamic data structure:

- its size or structure changes during the lifecycle of the program
- it is realized with self-referencing structure

Self-referencing structure

A compound data structure, that contains pointers pointing to itself

```
1 typedef struct listelem {
2   int data;                 /* the data we store */
3   struct listelem *next; /* address of next element */
4 } listelem;
```

Dynamic data structure

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Dynamic data structure:

- its size or structure changes during the lifecycle of the program
- it is realized with self-referencing structure

Self-referencing structure

A compound data structure, that contains pointers pointing to itself

```
1 typedef struct listelem {
2   int data;                 /* the data we store */
3   struct listelem *next; /* address of next element */
4 } listelem;
```

- `next` points to a structure that is of the same type, as the one containing the pointer itself.

Dynamic data structure

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Dynamic data structure:

- its size or structure changes during the lifecycle of the program
- it is realized with self-referencing structure

Self-referencing structure

A compound data structure, that contains pointers pointing to itself

```
1 typedef struct listelem {
2   int data;                 /* the data we store */
3   struct listelem *next; /* address of next element */
4 } listelem;
```

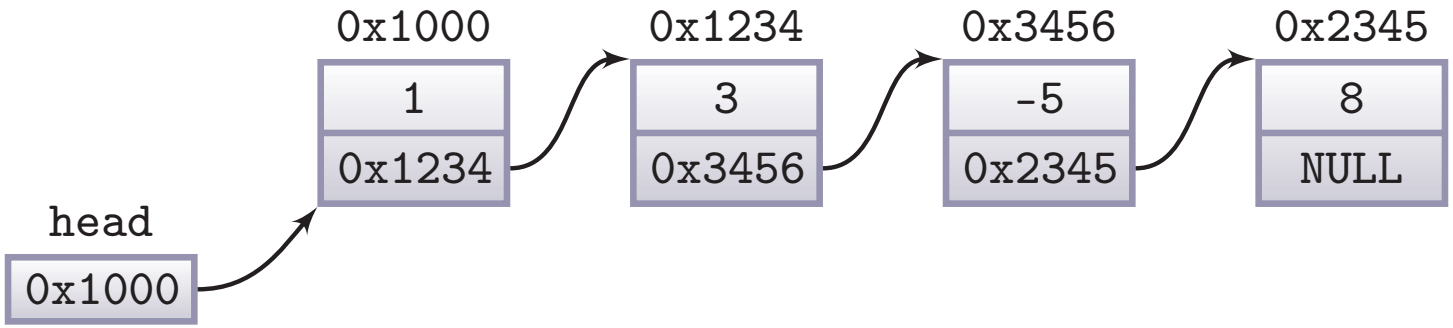
- `next` points to a structure that is of the same type, as the one containing the pointer itself.
- `struct listelem` structure is renamed to `listelem`, but when declaring `next`, we must use the long name (because the compiler doesn't know, what nickname we will give to it).

Chapter 2

Singly linked lists

Linked list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- List of `listelem` type variables
- Memory is allocated dynamically, separately for each element
- Elements do not form a continuous block in memory
- Each element contains the address of the next element
- The first element is defined by the `head` pointer
- The last element points to nowhere (`NULL`)

Linked list

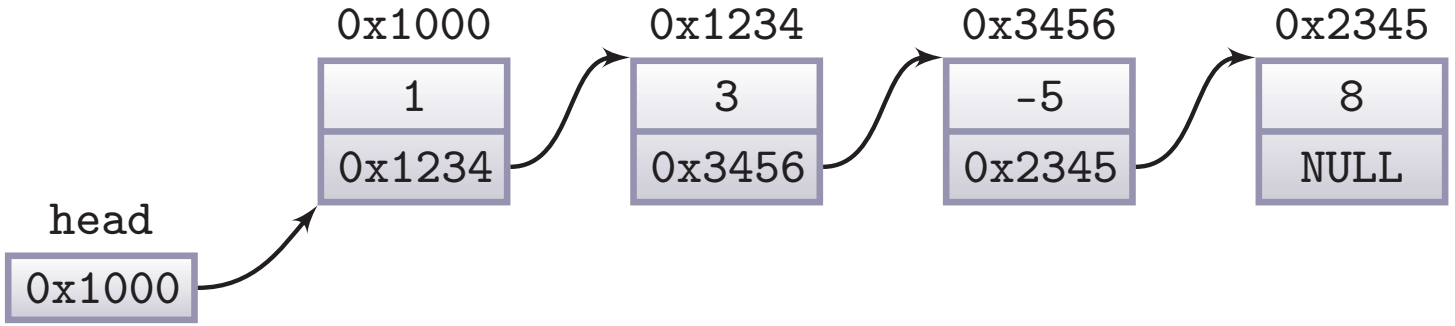
- Empty list

Linked list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Empty list

- List is a self-referencing (recursive) data structure. Each element points to a list.



List or array

- The array
  - occupies as much memory, as needed for storing the data
  - needs a continuous block of memory
  - any element can be accessed directly (immediately), by indexing
  - inserting a new data involves a lot of copying

List or array

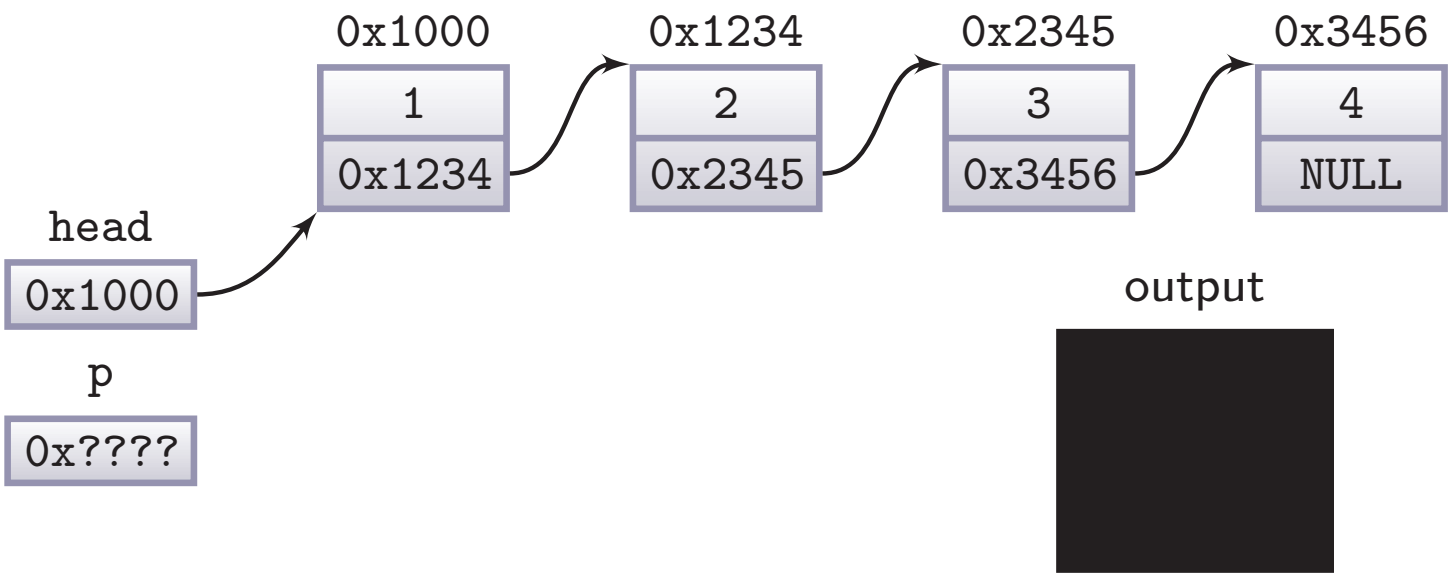
- The array
  - occupies as much memory, as needed for storing the data
  - needs a continuous block of memory
  - any element can be accessed directly (immediately), by indexing
  - inserting a new data involves a lot of copying
- The list
  - elements store the address of the next element, this may need a lot of memory
  - can make use of gaps in the fragmented memory
  - only the next element can be accessed immediately
  - inserting a new element involves only a little work

Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

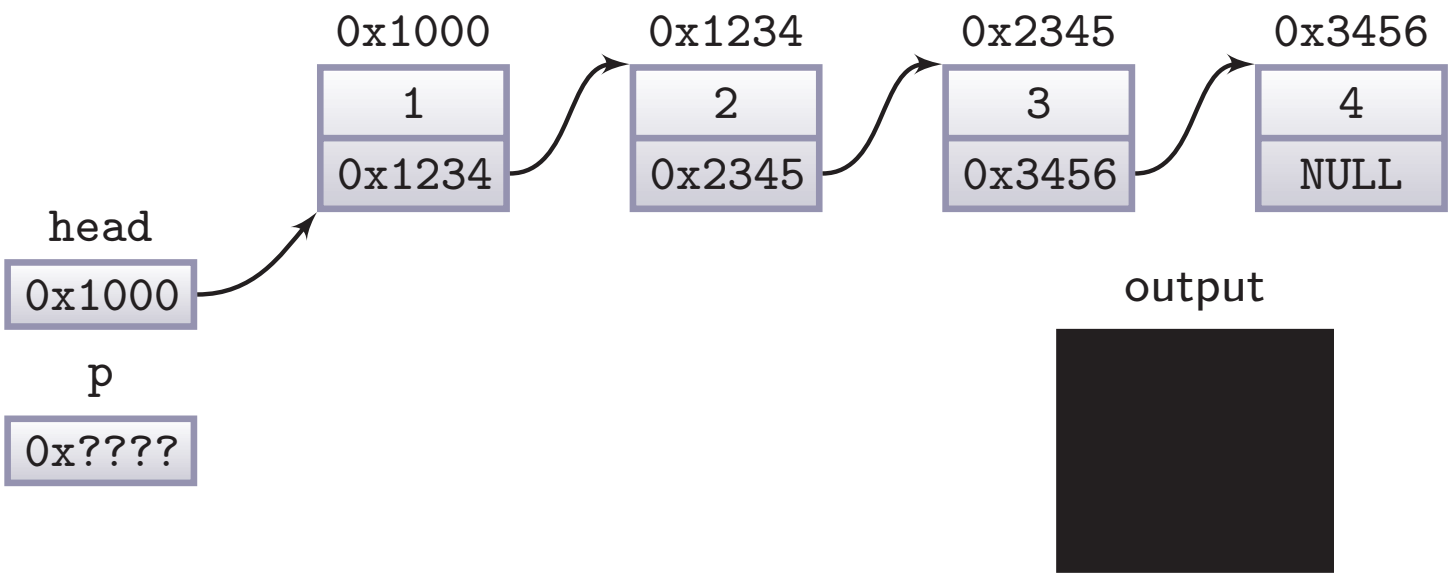


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

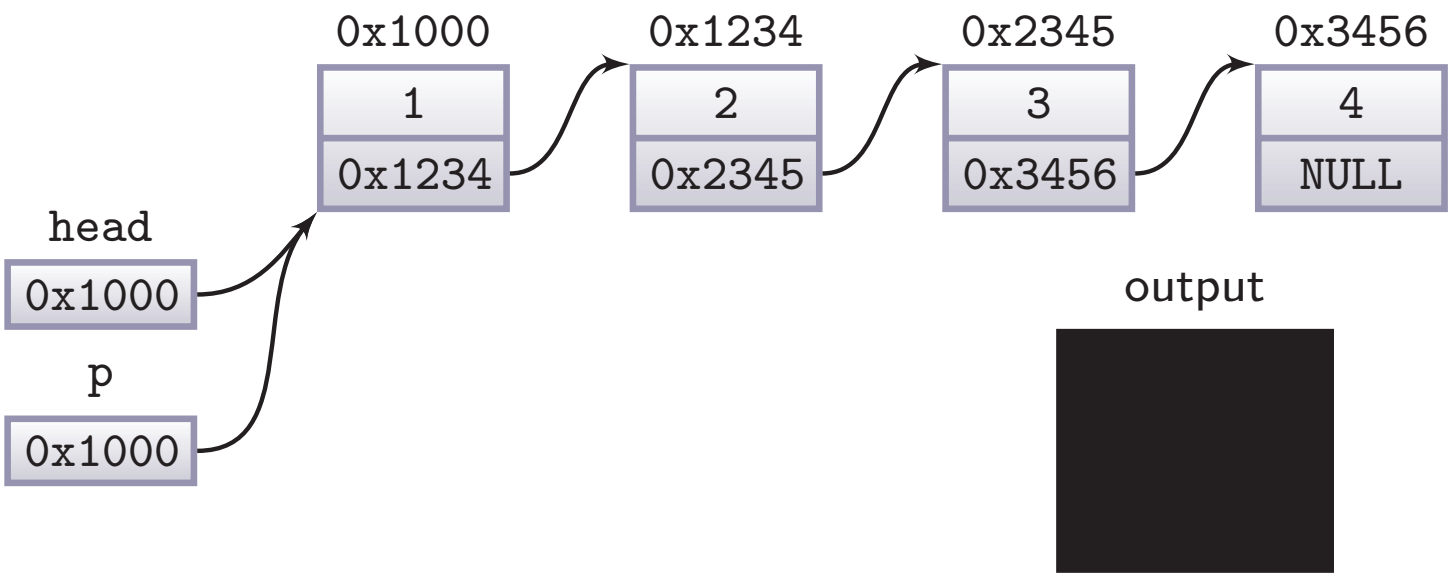


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

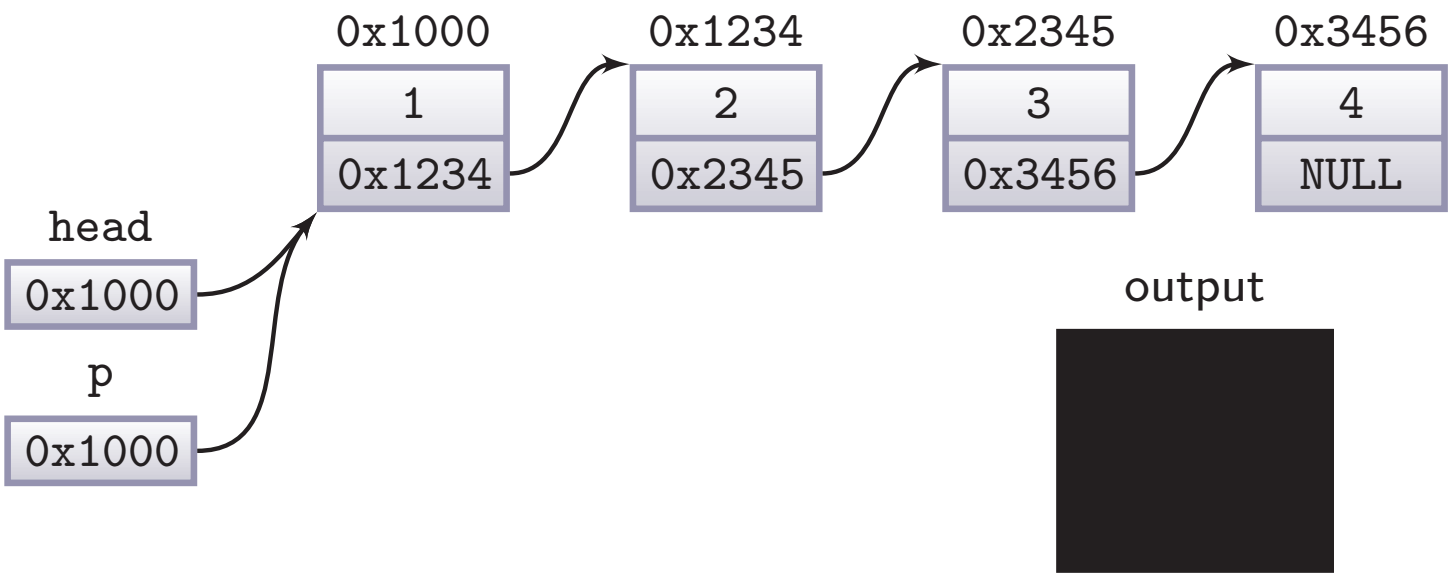


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

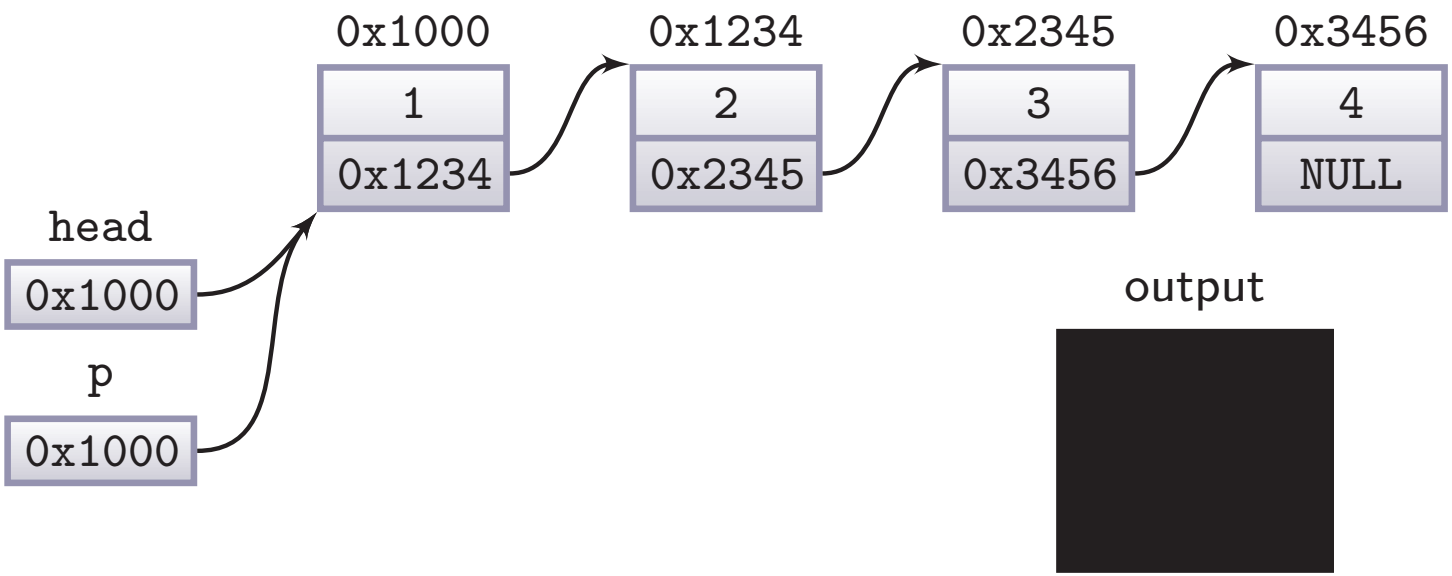


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

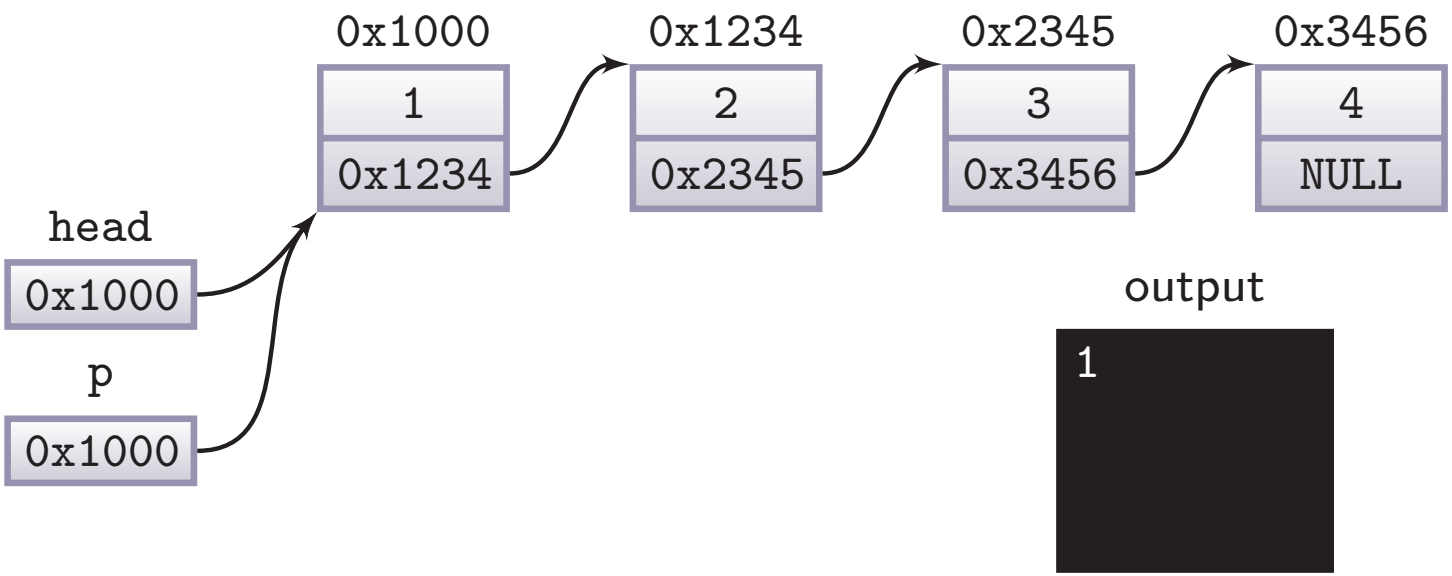


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

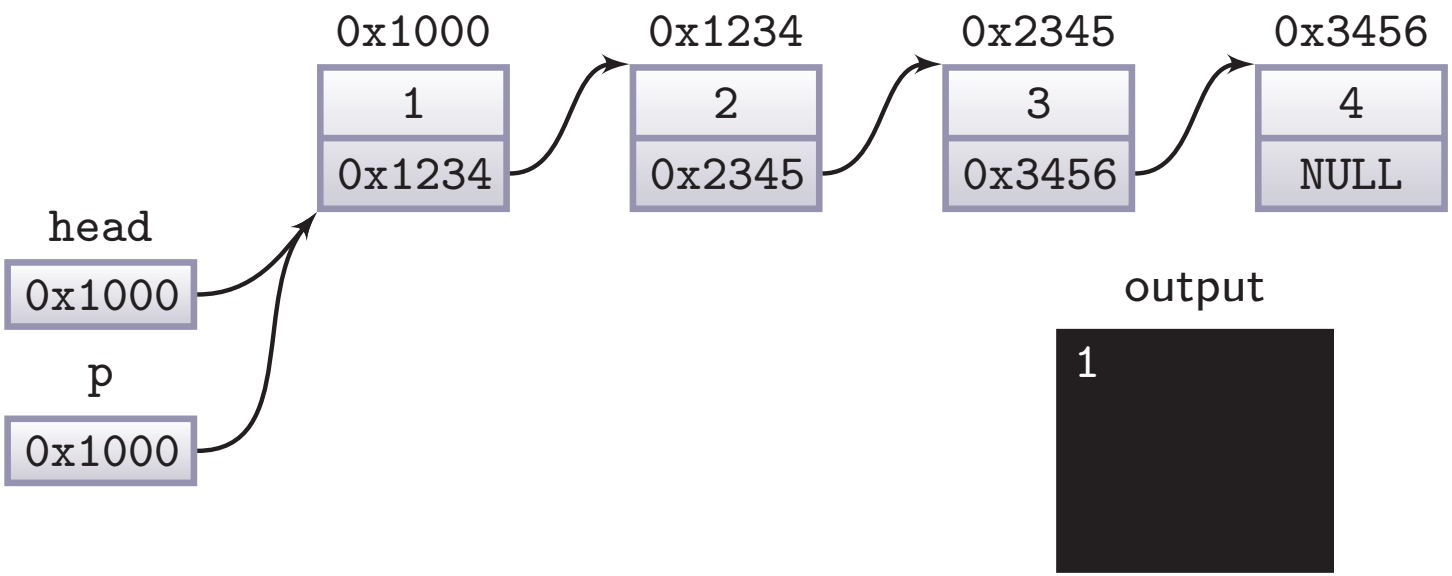


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

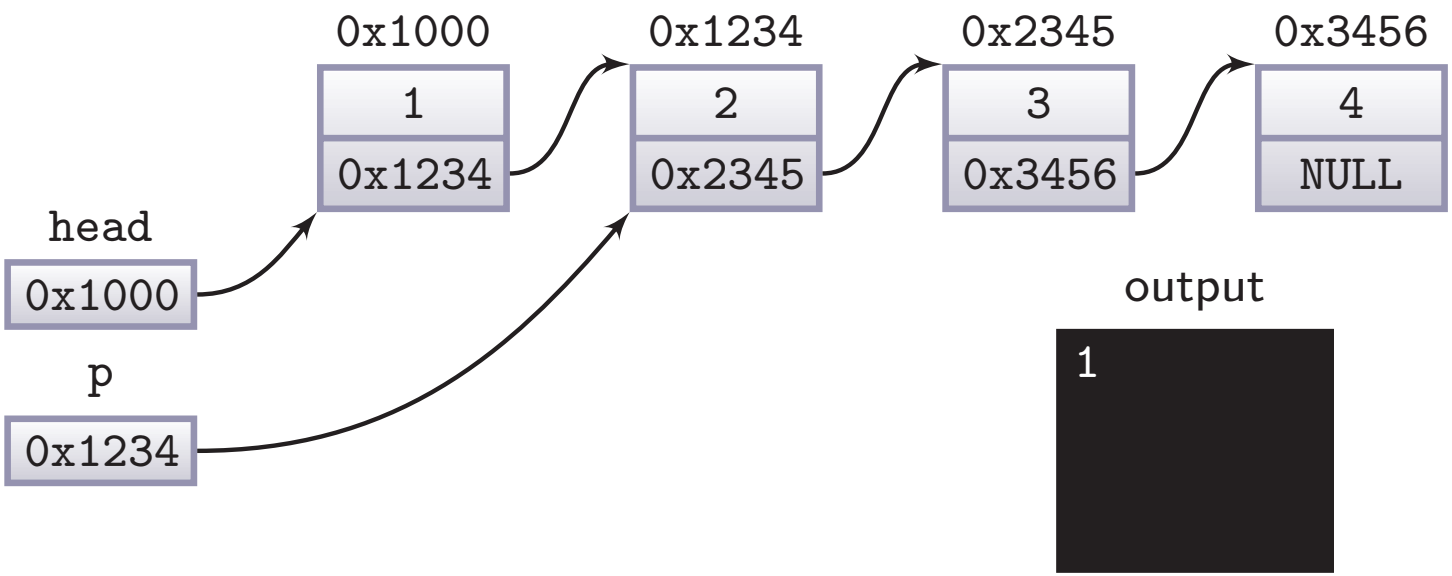


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

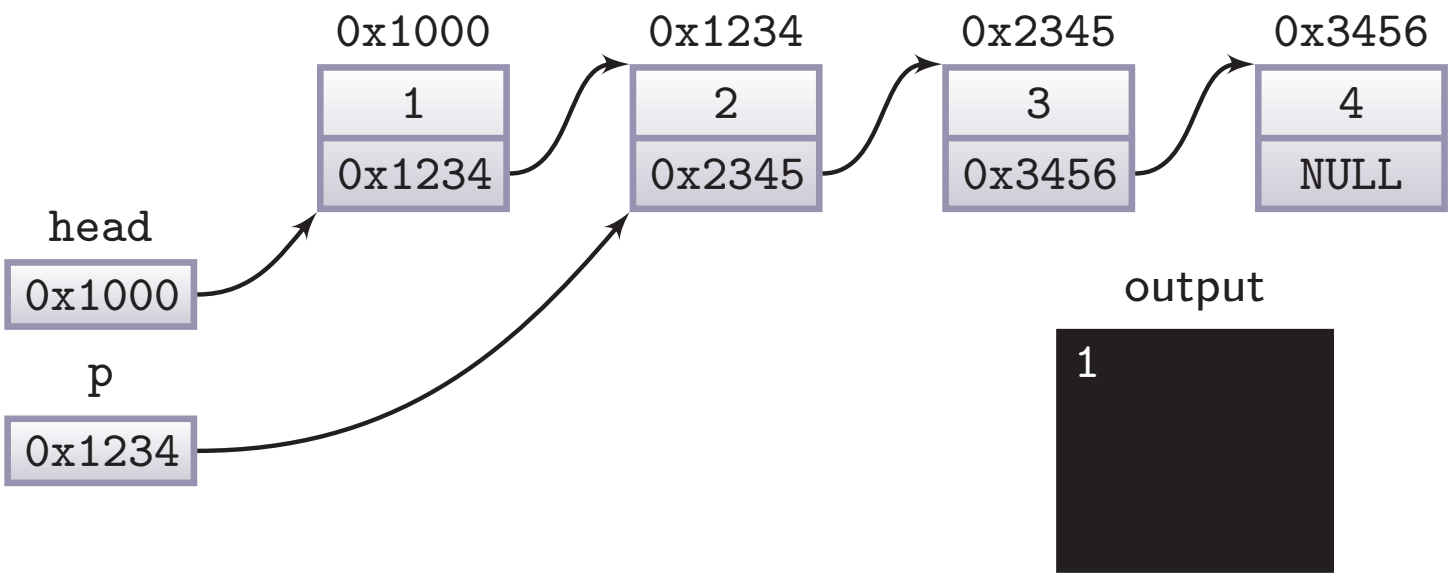


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

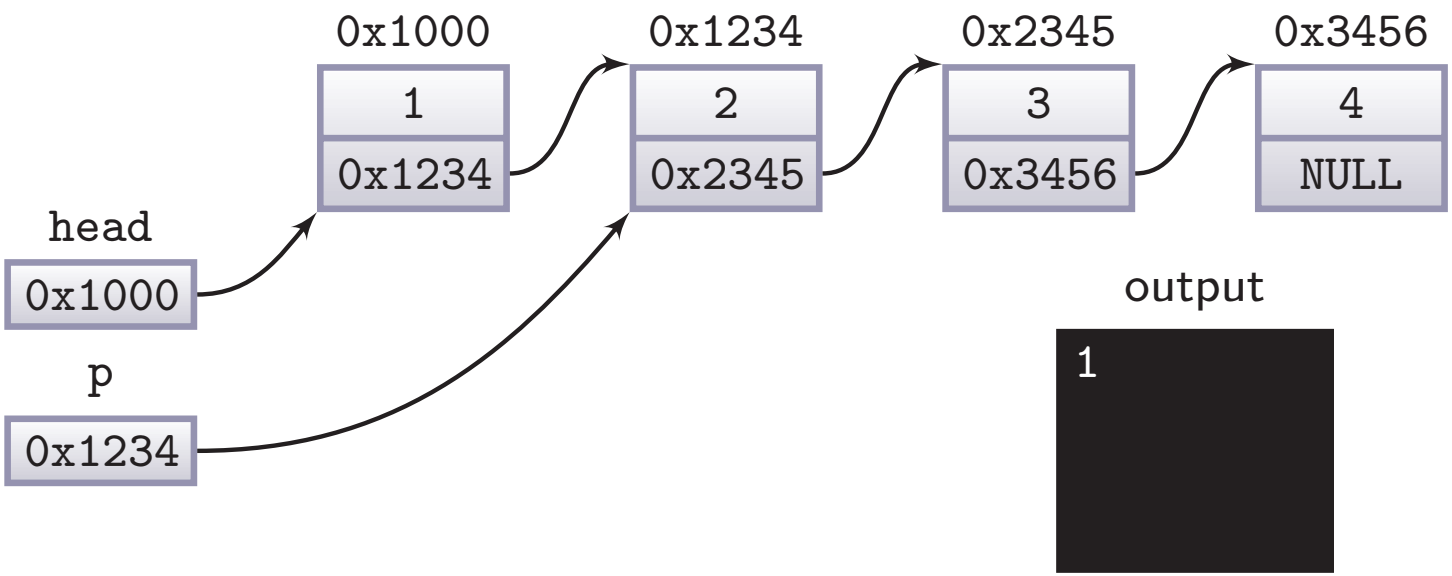


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

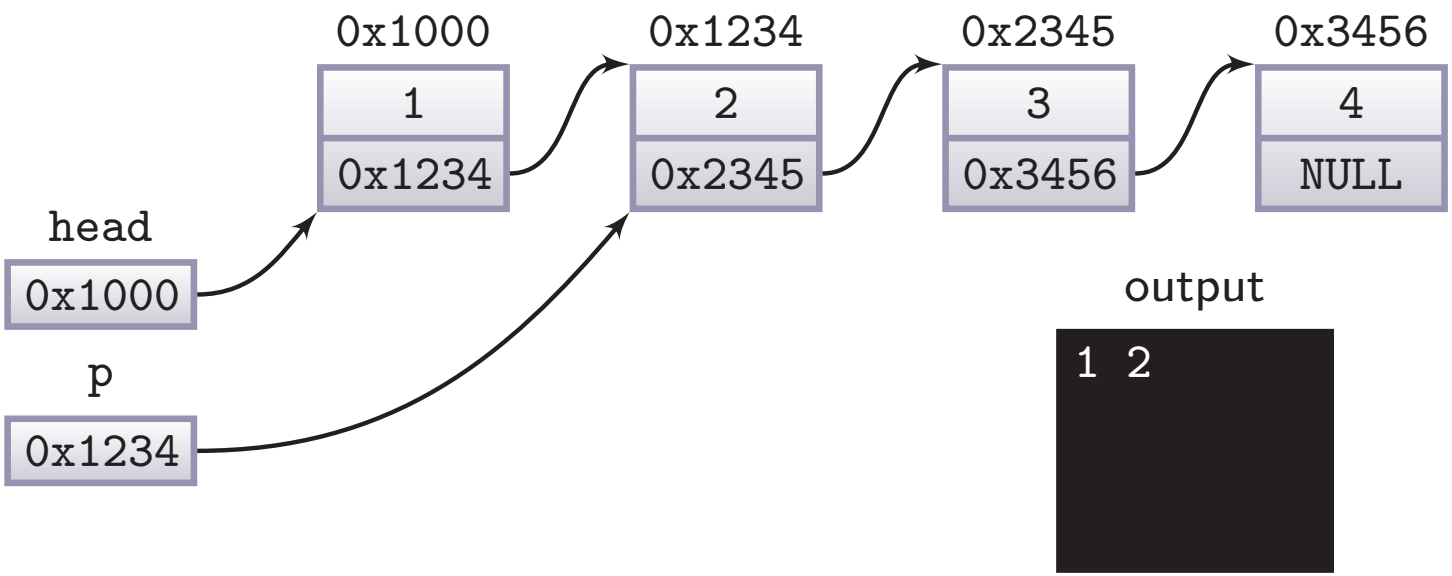


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

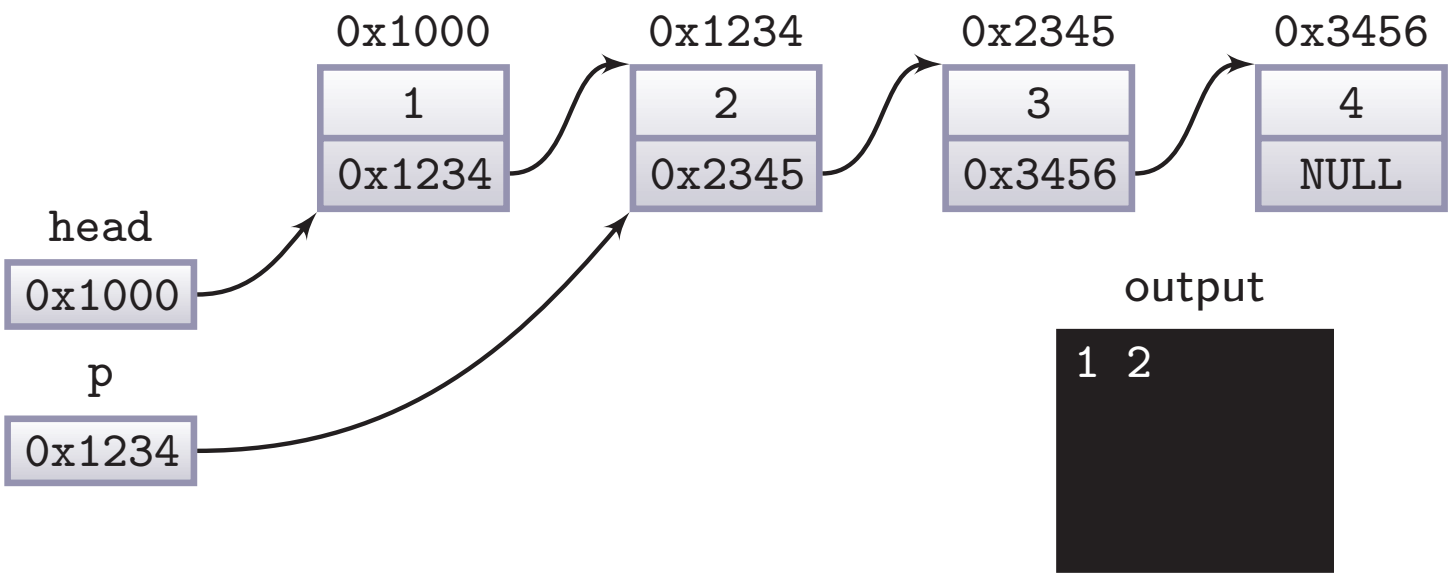


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

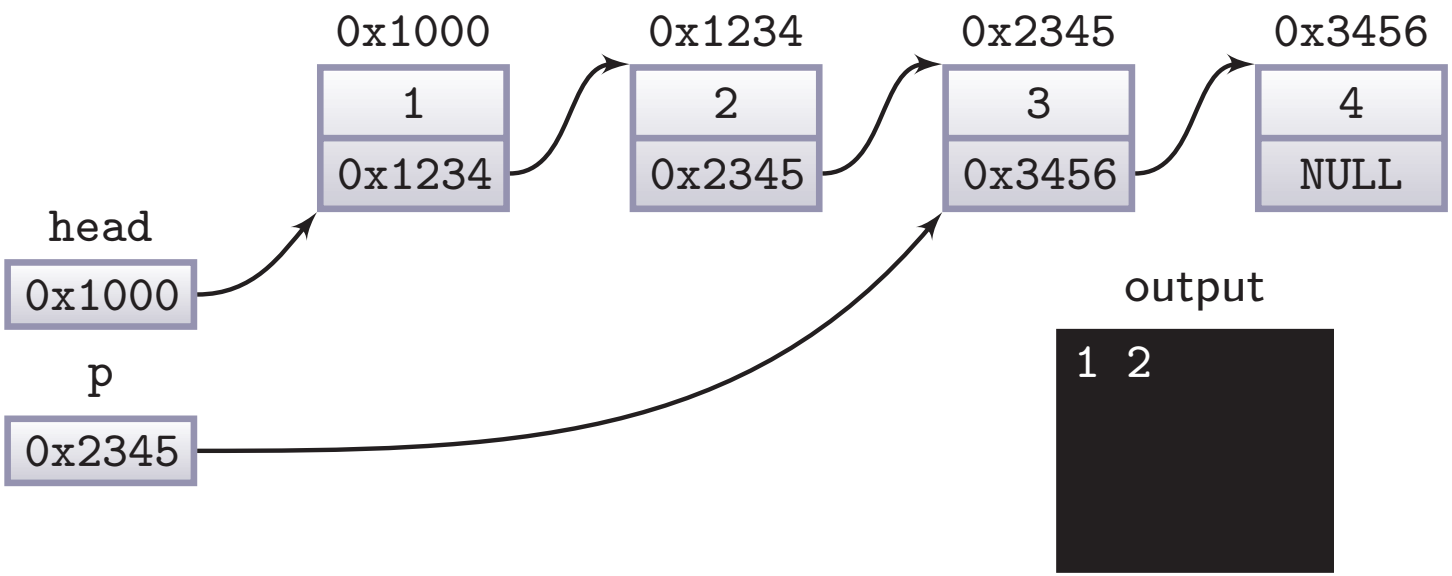


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

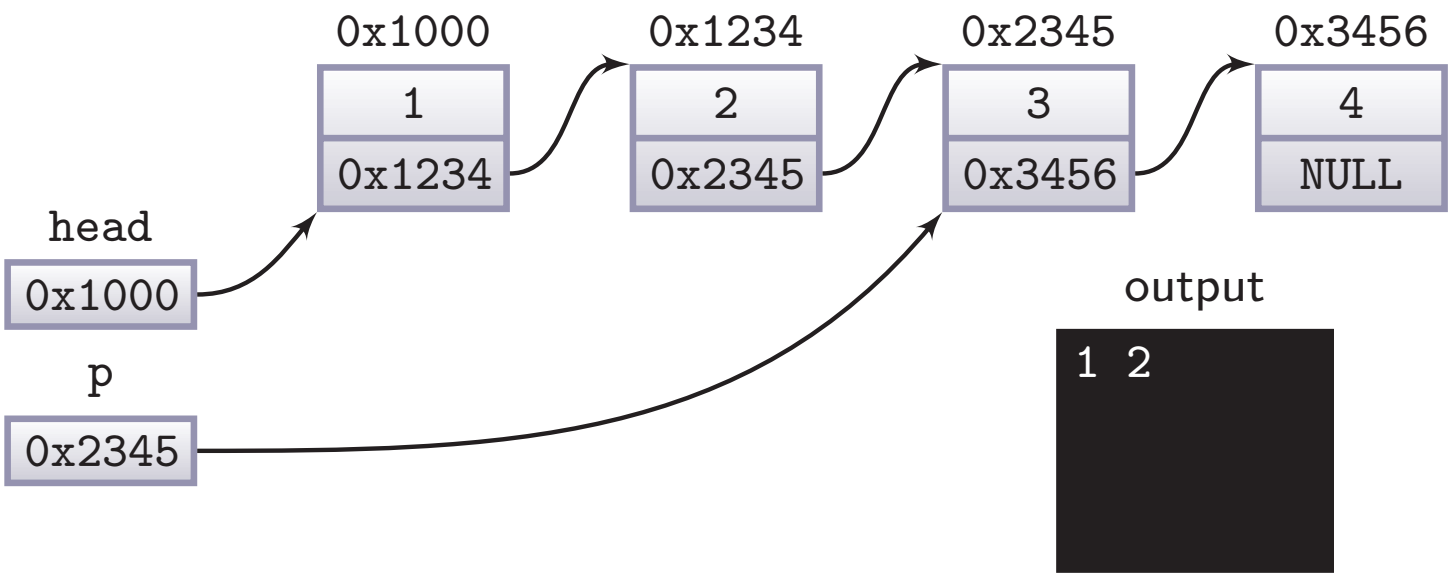


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

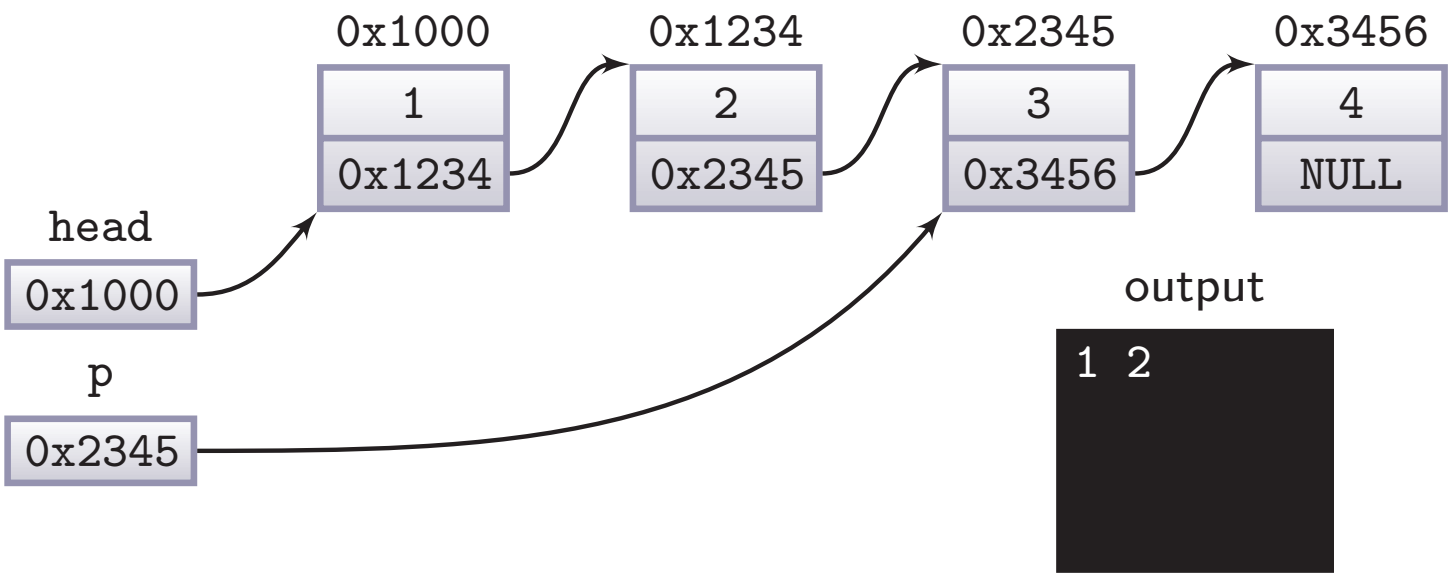


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

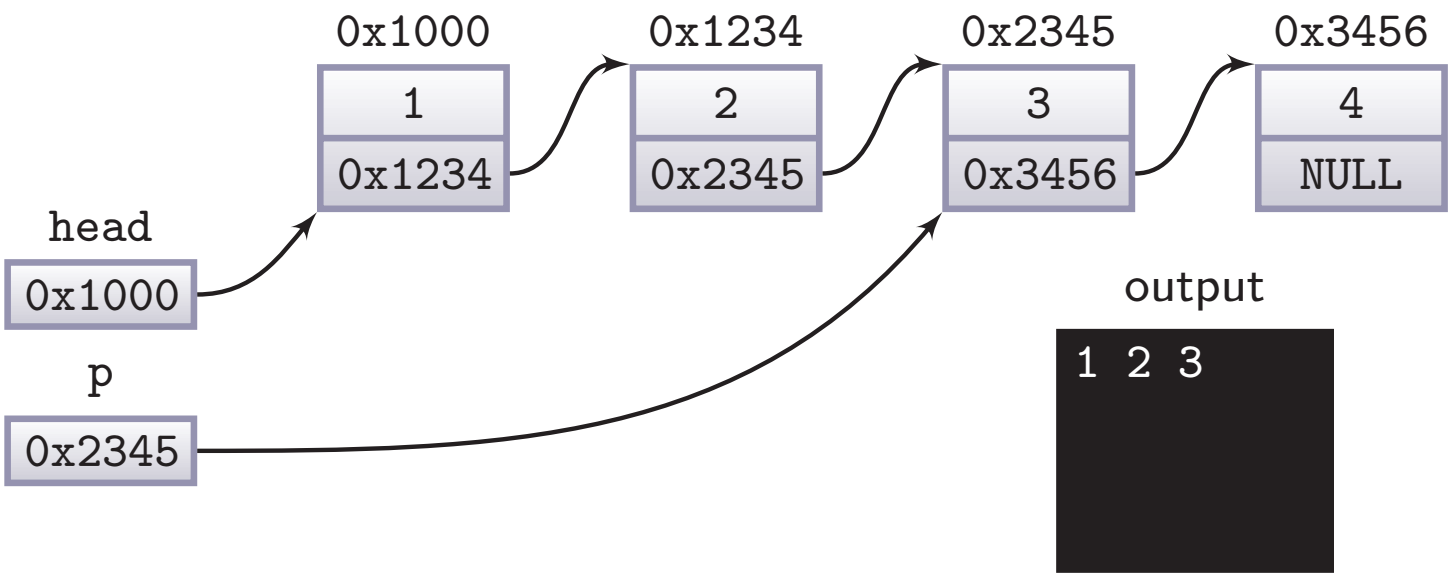


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

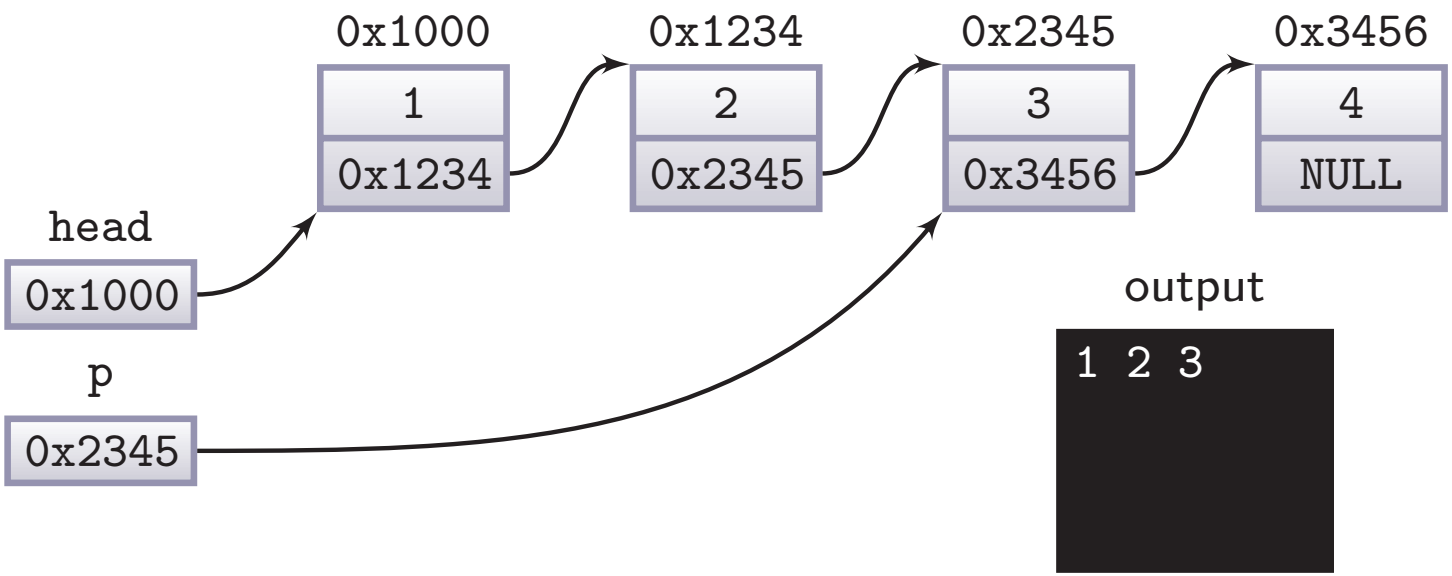


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

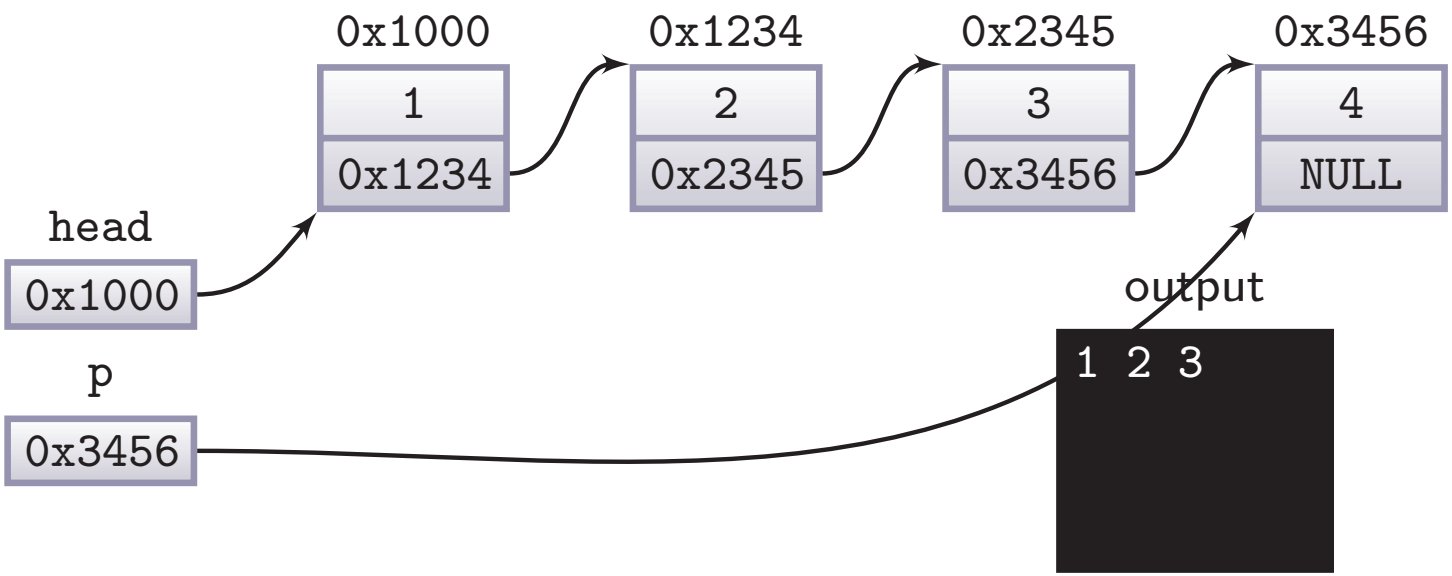


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

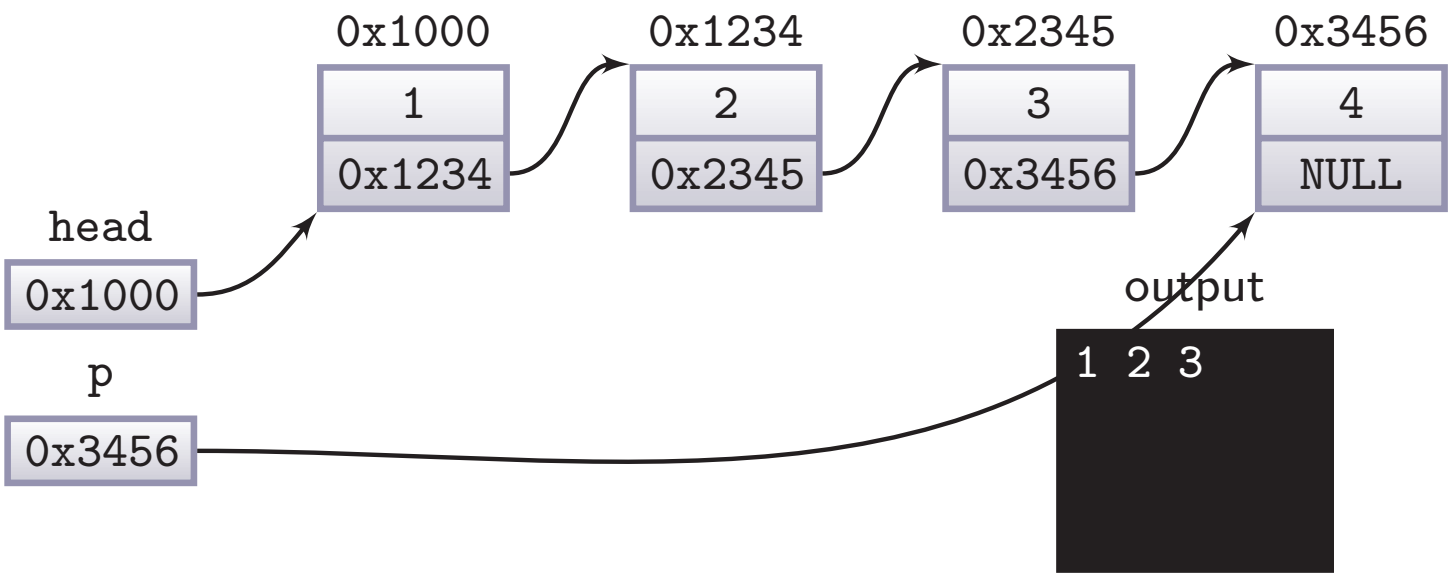


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

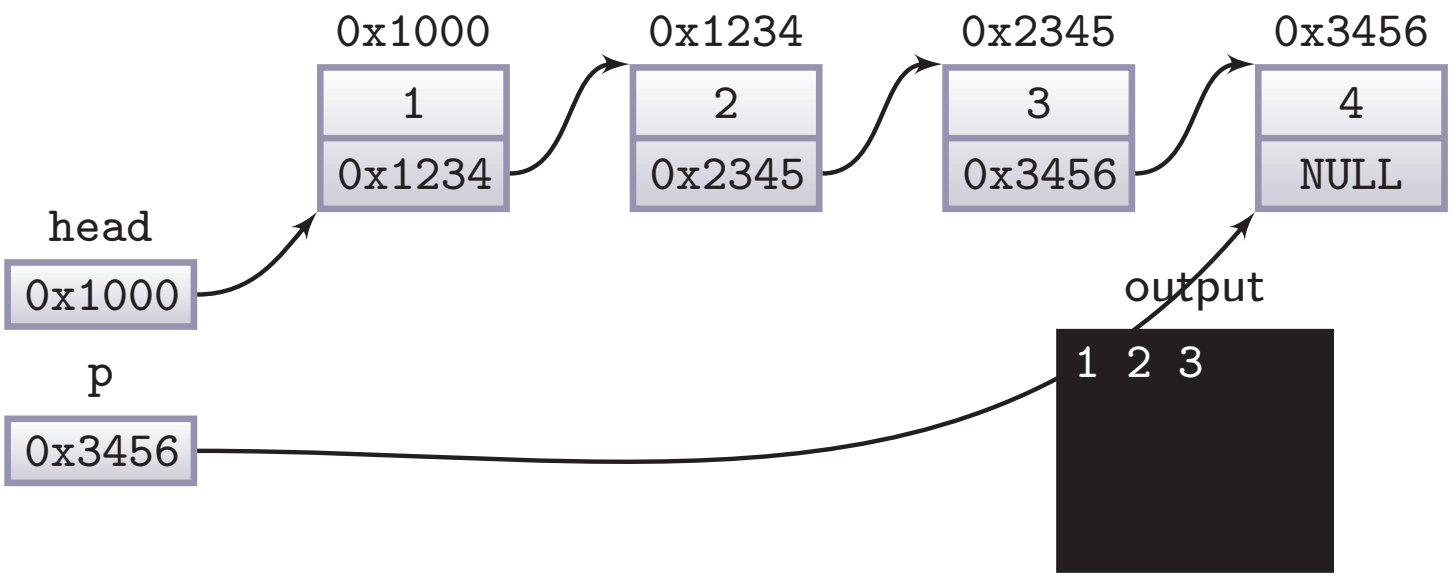


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

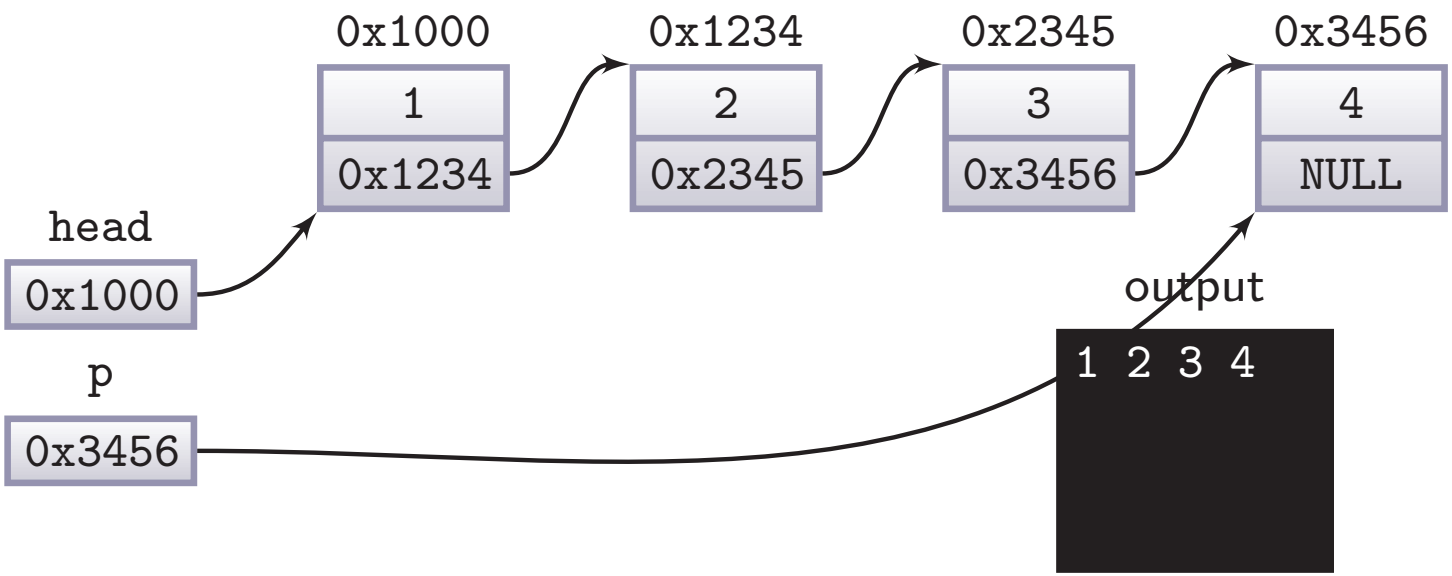


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

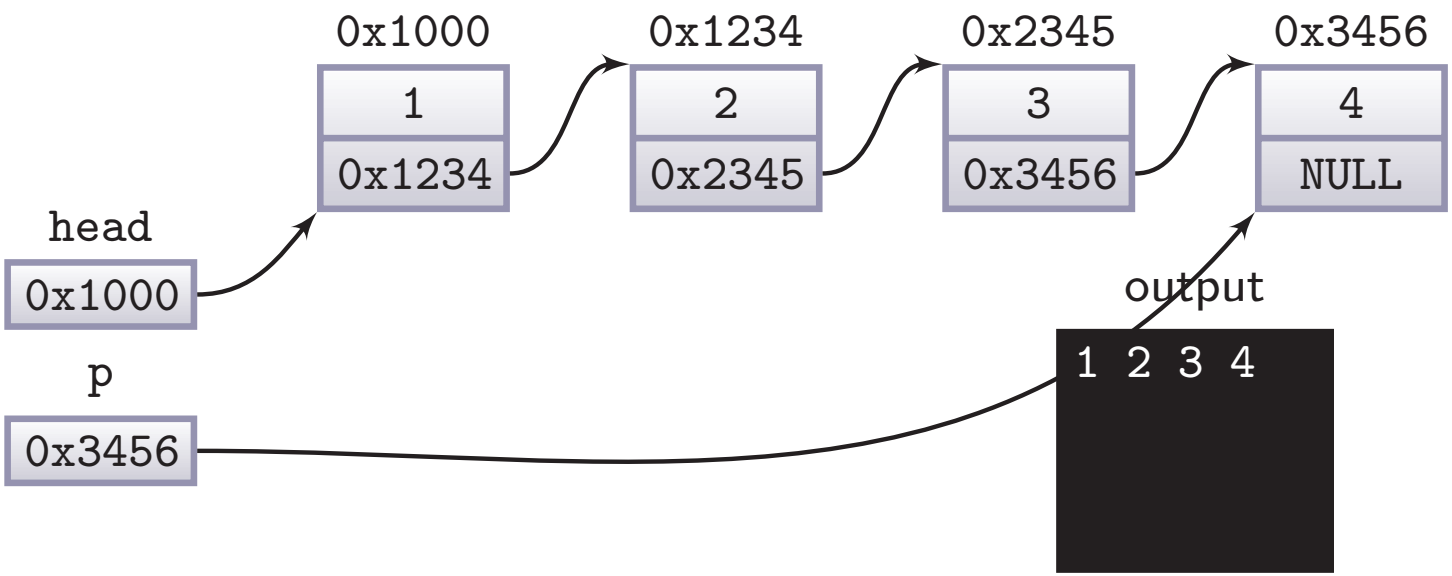


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

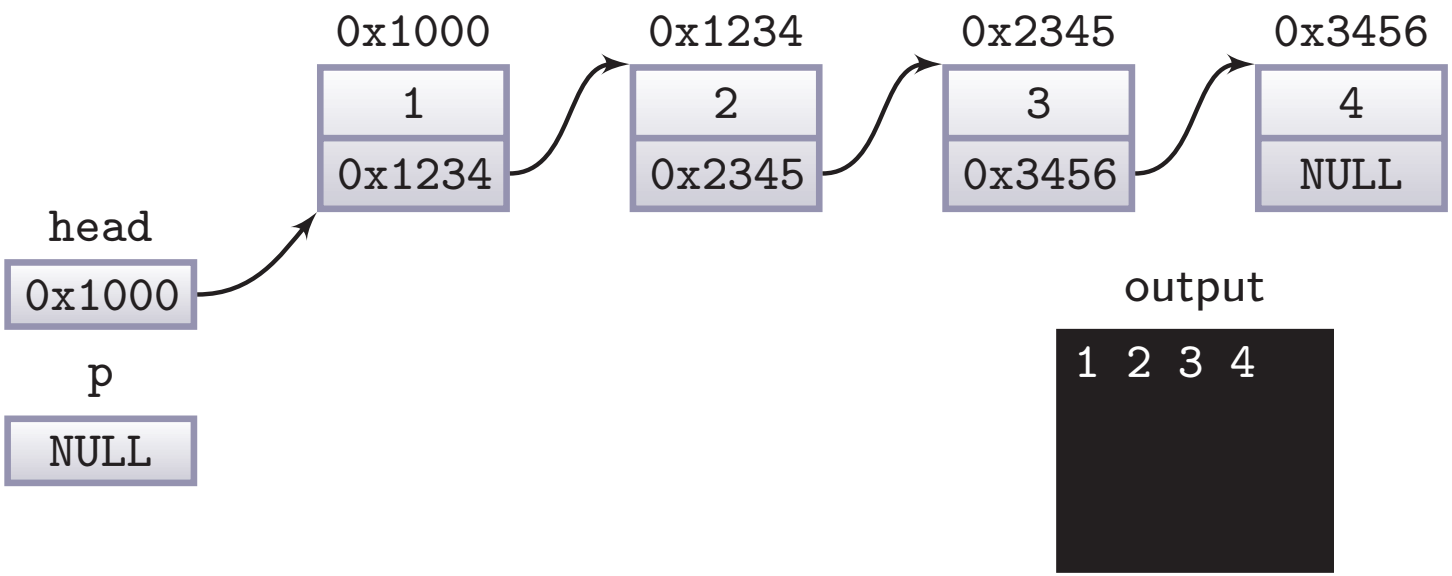


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

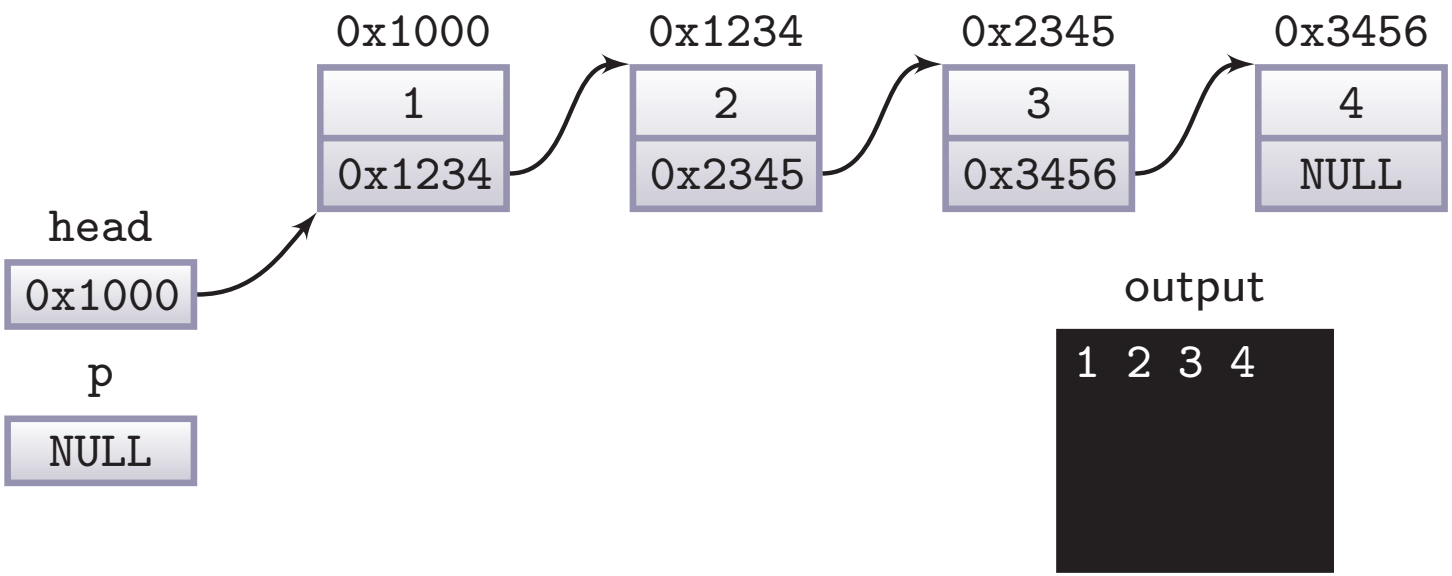


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```

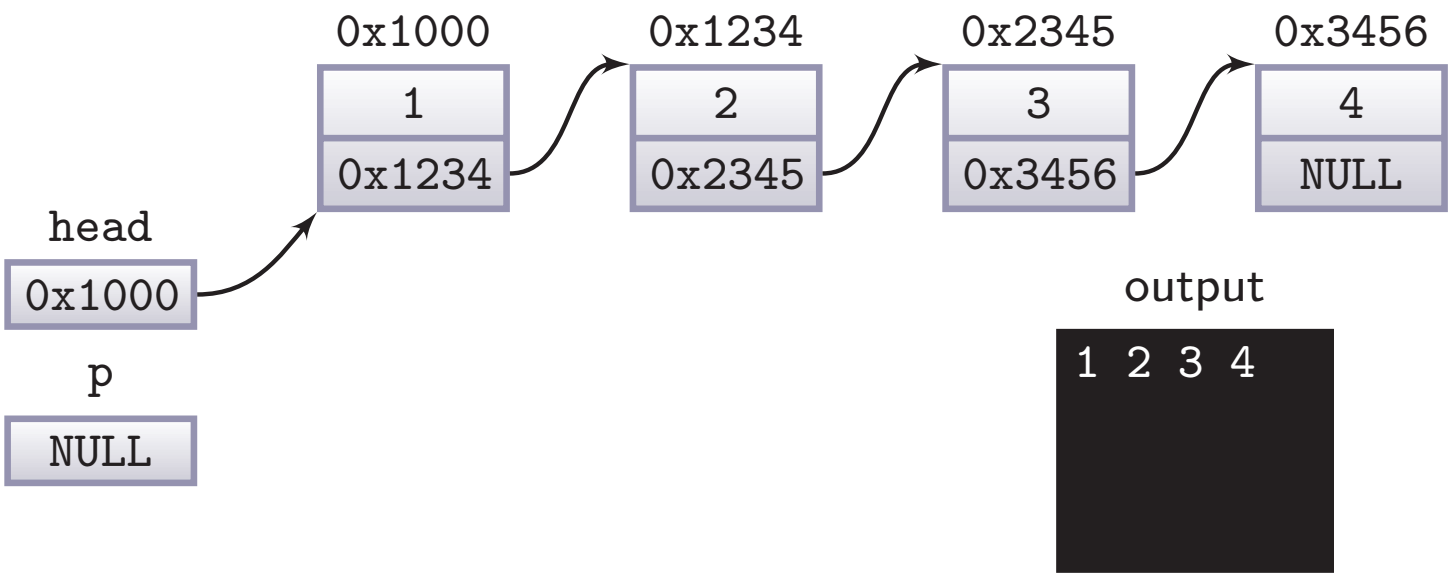


Traversing a list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- For traversing we need an auxiliary pointer (p), that will run along the list.

```
1  listelem *p = head;
2  while (p != NULL)
3  {
4    printf("%d ", p->data); /* p->data : (*p).data */
5    p = p->next;            /* arrow operator */
6  }
```



Passing a list to a function

- As a list is determined by its starting address, we only need to pass the startig address for the function

```
1  void traverse(listelem *head) {
2    listelem *p = head;
3    while (p != NULL)
4    {
5      printf("%d ", p->data);
6      p = p->next;
7    }
8  }
```

link

Passing a list to a function

- As a list is determined by its starting address, we only need to pass the startig address for the function

```c
void traverse(listelem *head) {
  listelem *p = head;
  while (p != NULL)
  {
    printf("%d ", p->data);
    p = p->next;
  }
}
```

link

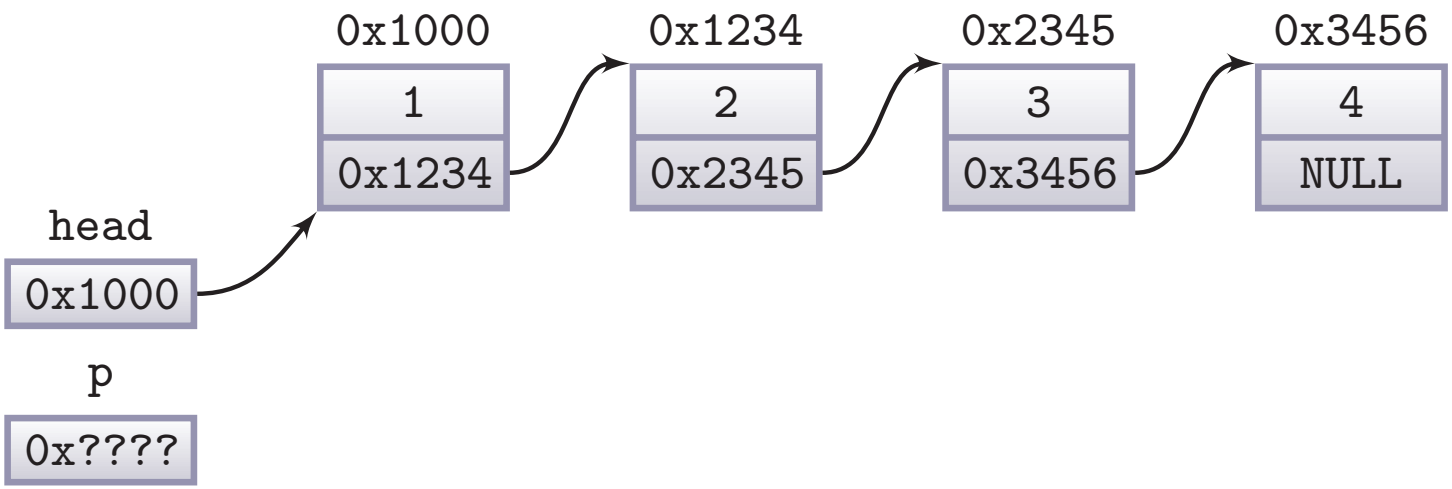
- the same with `for` loop

```c
void traverse(listelem *head) {
  listelem *p;
  for (p = head; p != NULL; p = p->next)
    printf("%d ", p->data);
}
```

Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

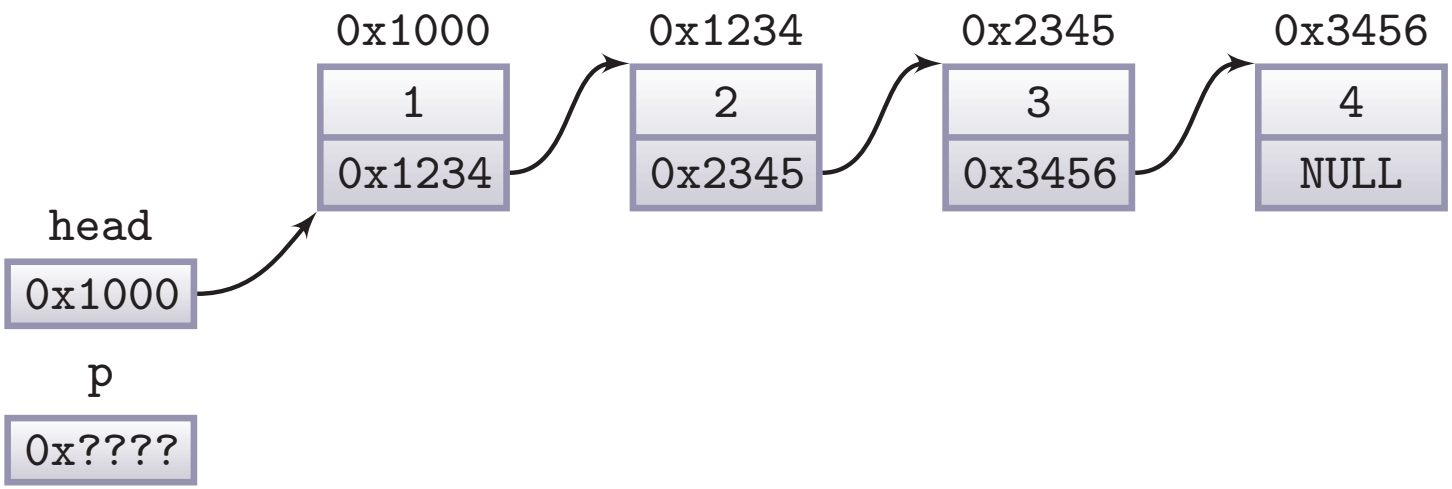
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

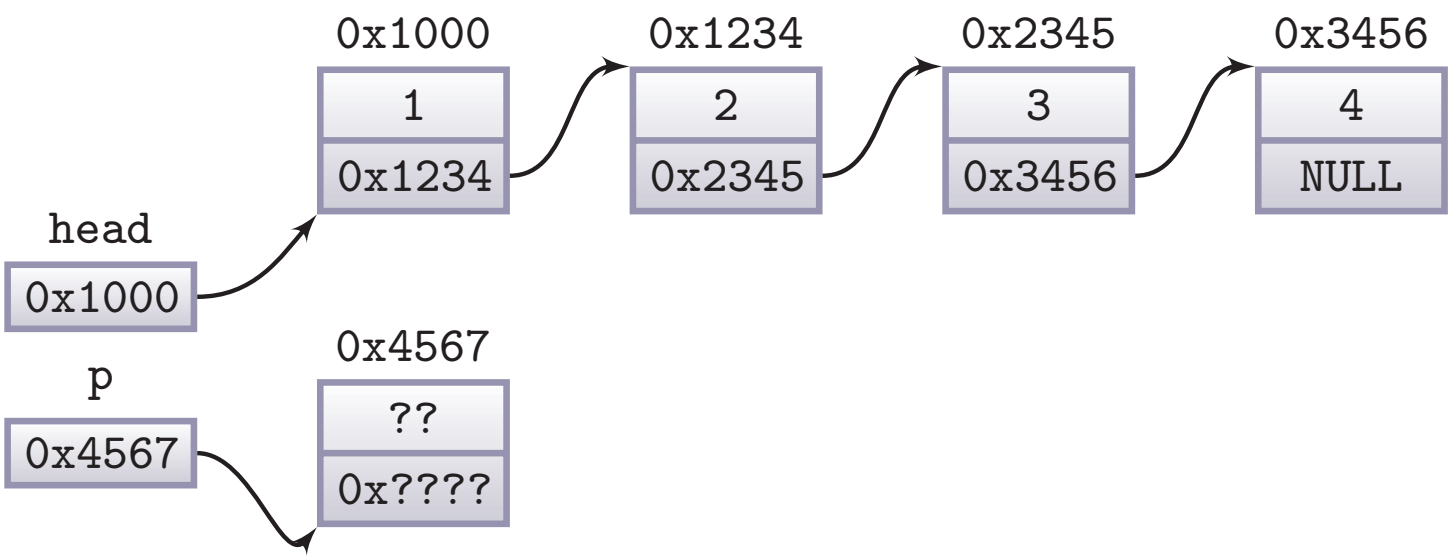
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

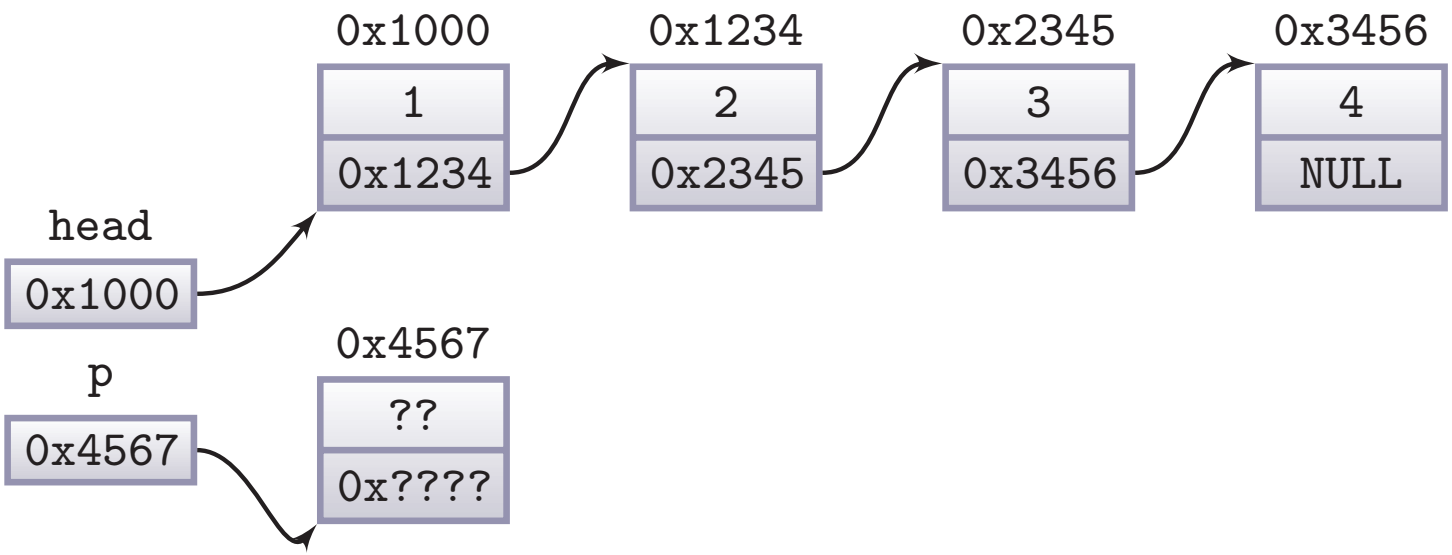
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

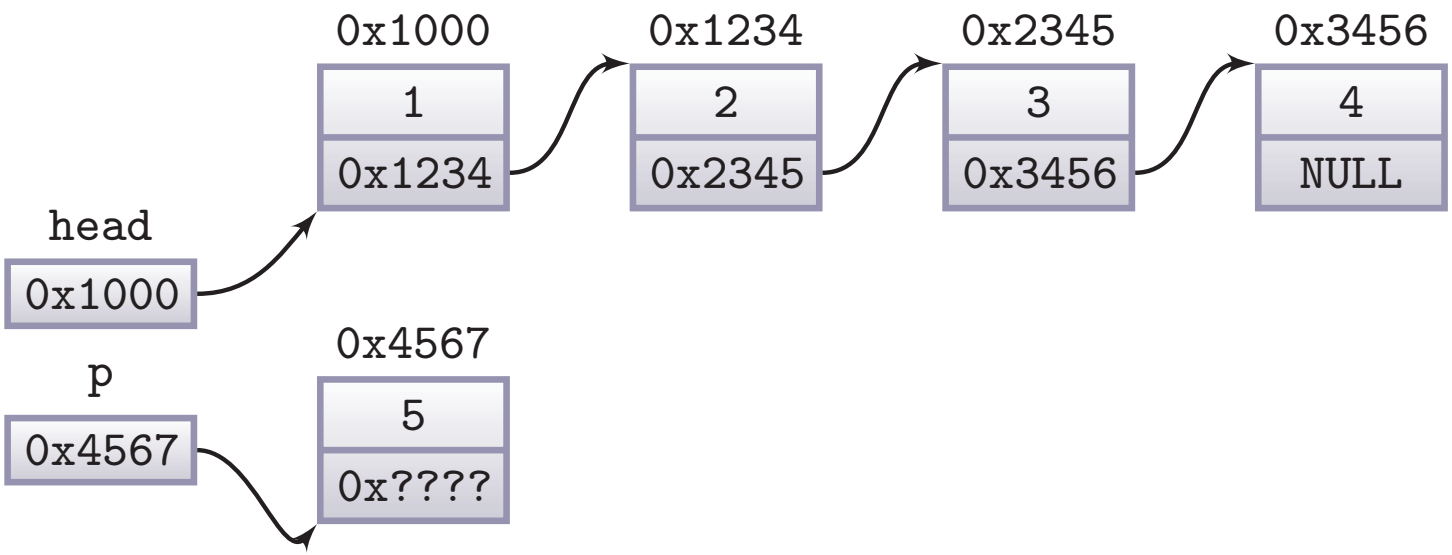
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

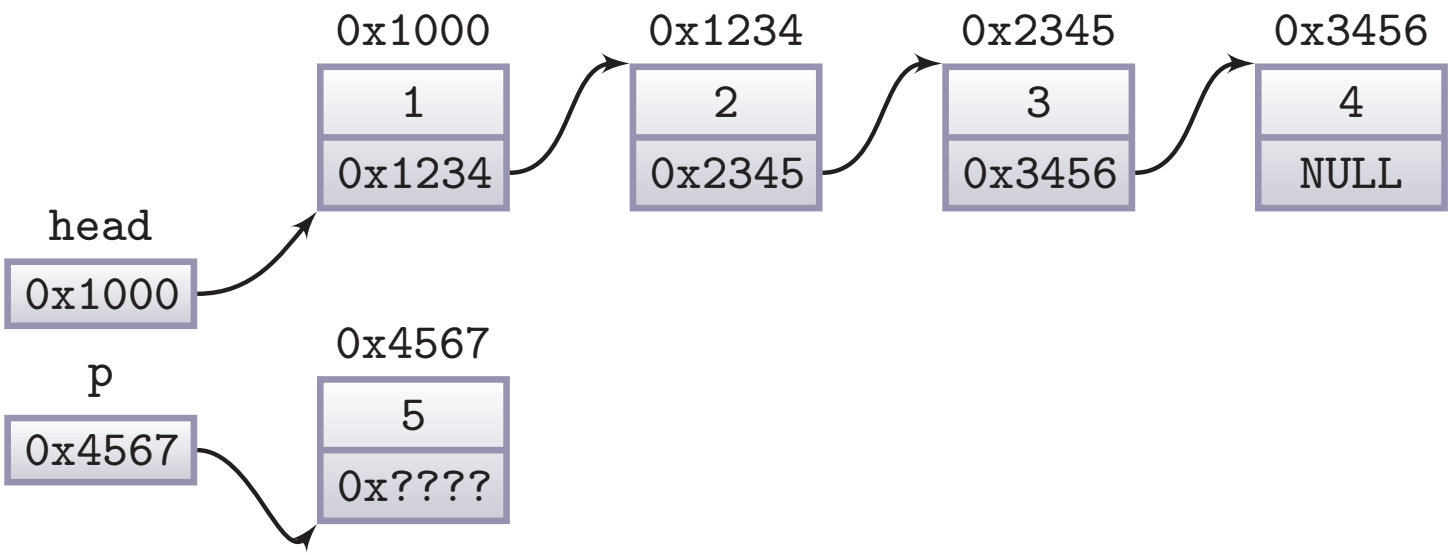
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

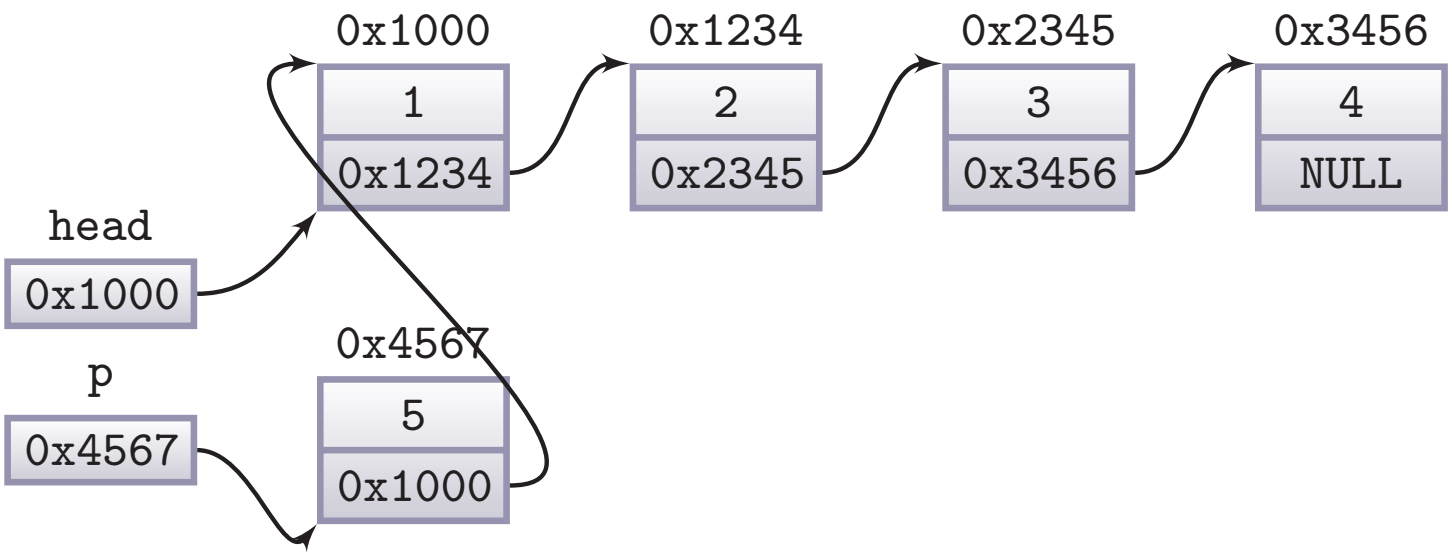
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

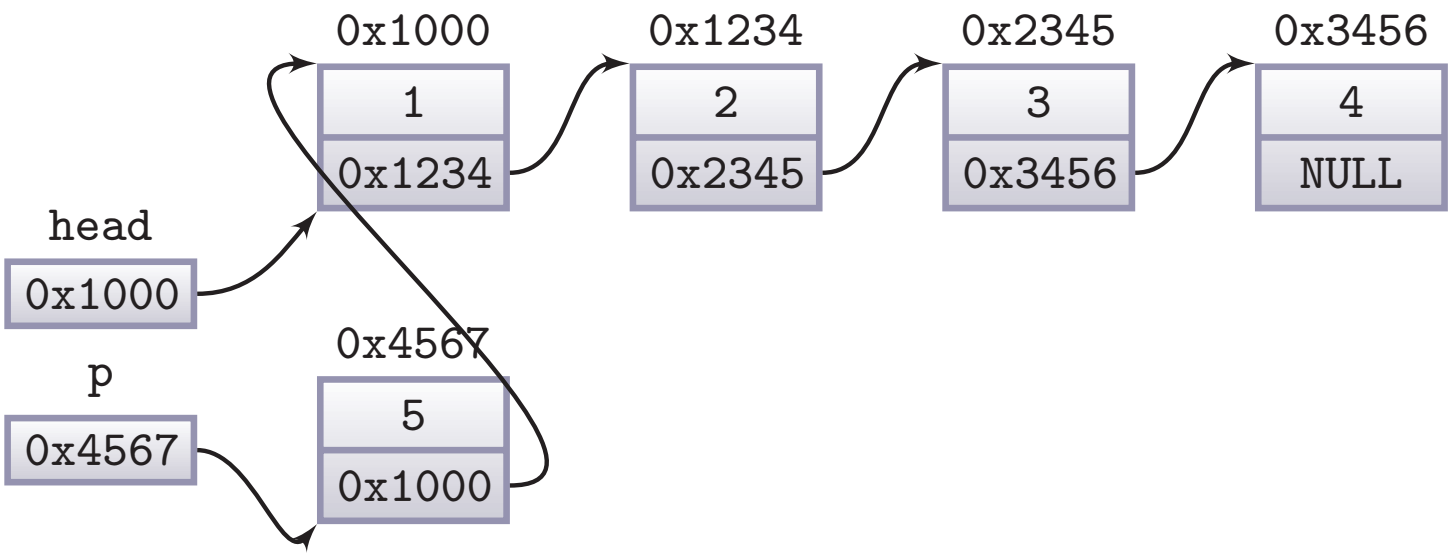
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

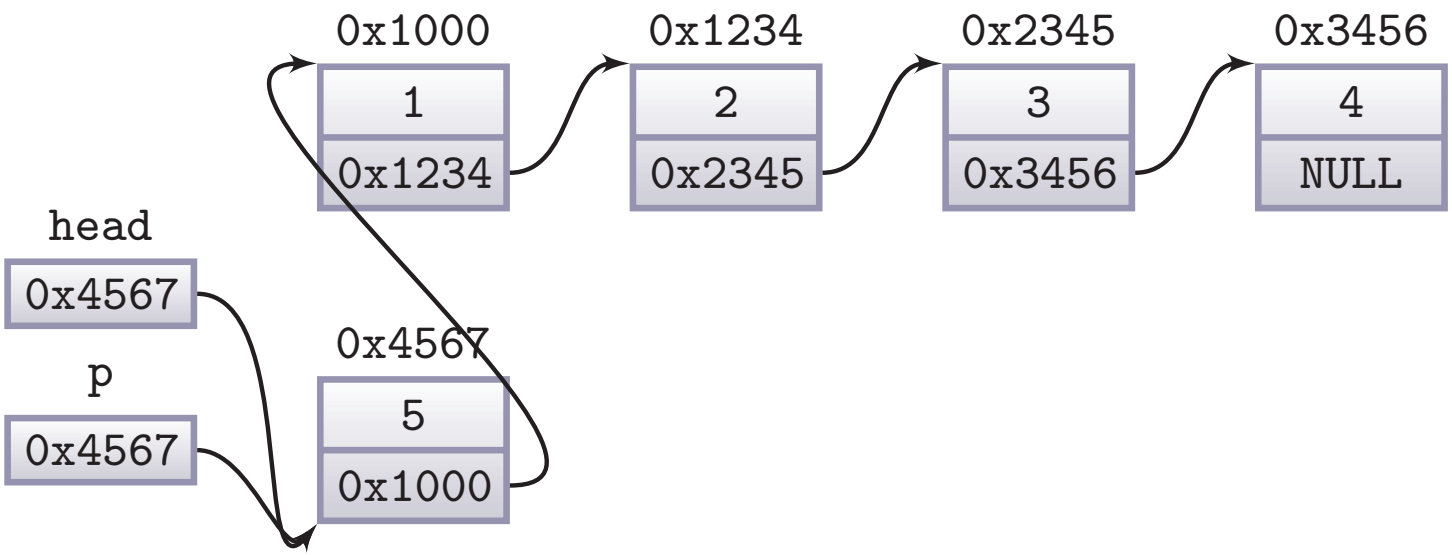
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

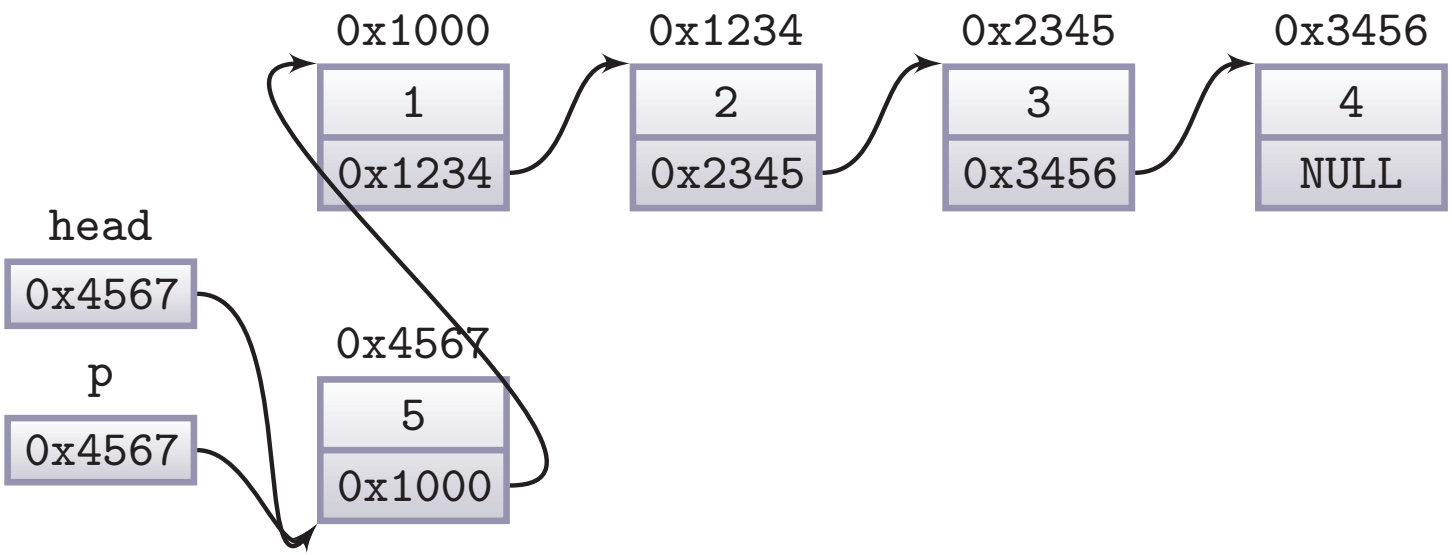
```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  p = (listelem*)malloc(sizeof(listelem));
2  p->data = 5;
3  p->next = head;
4  head = p;
```



Inserting element to the front of the list, with a function

- As the starting address is changed when inserting, we have to return it (pass it back)

```
1  listelem *push_front(listelem *head, int d)
2  {
3    listelem *p = (listelem*)malloc(sizeof(listelem));
4    p->data = d;
5    p->next = head;
6    head = p;
7    return head;
8  }
```

link

Inserting element to the front of the list, with a function

- As the starting address is changed when inserting, we have to return it (pass it back)

```
1  listelem *push_front(listelem *head, int d)
2  {
3    listelem *p = (listelem*)malloc(sizeof(listelem));
4    p->data = d;
5    p->next = head;
6    head = p;
7    return head;
8  }
```

link

- Usage of function

```
1  listelem *head = NULL;          /* empty list */
2  head = push_front(head, 2);     /* head is changed! */
3  head = push_front(head, 4);
```

Inserting element to the front of the list, with a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Another option is to pass the starting address by its address

```
1  void push_front(listelem **head, int d)
2  {
3    listelem *p = (listelem*)malloc(sizeof(listelem));
4    p->data = d;
5    p->next = *head;
6    *head = p; /* *head is changes, this is not lost */
7  }
```

link

Inserting element to the front of the list, with a function

- Another option is to pass the starting address by its address

```
1  void push_front(listelem **head, int d)
2  {
3    listelem *p = (listelem*)malloc(sizeof(listelem));
4    p->data = d;
5    p->next = *head;
6    *head = p; /* *head is changes, this is not lost */
7  }
```

link

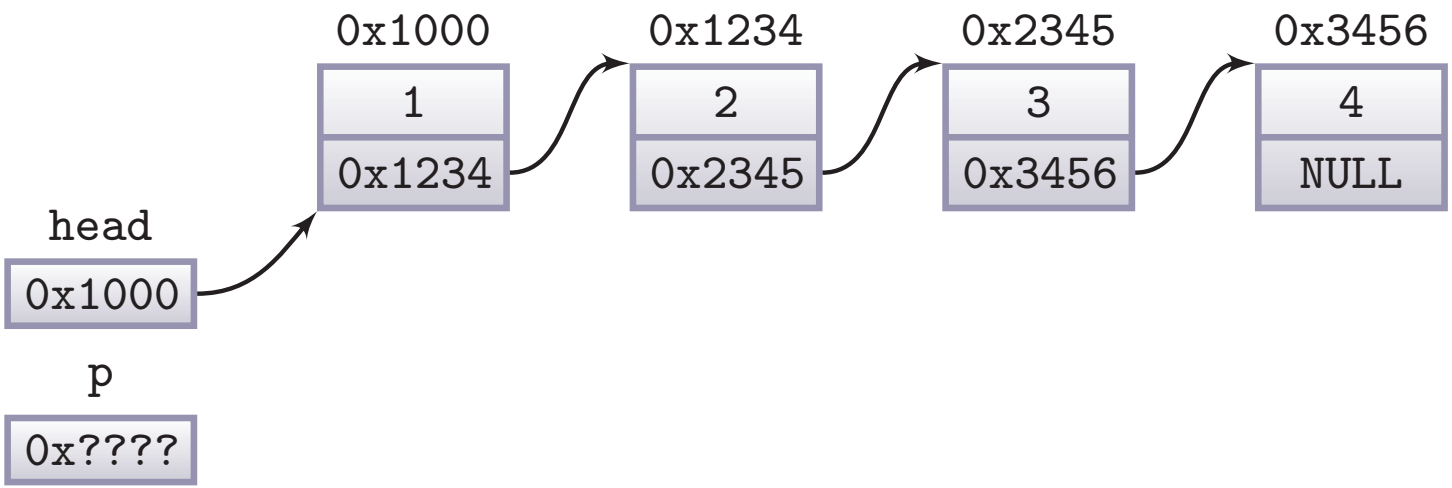
- In this case the usage of the function is:

```
1  listelem *head = NULL;   /* empty list */
2  push_front(&head, 2);    /* calling with address */
3  push_front(&head, 4);
```

Deleting element from the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

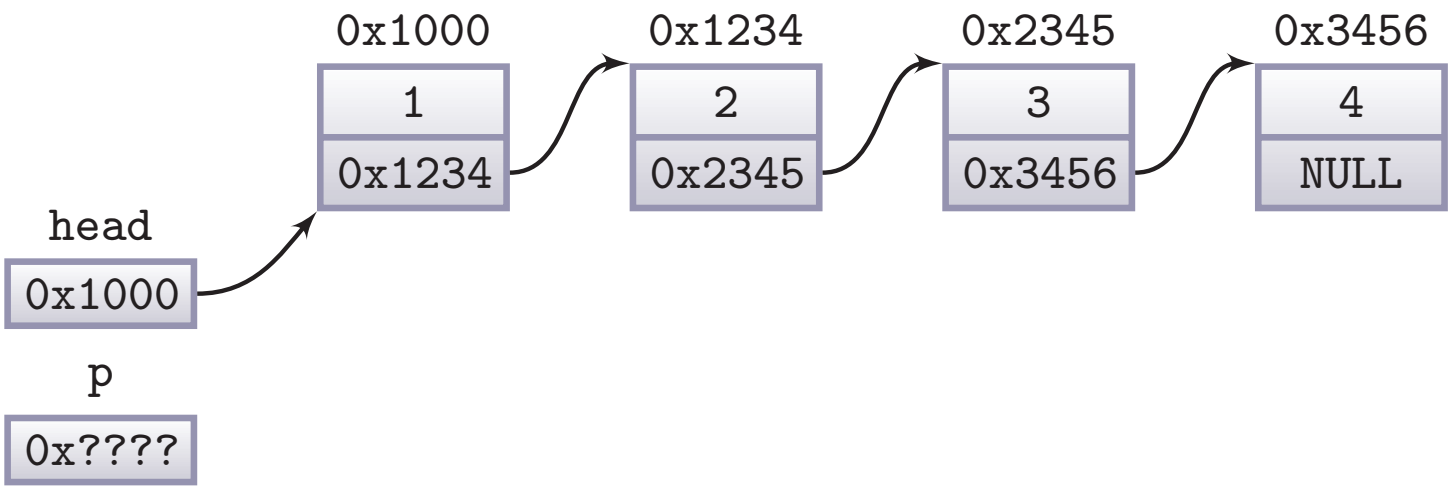
```
1  p = head;
2  head = head->next;
3  free(p);
```



Deleting element from the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

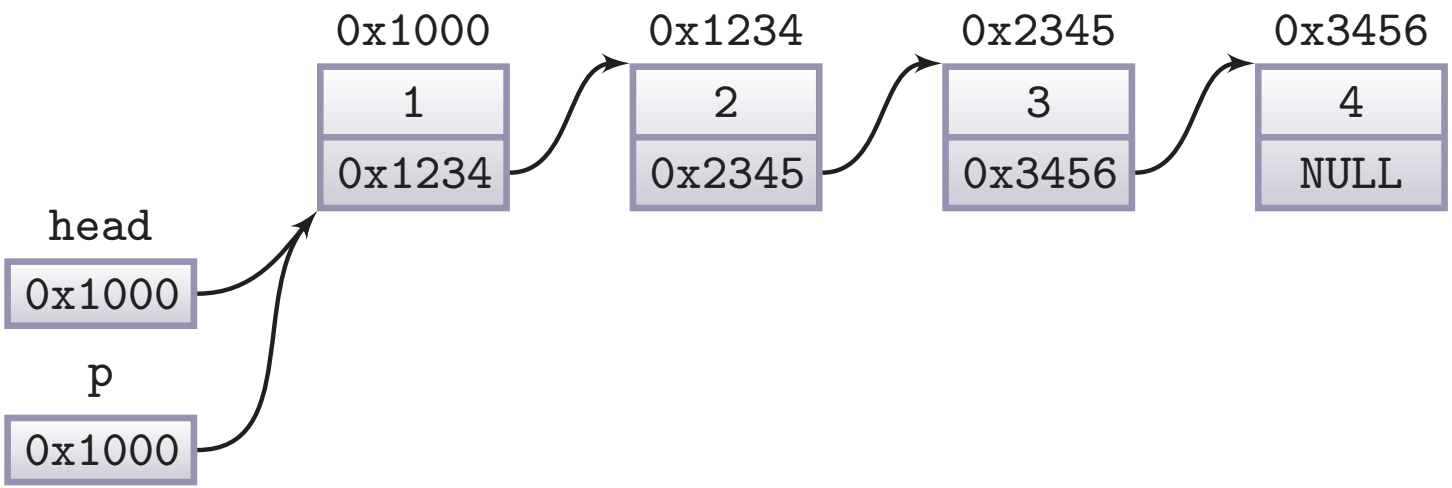
```
1  p = head;
2  head = head->next;
3  free(p);
```



Deleting element from the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

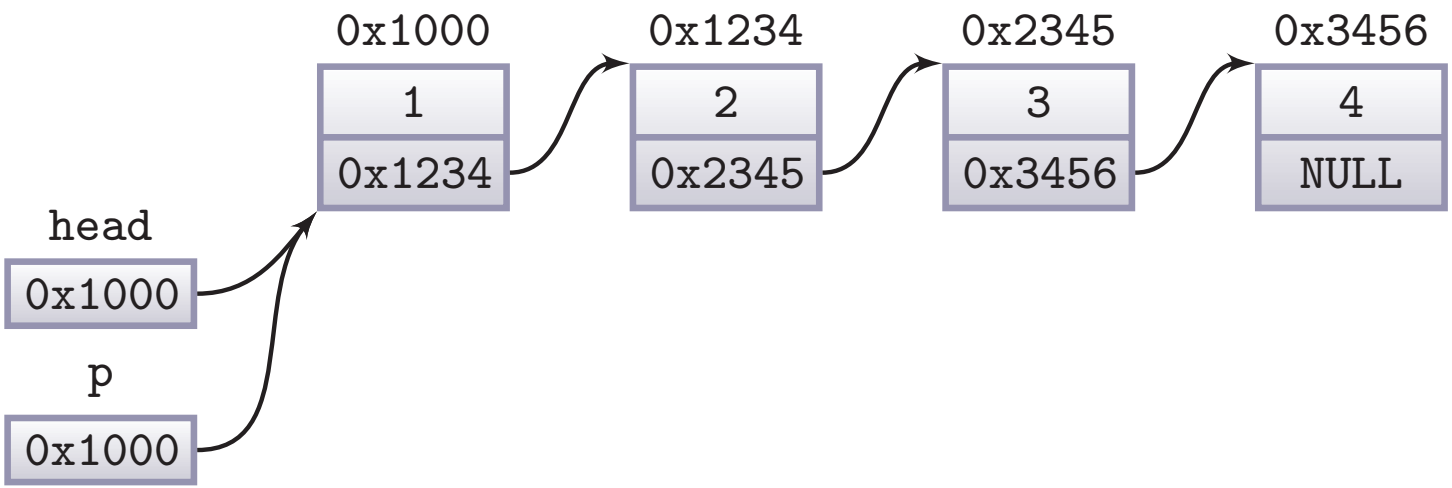
```
1  p = head;
2  head = head->next;
3  free(p);
```



Deleting element from the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

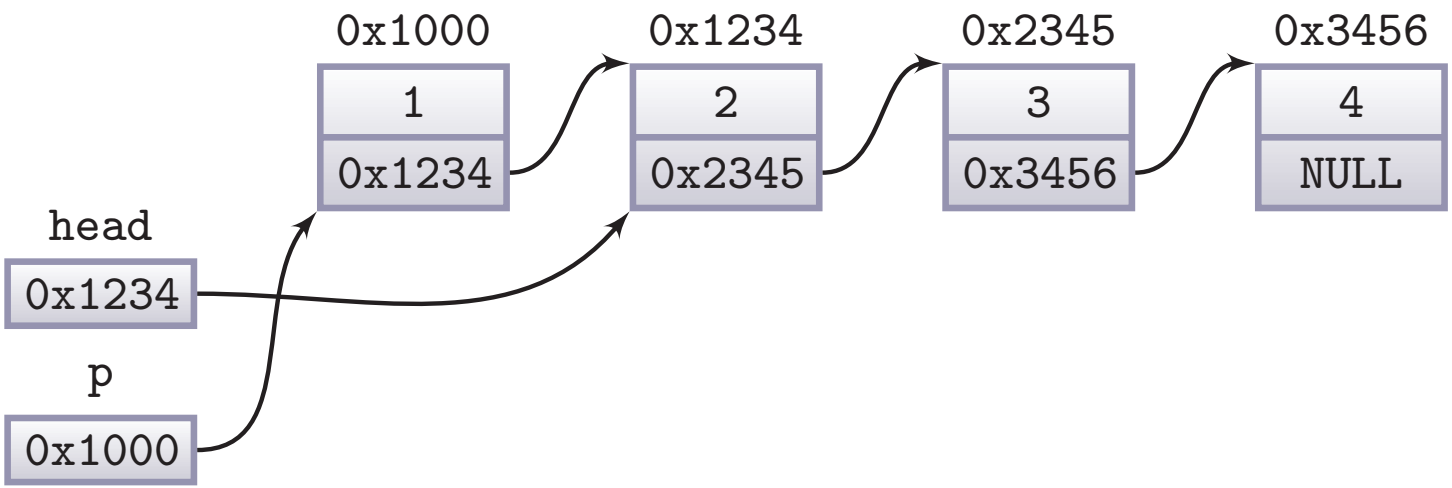
```
1  p = head;
2  head = head->next;
3  free(p);
```



Deleting element from the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

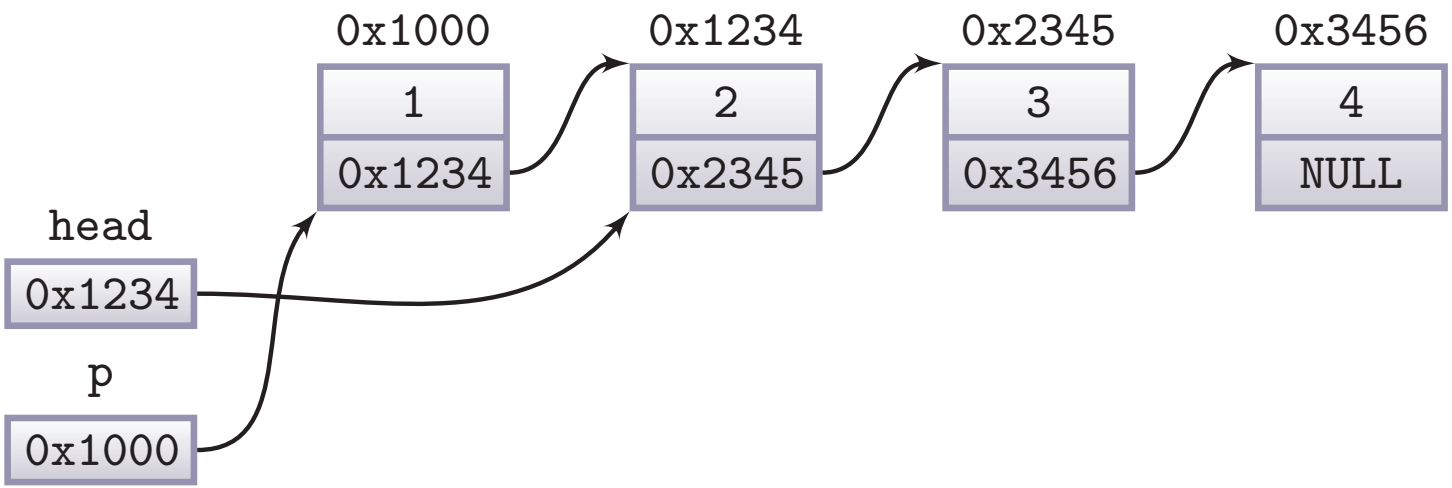
```
1  p = head;
2  head = head->next;
3  free(p);
```



Deleting element from the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

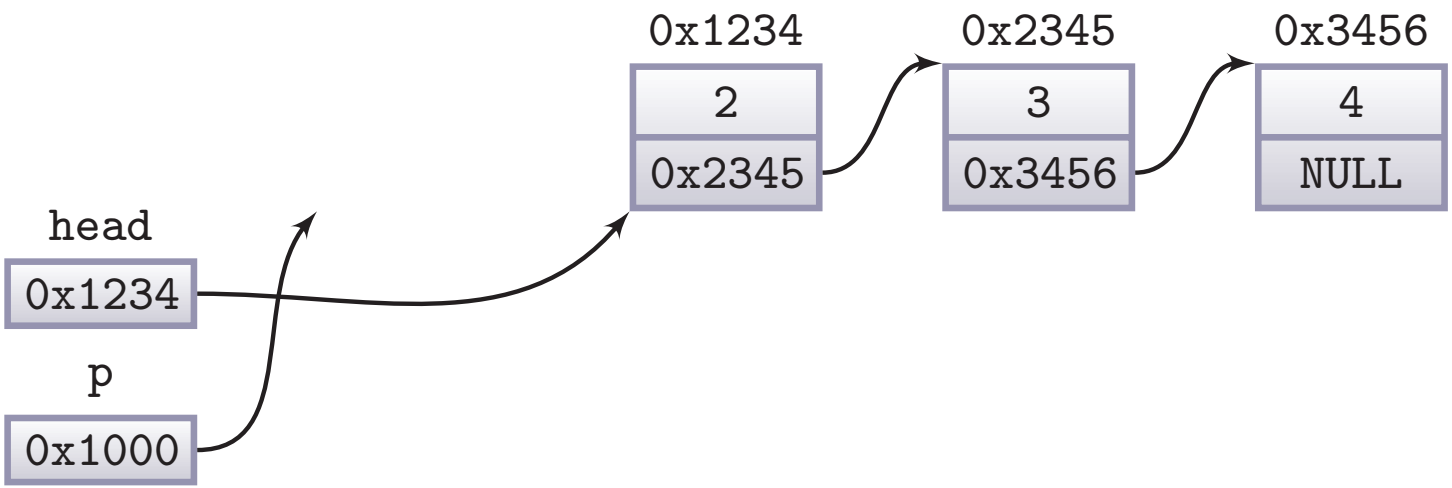
```
1  p = head;
2  head = head->next;
3  free(p);
```



Deleting element from the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

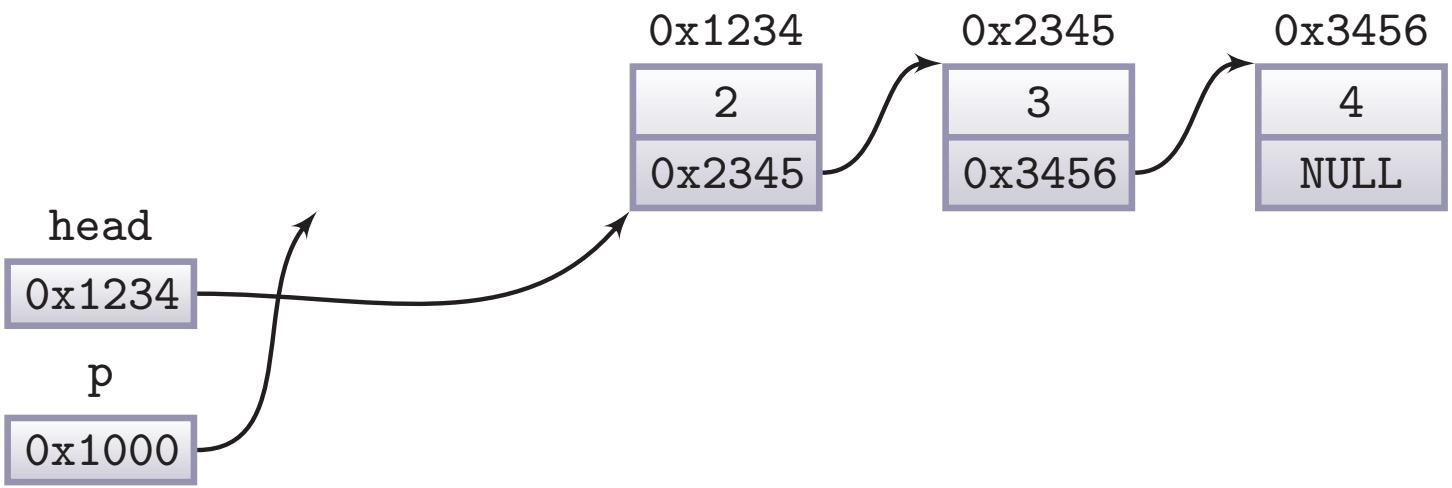
```
1  p = head;
2  head = head->next;
3  free(p);
```



Deleting element from the front of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  p = head;
2  head = head->next;
3  free(p);
```



Deleting element from front of the list with a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  listelem *pop_front(listelem *head)
2  {
3    if (head != NULL) /* not empty */
4    {
5      listelem *p = head;
6      head = head->next;
7      free(p);
8    }
9    return head;
10 }
```

link

- An empty list must be handled separately

Deleting element from front of the list with a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  listelem *pop_front(listelem *head)
2  {
3    if (head != NULL) /* not empty */
4    {
5      listelem *p = head;
6      head = head->next;
7      free(p);
8    }
9    return head;
10 }
```

link

- An empty list must be handled separately
- Of course we could use the solution when calling the functionwith the address of `head`

Stack

- What we have so far is already enough for storing the undo-list

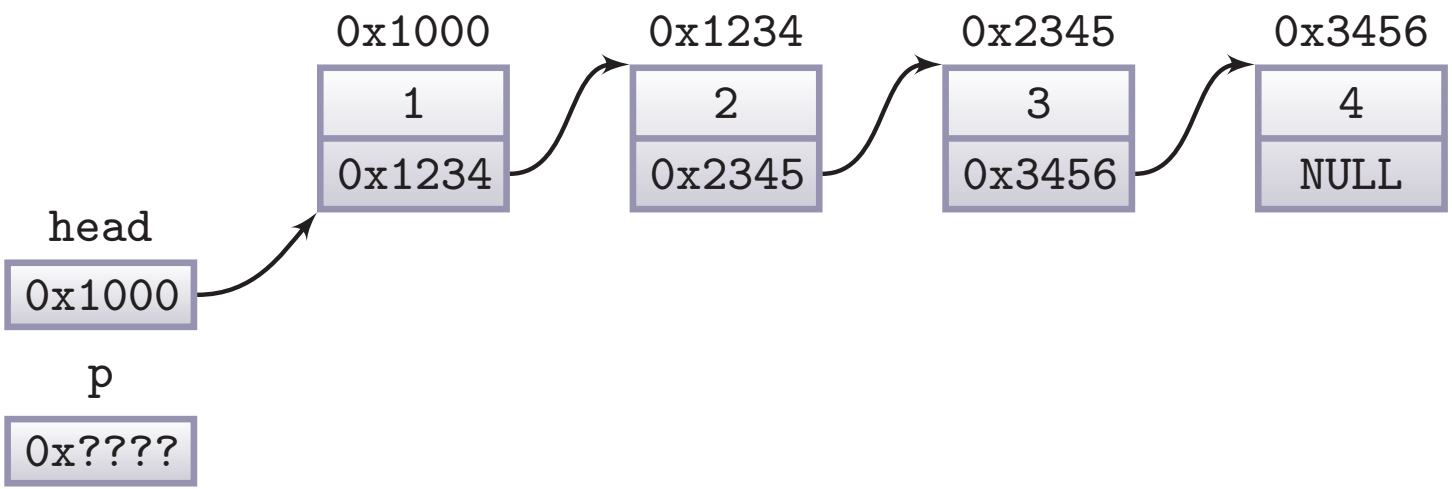
```
1  listelem *head = NULL;                  /* empty list */
2  head = push_front(head, 2);             /* step */
3  head = push_front(head, 4);             /* step */
4  printf("The last inserted element: %d\n", head->data);
5  head = pop_front(head);                 /* undo */
6  head = push_front(head, 5);             /* step */
7  head = pop_front(head);                 /* step */
8  head = pop_front(head);                 /* strep */
```

- The stack is a LIFO: Last In, First Out
- We can access the last inserted element first

Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

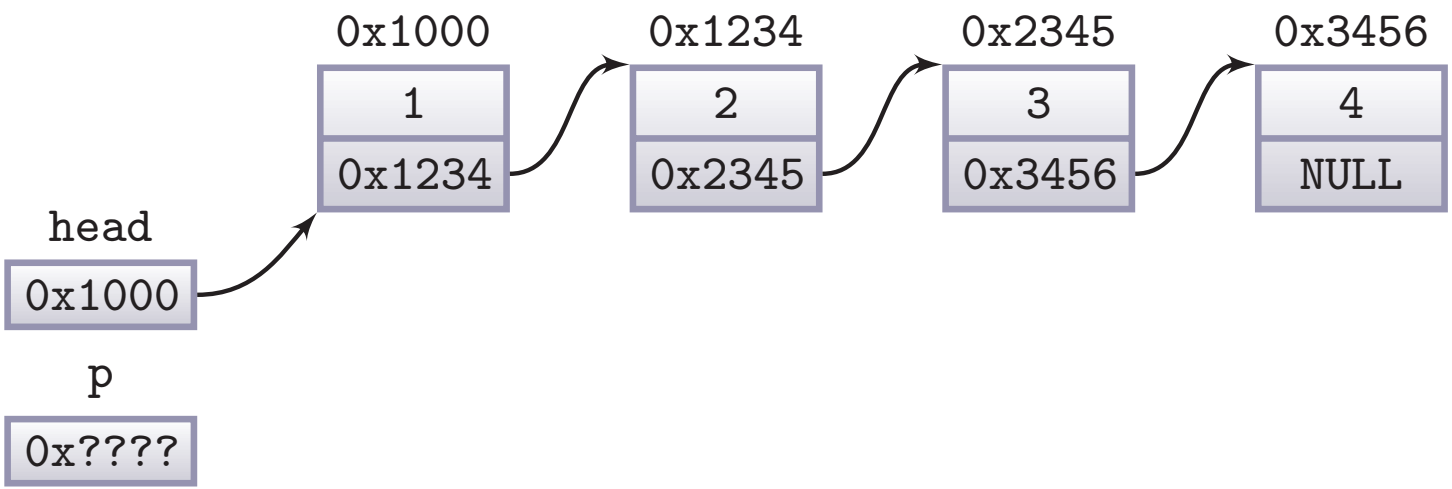
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

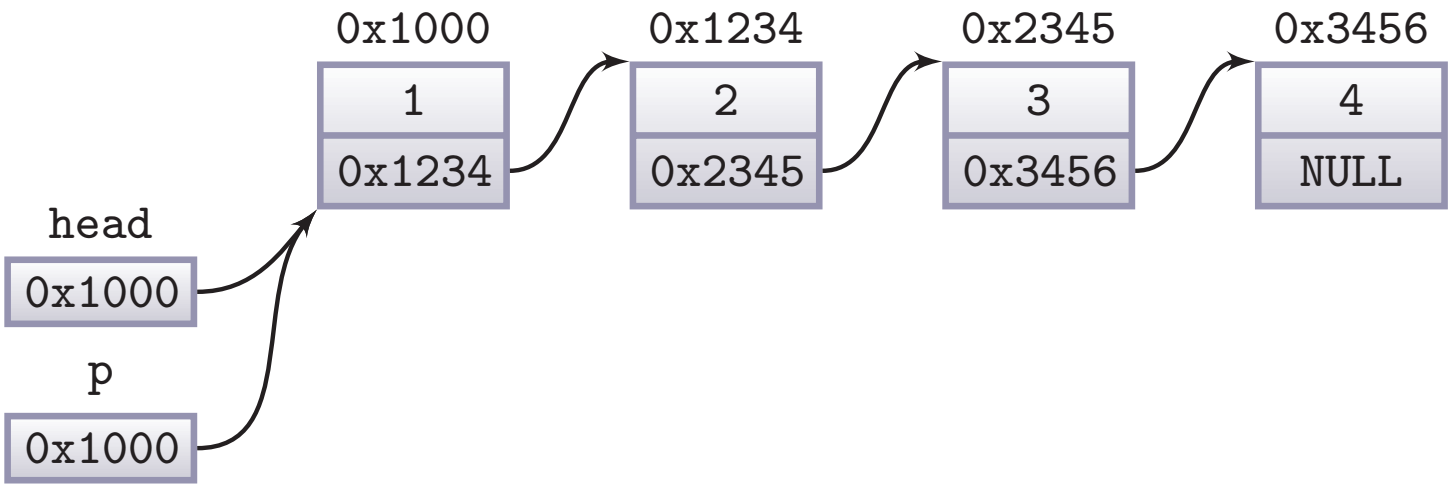
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

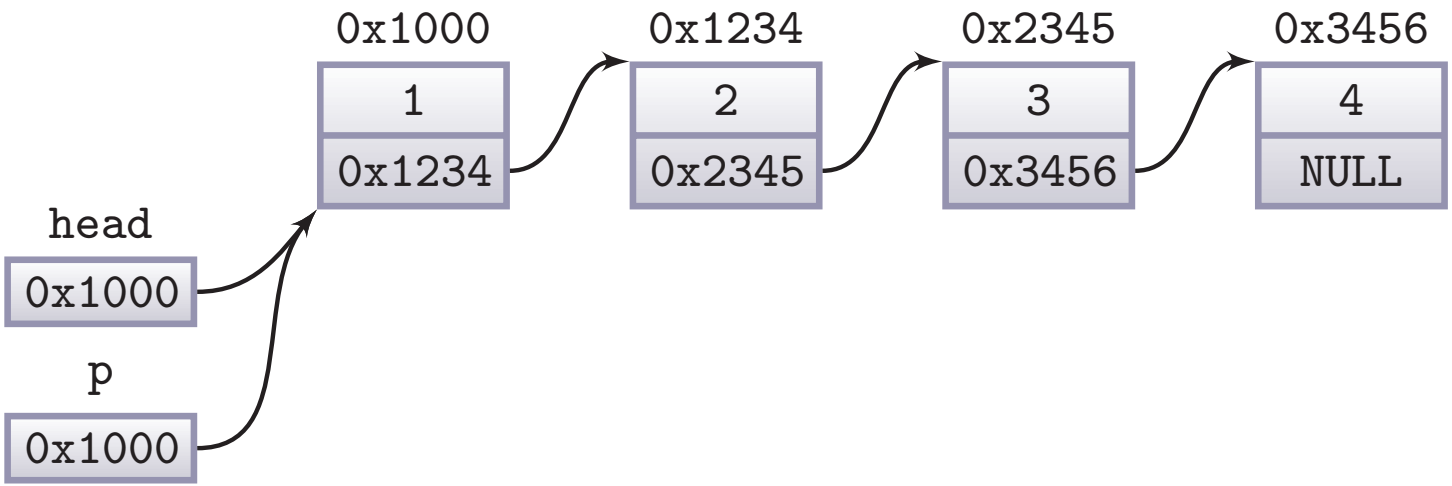
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

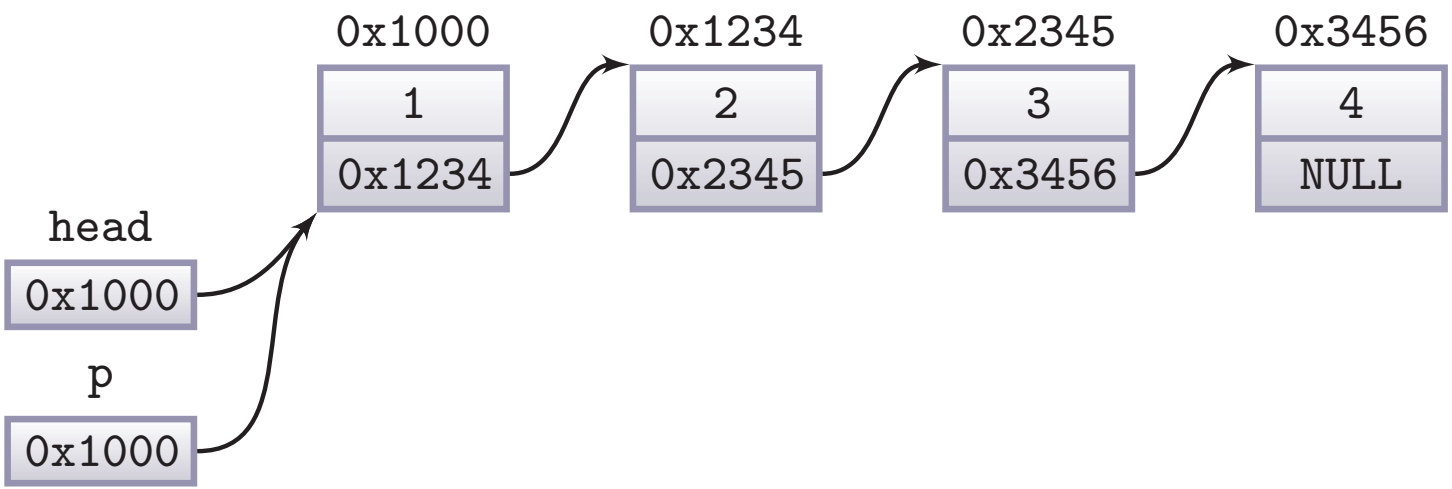
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

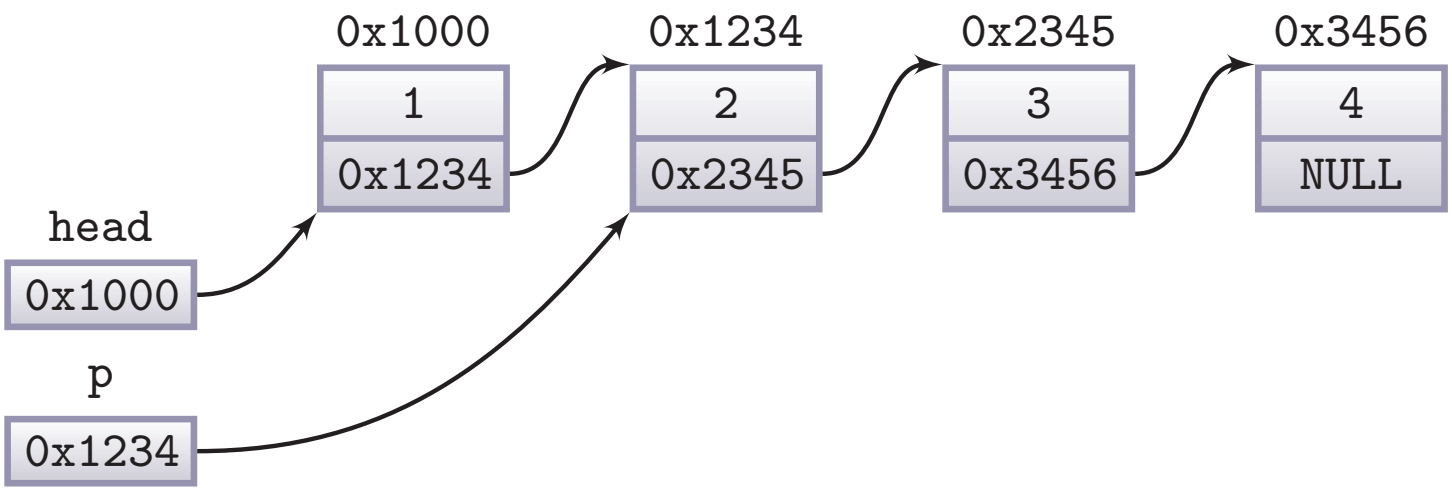
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

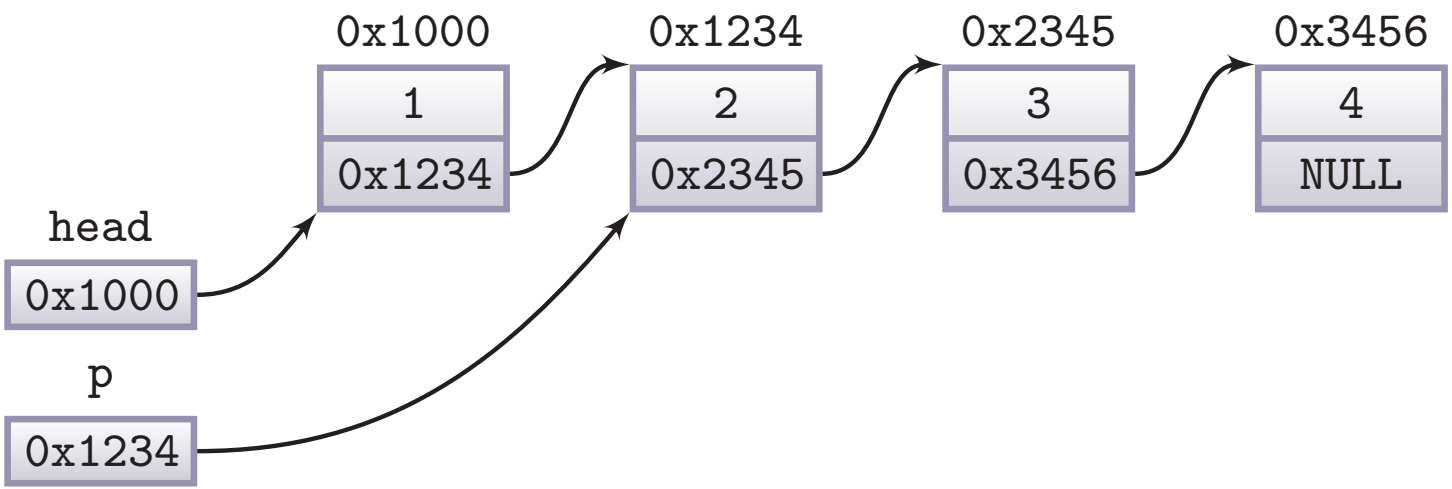
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

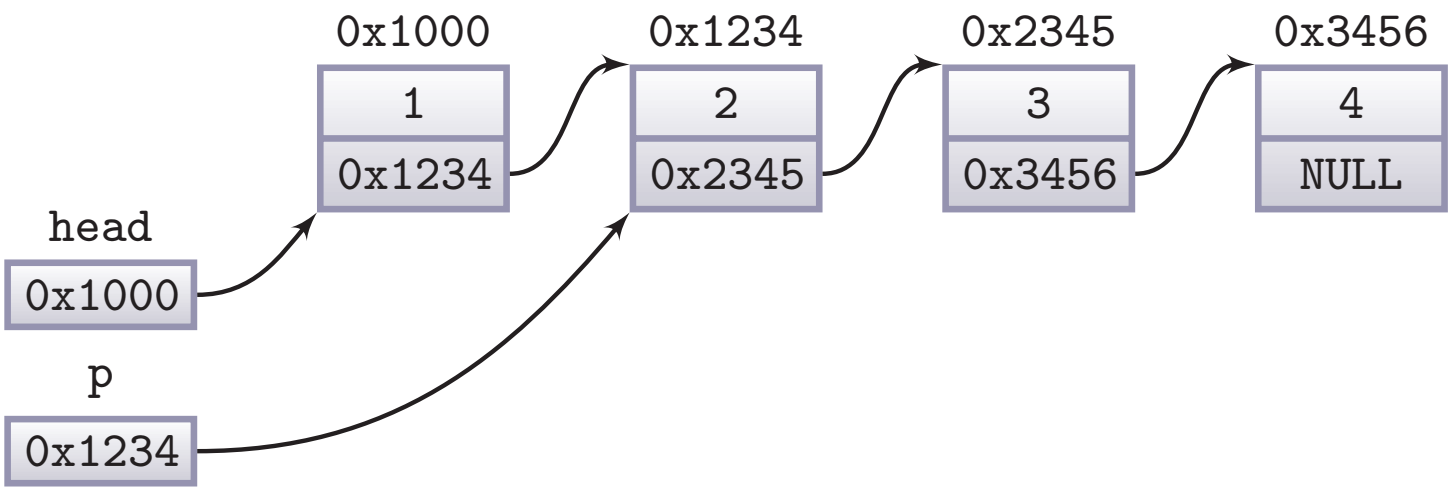
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

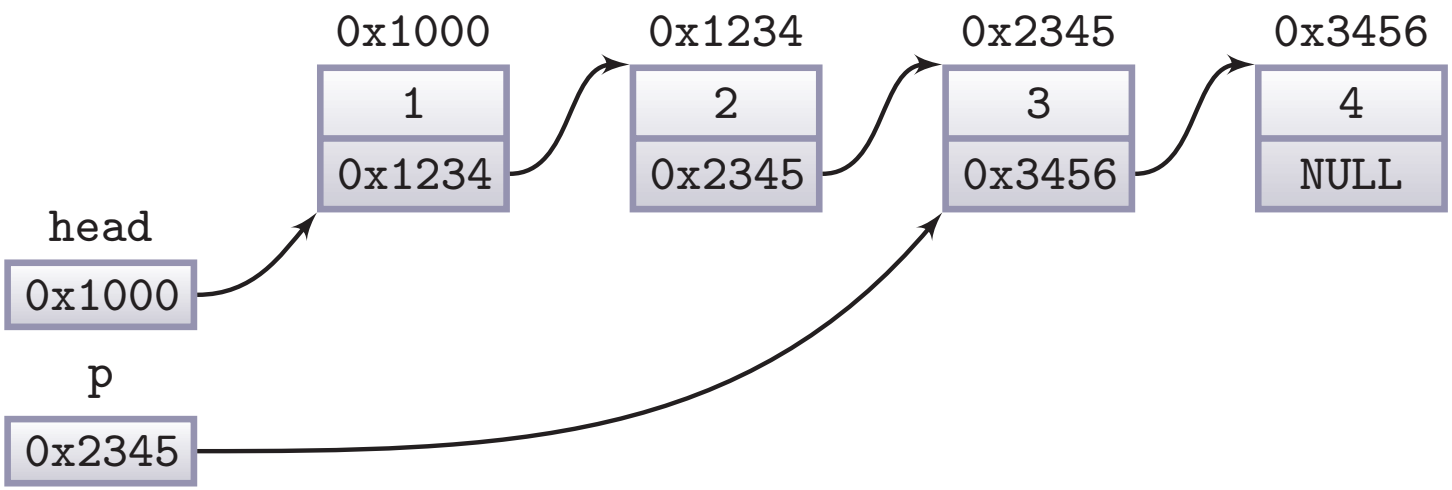
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

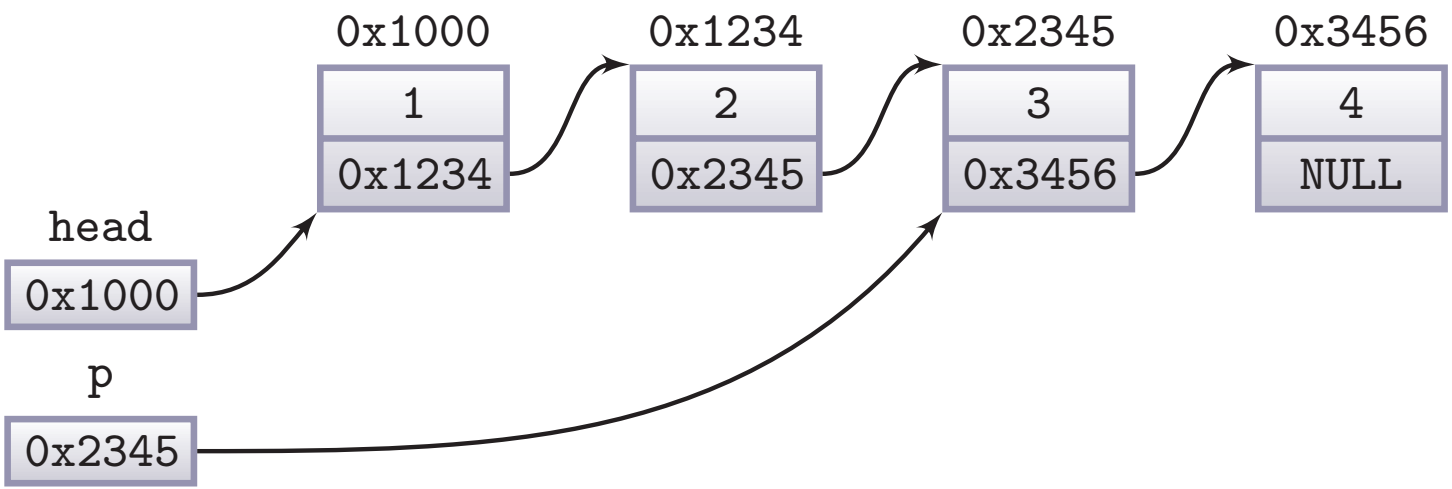
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

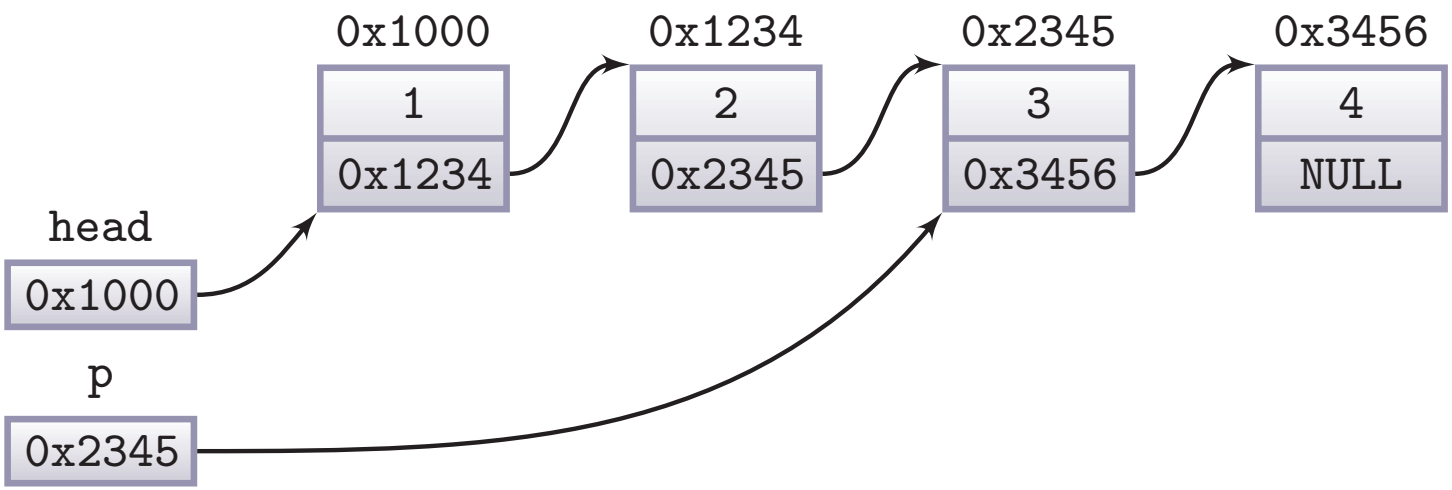
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

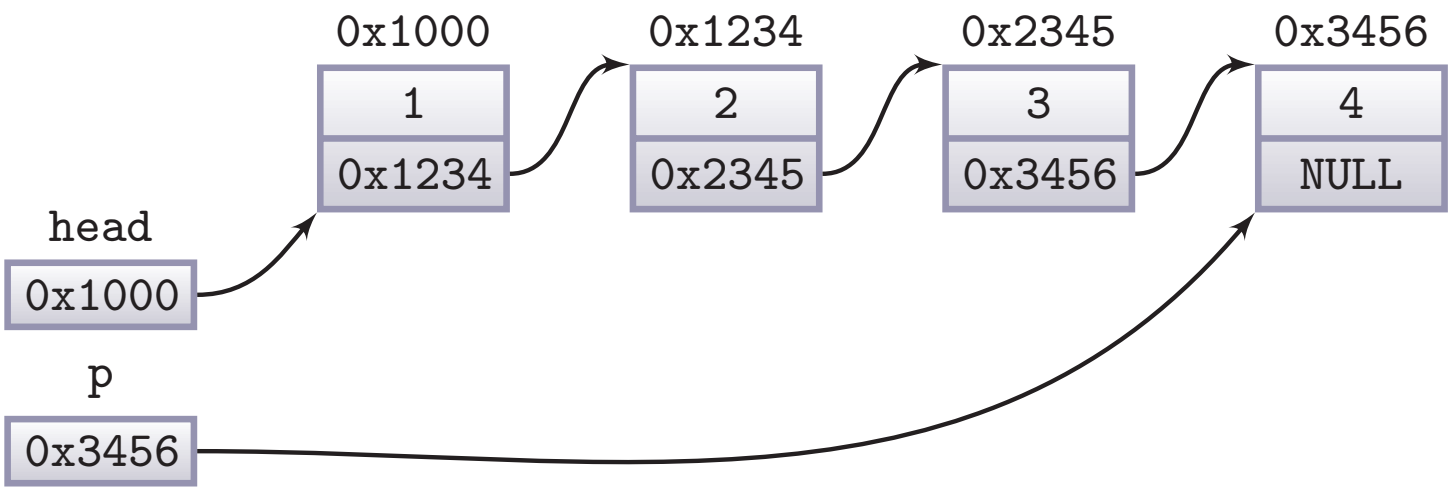
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

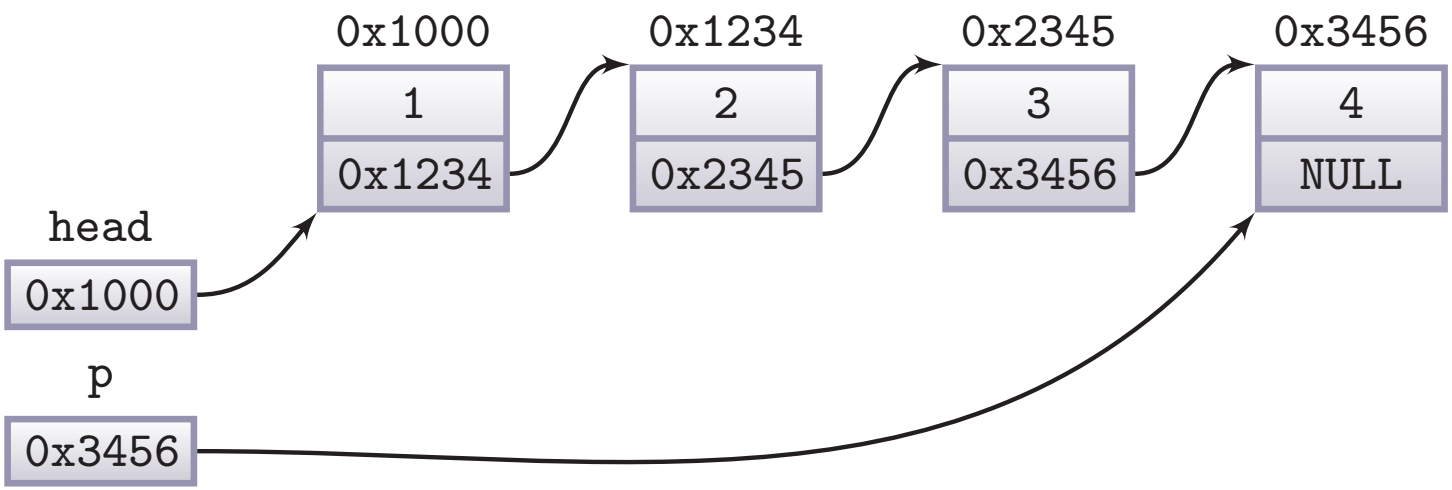
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

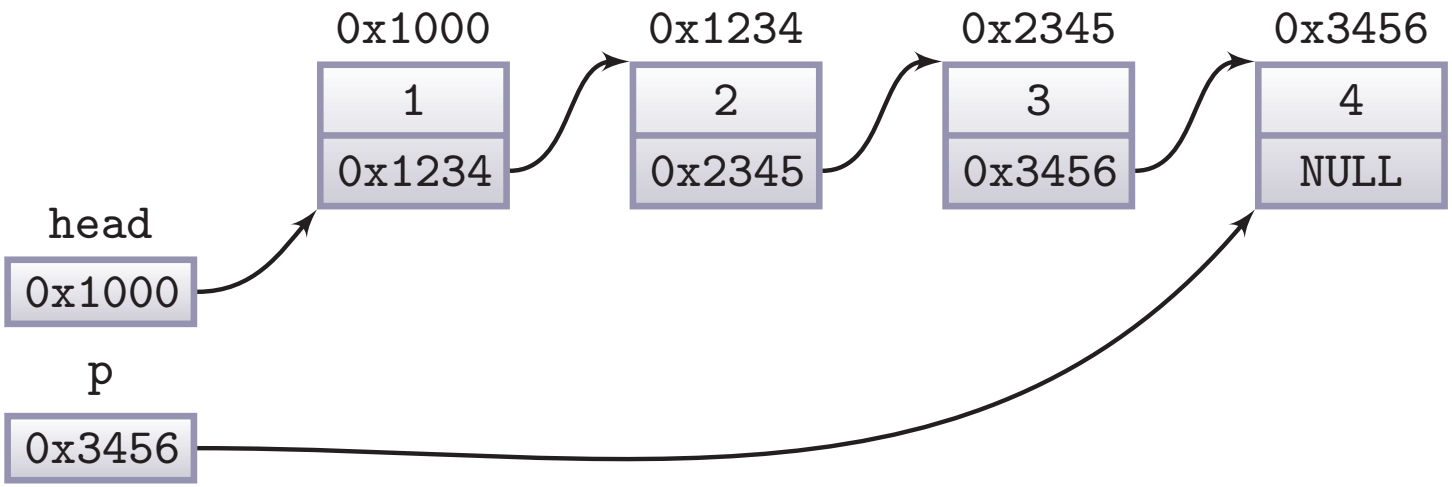
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

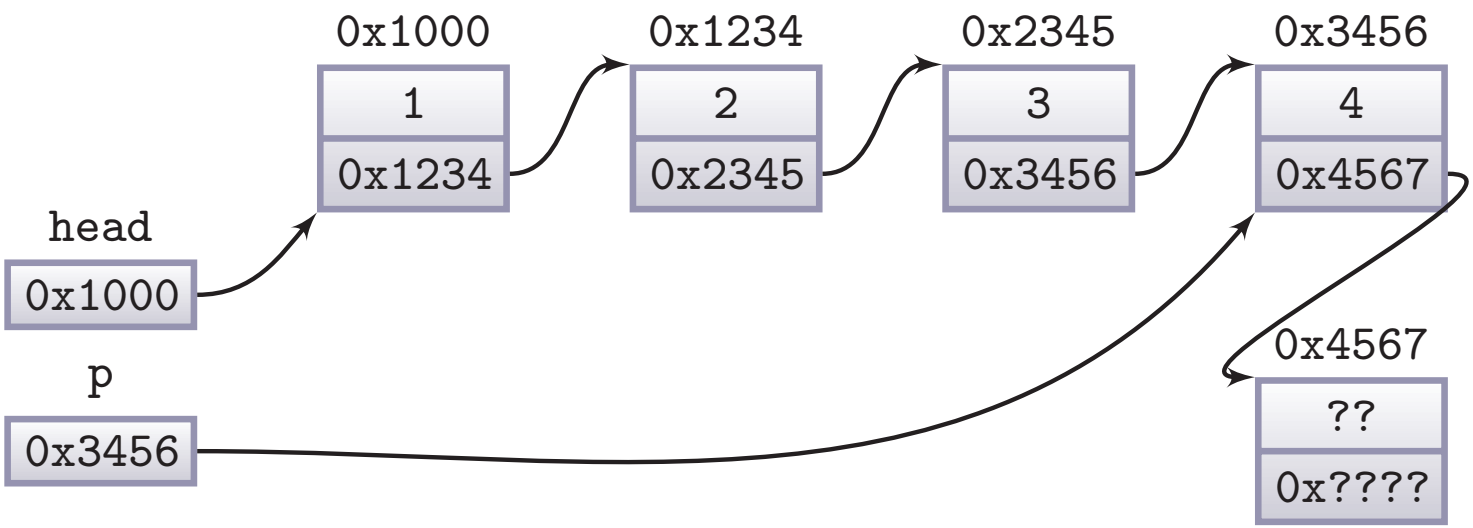
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

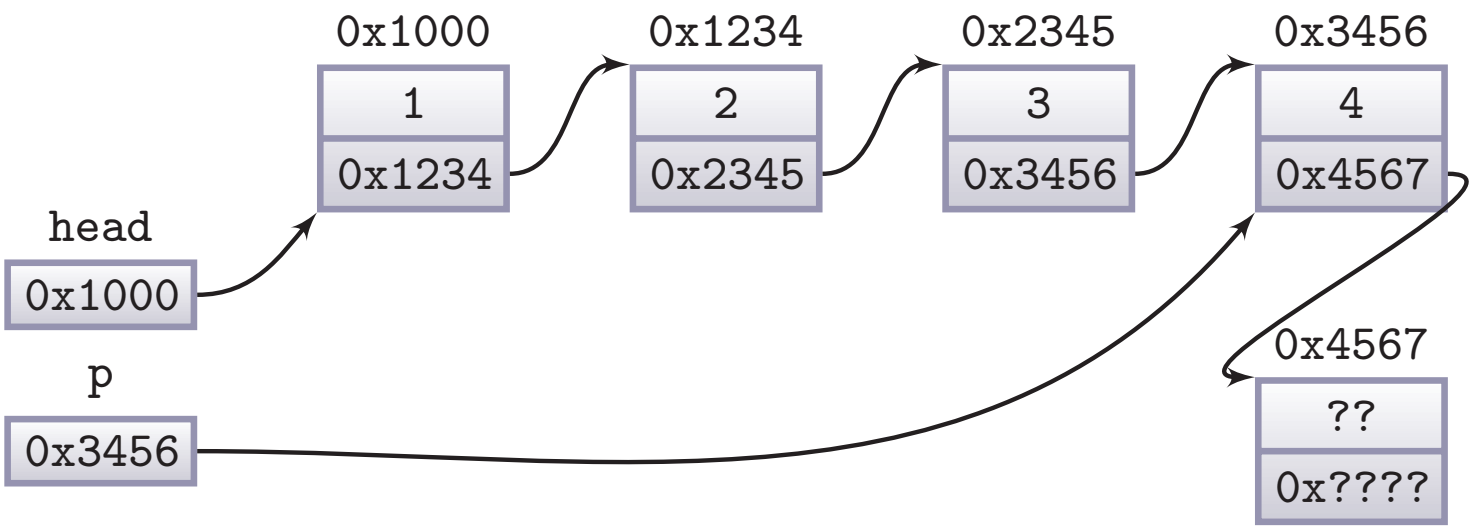
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

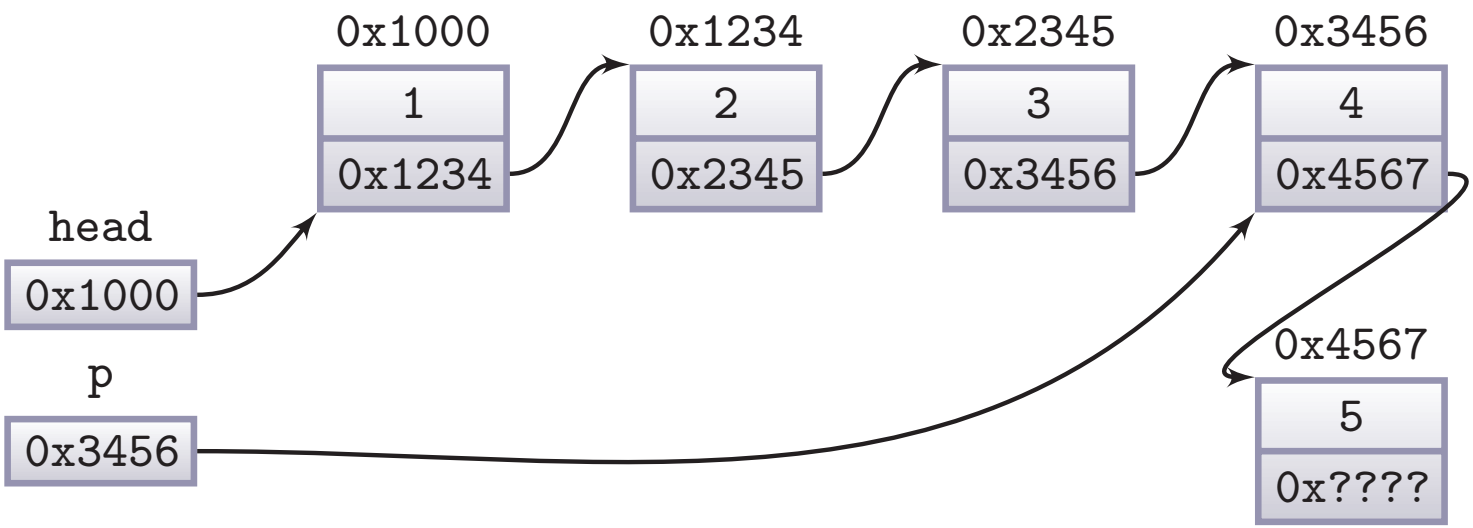
```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

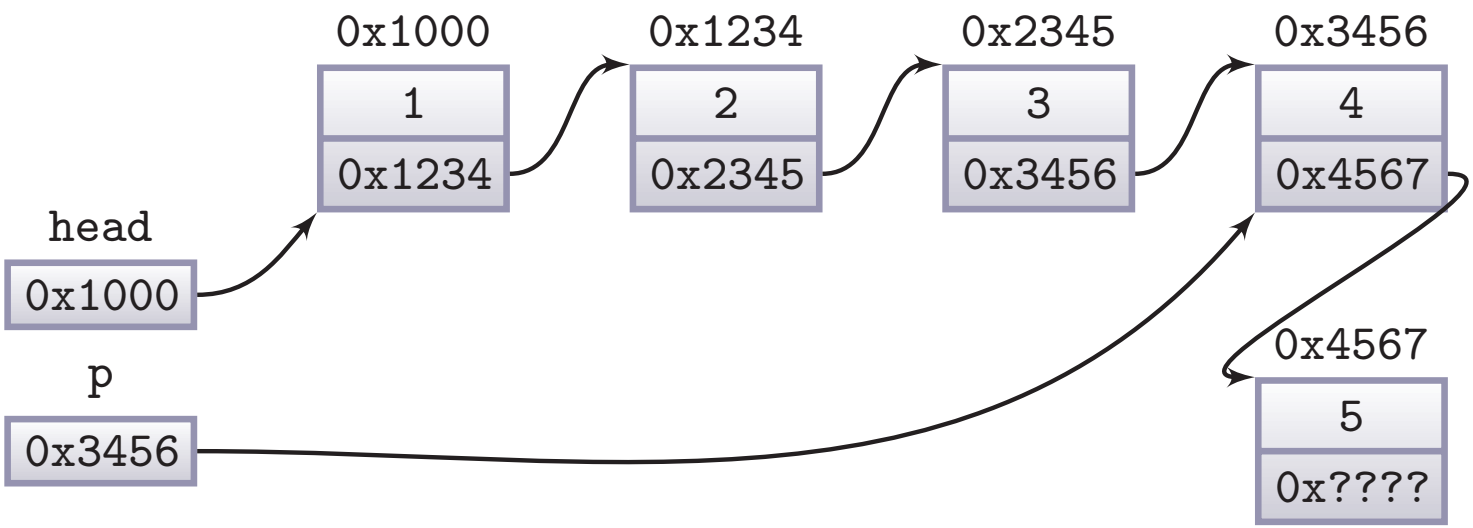
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



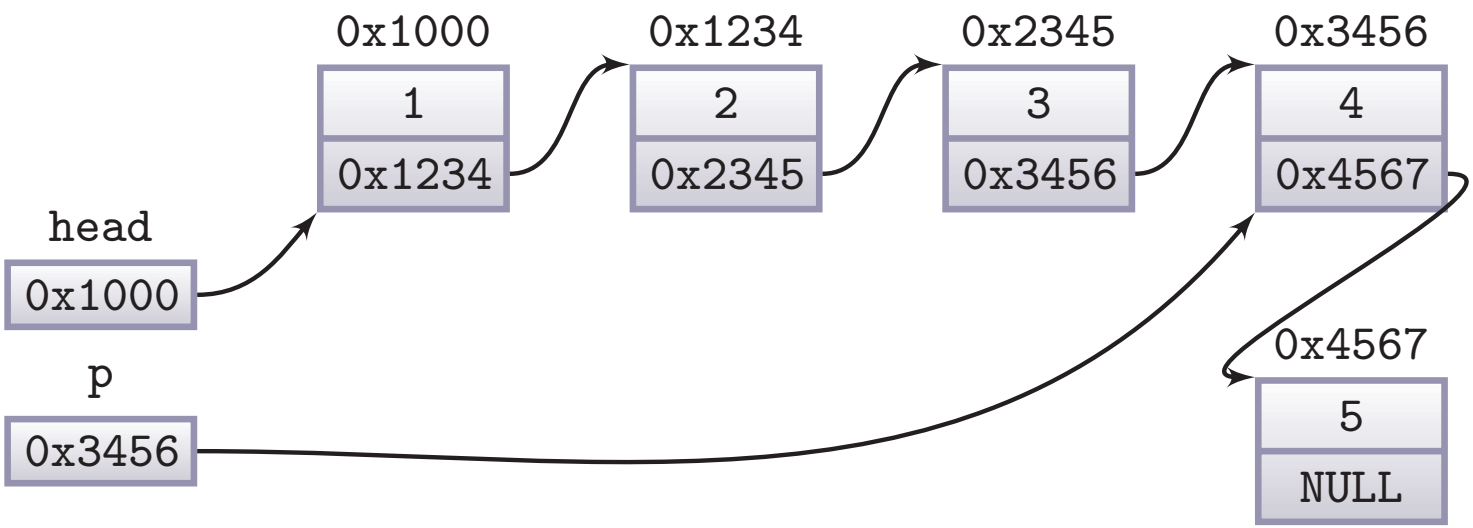
Inserting element to the end of the list


```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



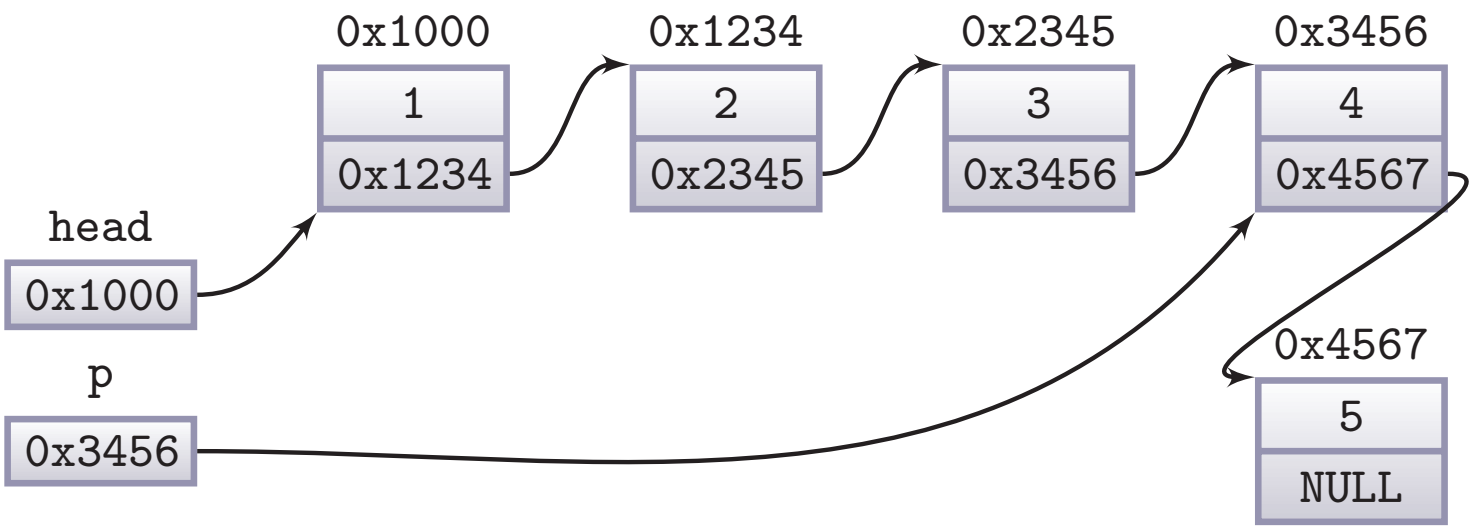
Inserting element to the end of the list


```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

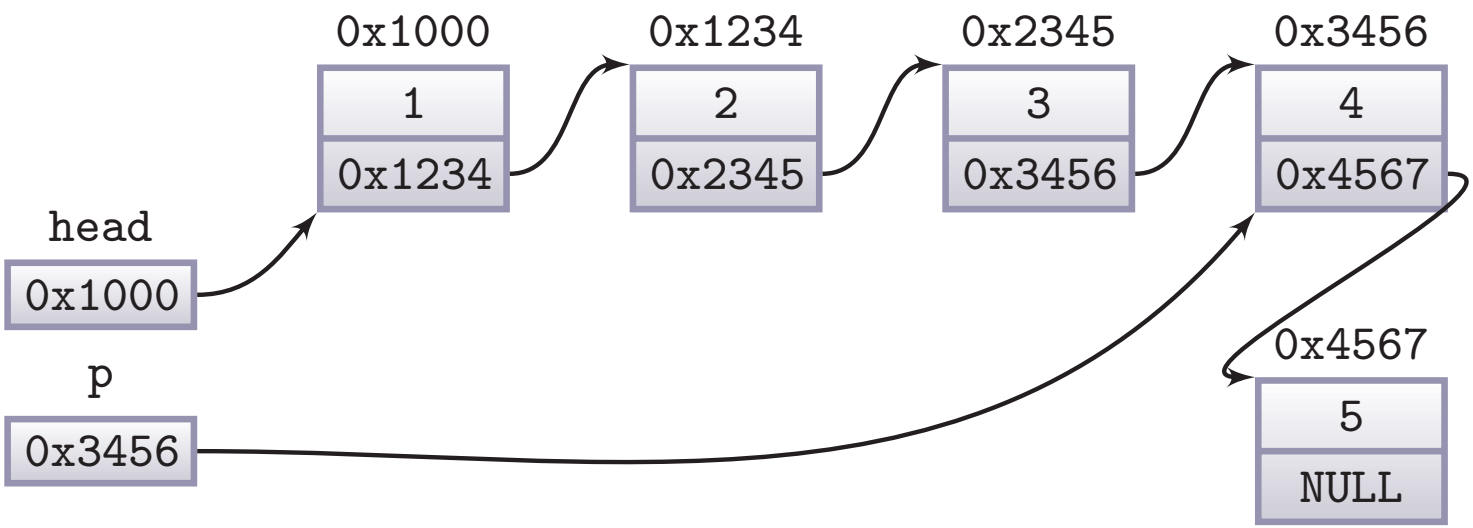

```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



Inserting element to the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  for (p = head; p->next != NULL; p = p->next);
2  p->next = (listelem*)malloc(sizeof(listelem));
3  p->next->data = 5;
4  p->next->next = NULL;
```



- If the list is empty, checking `p->next != NULL` is not possible, this case must be managed separately!

Inserting element to the end of the list with a function

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  listelem *push_back(listelem *head, int d)
2  {
3    listelem *p;
4
5    if (head == NULL)   /* empty list should be
6                managed separately */
7      return push_front(head, d);
8
9    for (p = head; p->next != NULL; p = p->next);
10   p->next = (listelem*)malloc(sizeof(listelem));
11   p->next->data = d;
12   p->next->next = NULL;
13   return head;
14 }
```

link

```
1  listelem *head = NULL;
2  head = push_back(head, 2);
```

Inserting element into a sorted list

- If we have to traverse and process our data several times, it is worth sorting it
- Arrays:
  - re-locating a single element involves a lot of data movements
  - we fill up the array and order it afterwards

Inserting element into a sorted list

- If we have to traverse and process our data several times, it is worth sorting it
- Arrays:
  - re-locating a single element involves a lot of data movements
  - we fill up the array and order it afterwards
- Lists:
  - re-locating a single element involves only the modification of pointers, the elements will remain at the same address in the memory
  - it is better to build up our list in a sorted way

Inserting element into a sorted list

- If we have to traverse and process our data several times, it is worth sorting it
- Arrays:
  - re-locating a single element involves a lot of data movements
  - we fill up the array and order it afterwards
- Lists:
  - re-locating a single element involves only the modification of pointers, the elements will remain at the same address in the memory
  - it is better to build up our list in a sorted way
- The new element must be inserted before the first element that is larger then it
- In the present structure each element "can see" only behind itself, so we cannot insert element before another

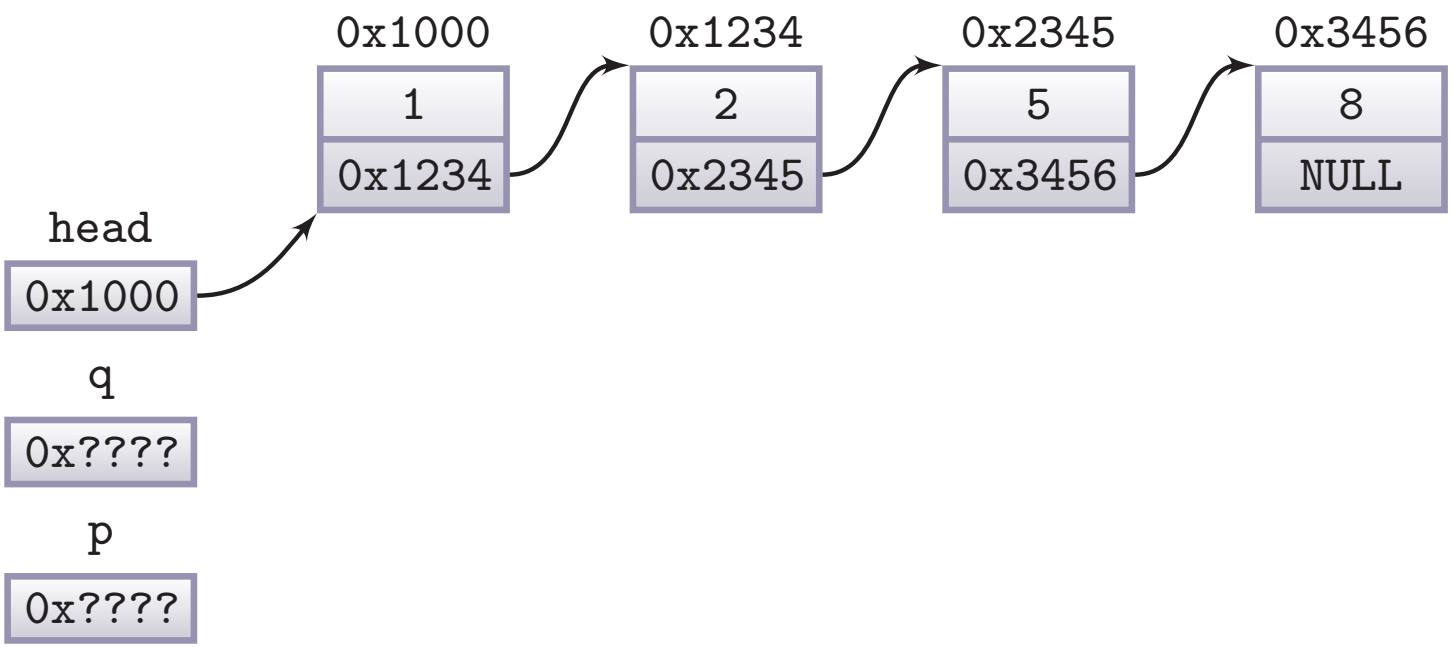
Inserting element into a sorted list

- If we have to traverse and process our data several times, it is worth sorting it
- Arrays:
  - re-locating a single element involves a lot of data movements
  - we fill up the array and order it afterwards
- Lists:
  - re-locating a single element involves only the modification of pointers, the elements will remain at the same address in the memory
  - it is better to build up our list in a sorted way
- The new element must be inserted before the first element that is larger then it
- In the present structure each element "can see" only behind itself, so we cannot insert element before another
- We will use two pointers for traversing the list, one of them will be one step behind (delayed)
- We will insert after the delayed pointer

Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

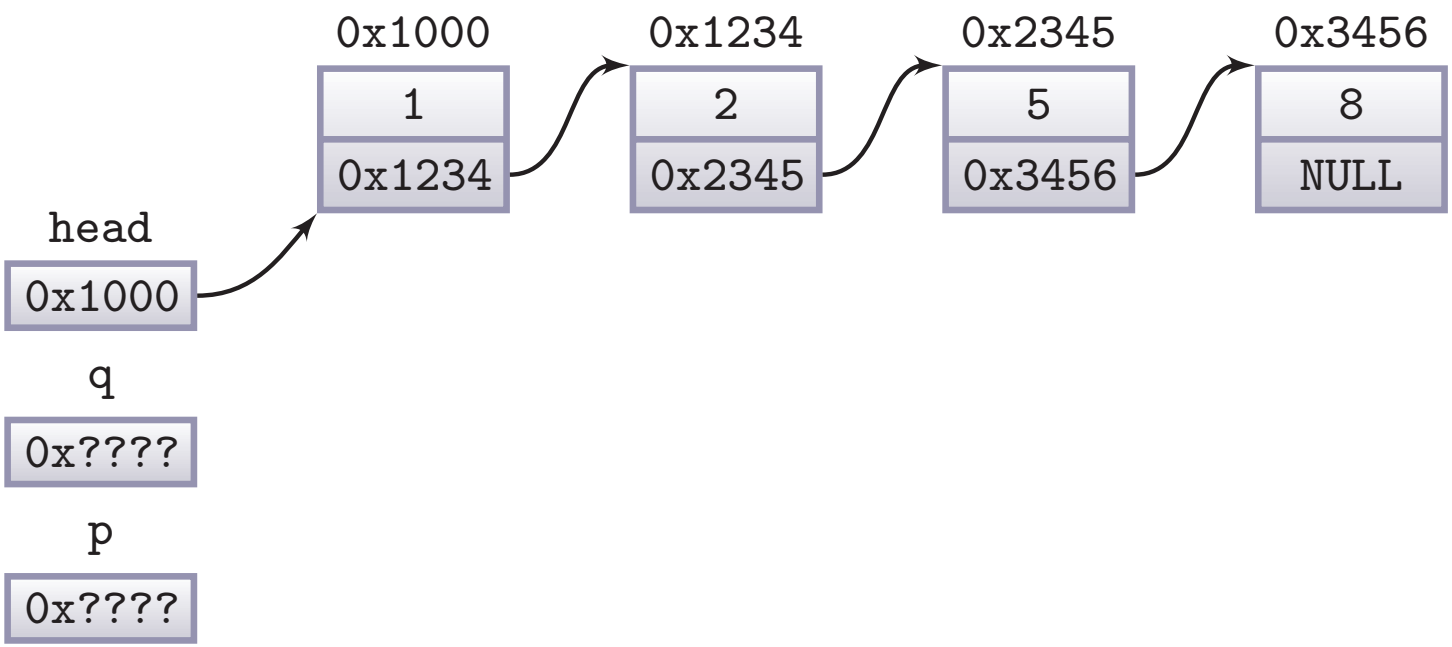
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

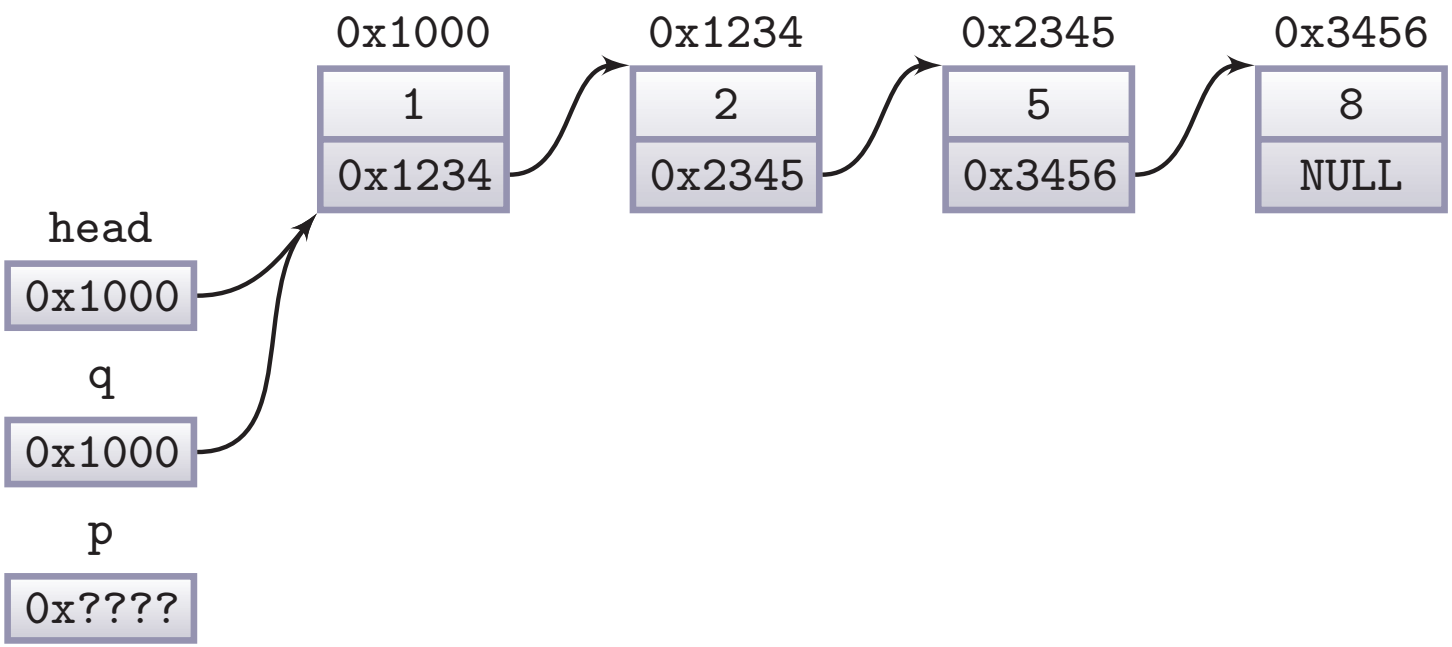
```
1  q = head;   p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;   p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

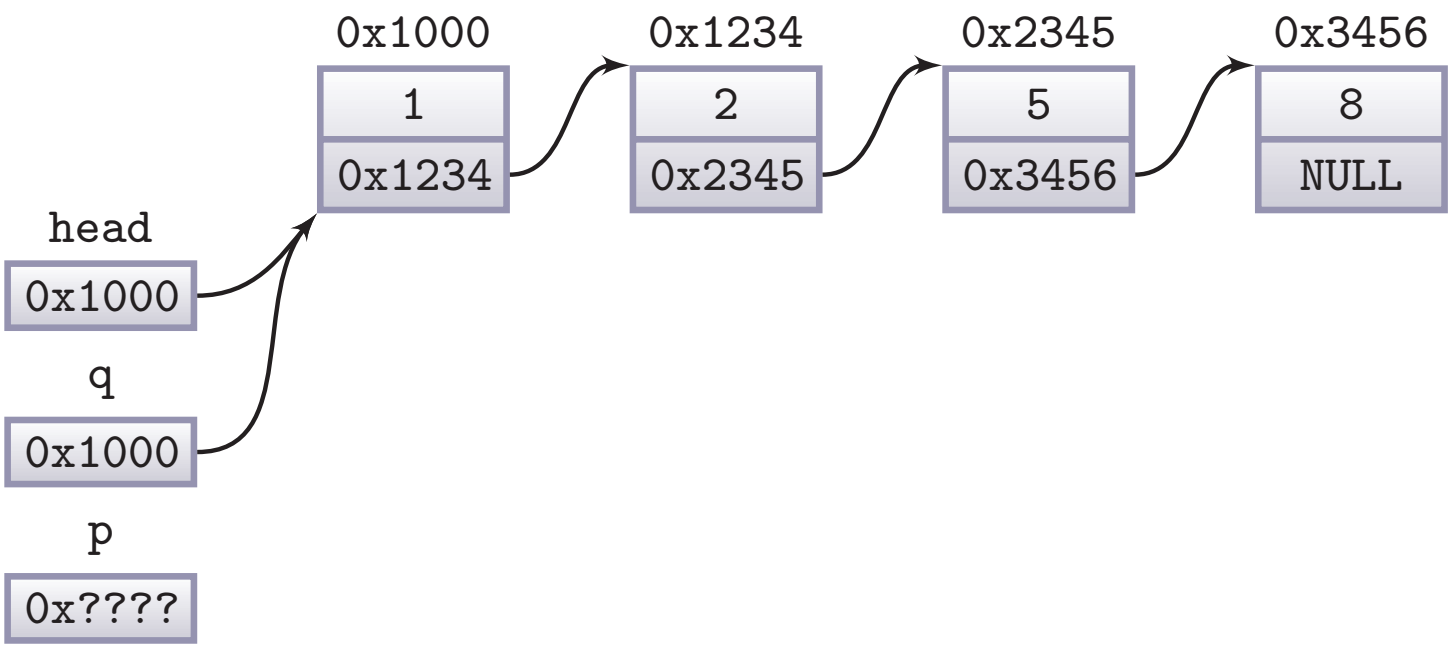
```
1  q = head;   p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;   p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

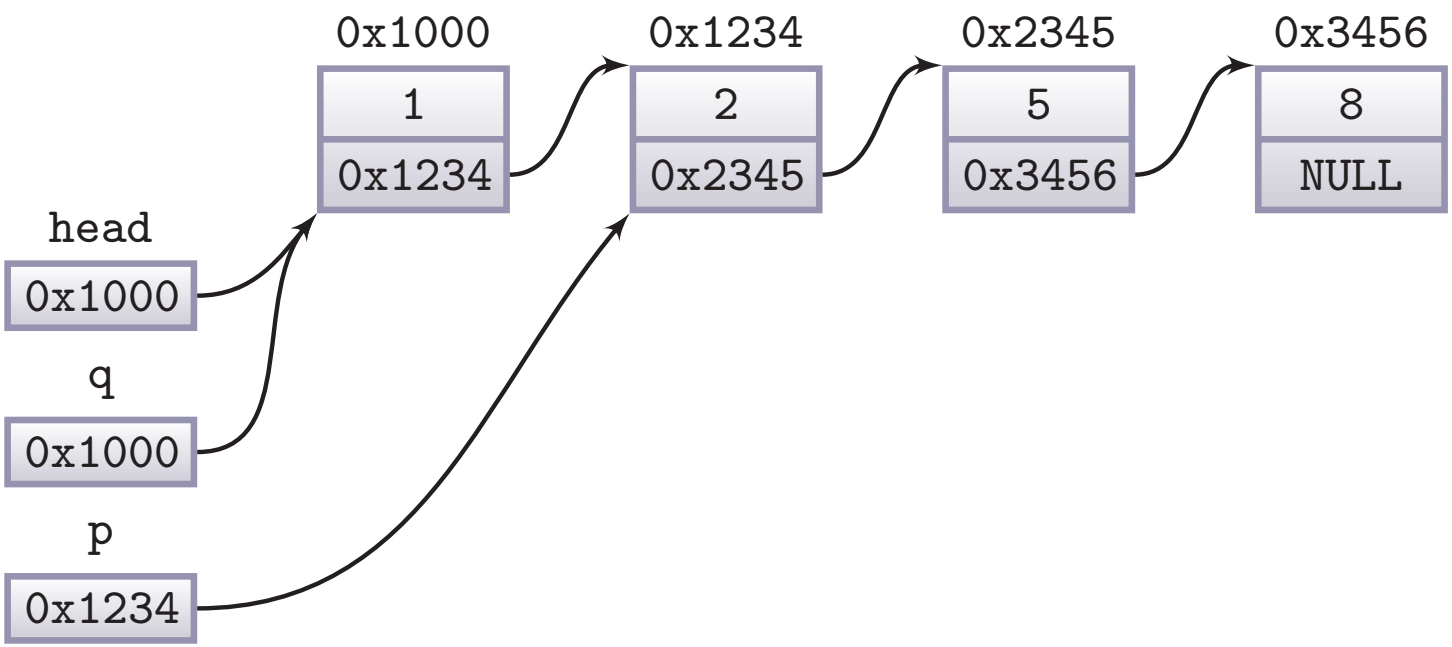
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

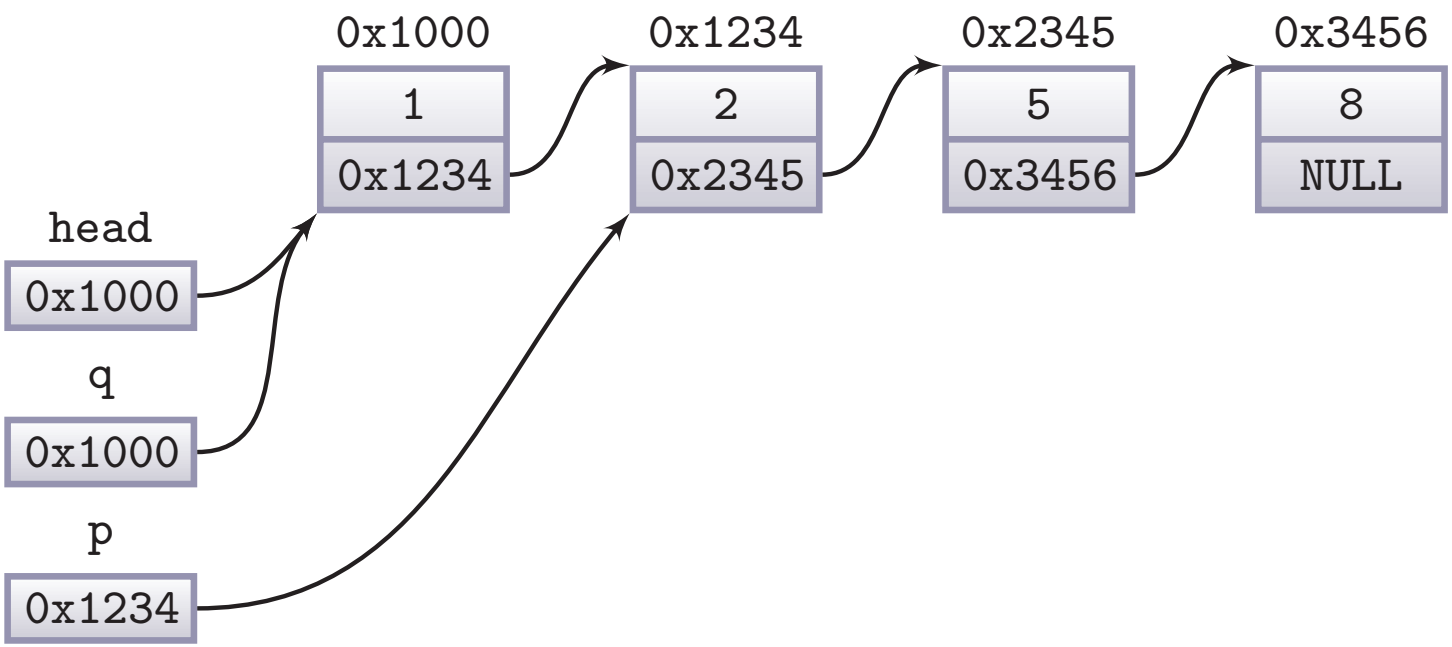
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

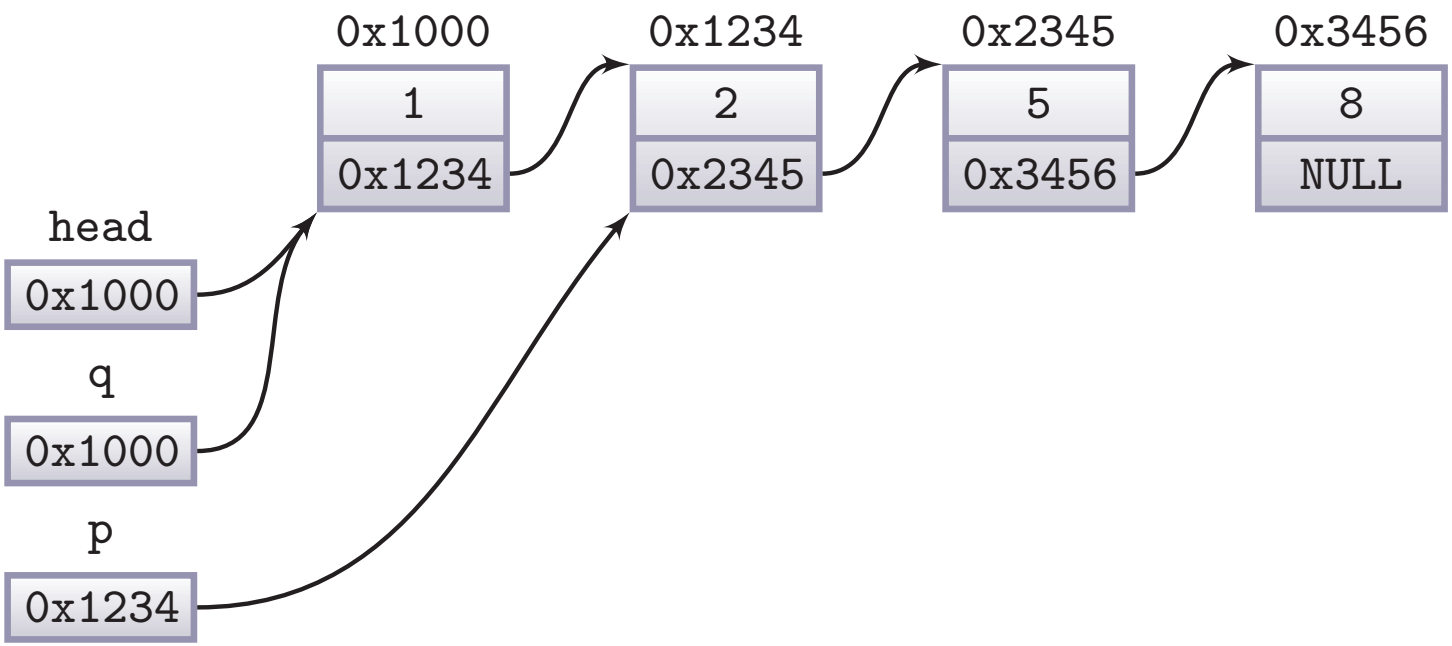
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

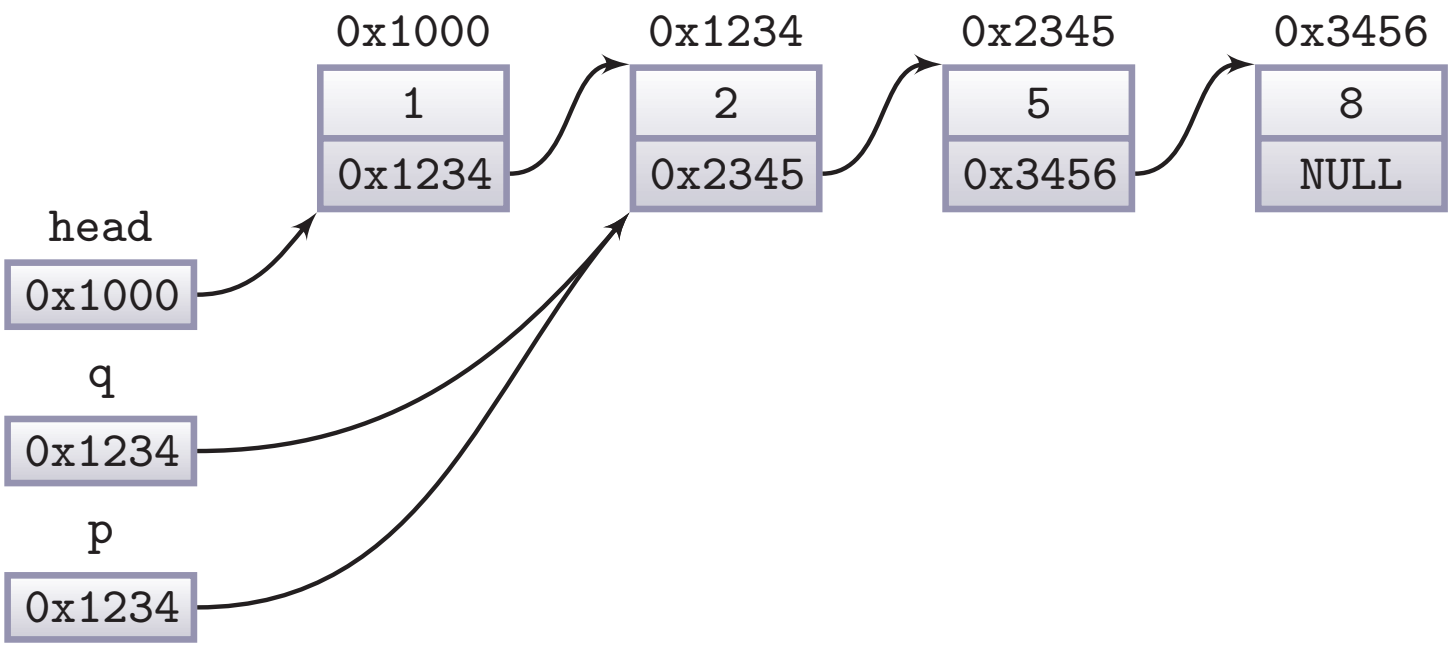
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

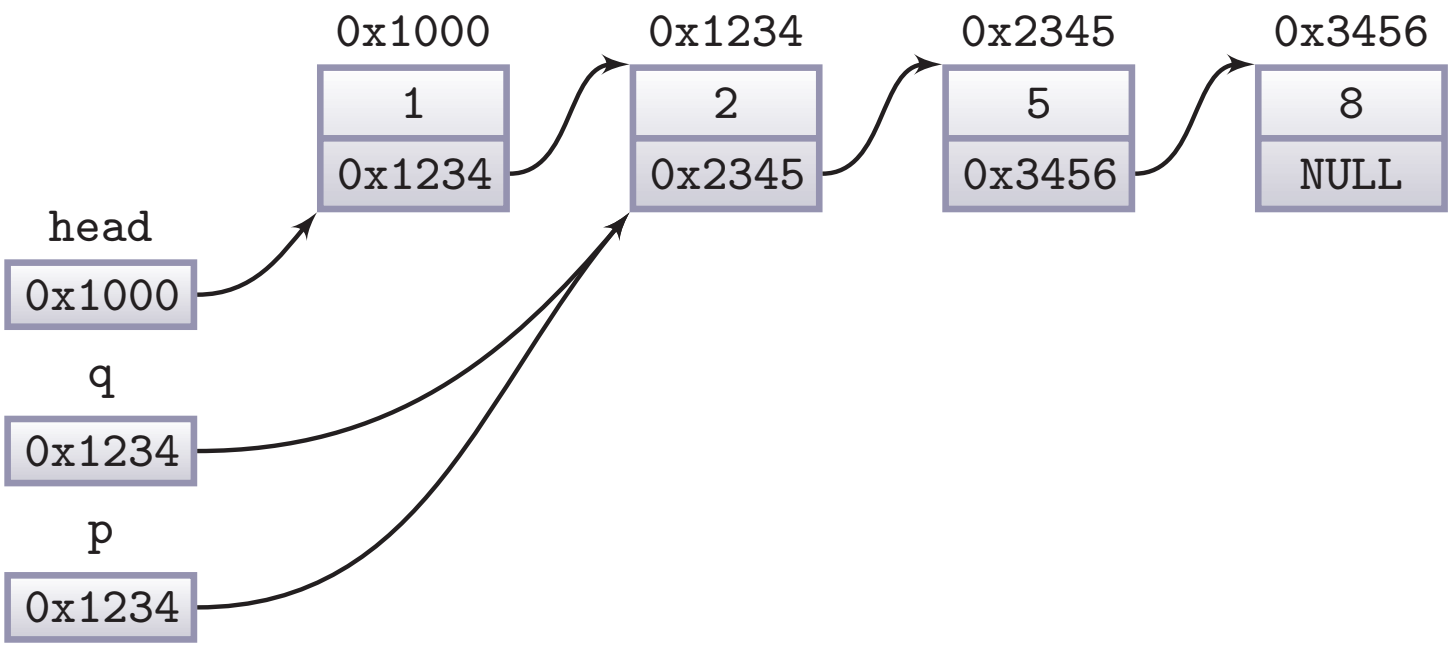
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

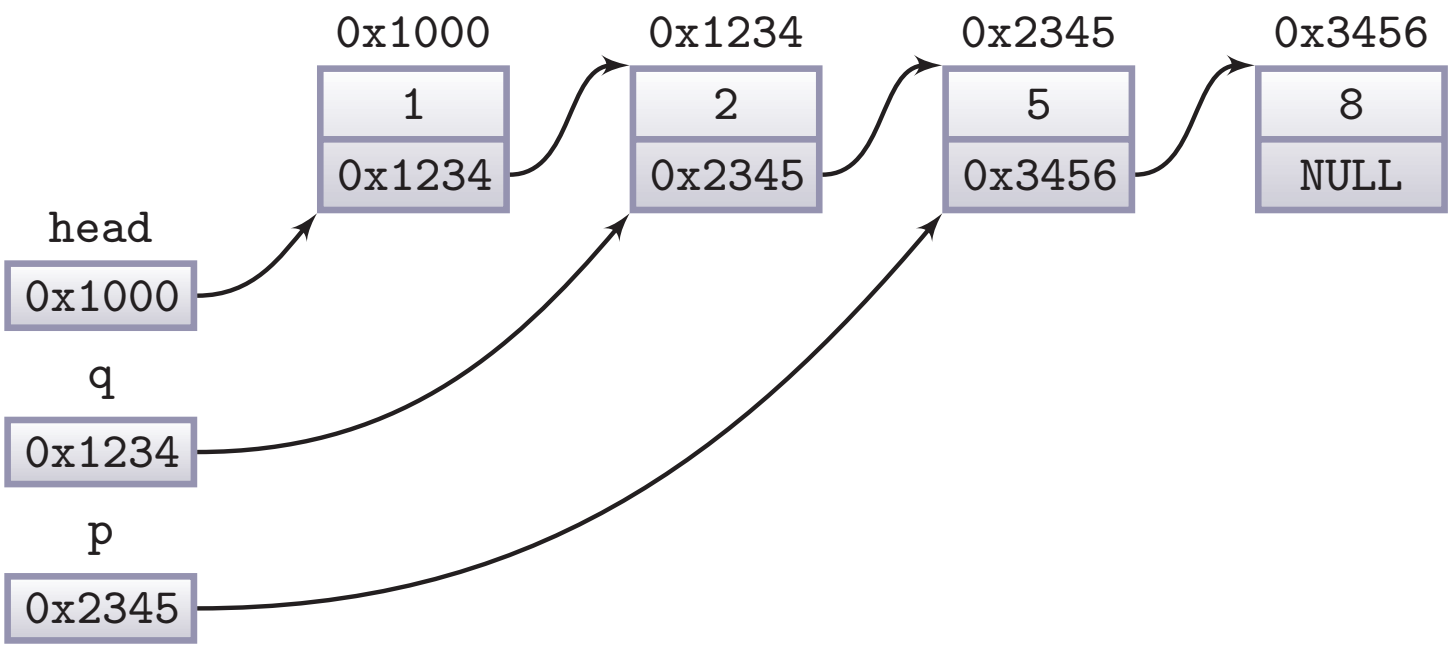
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

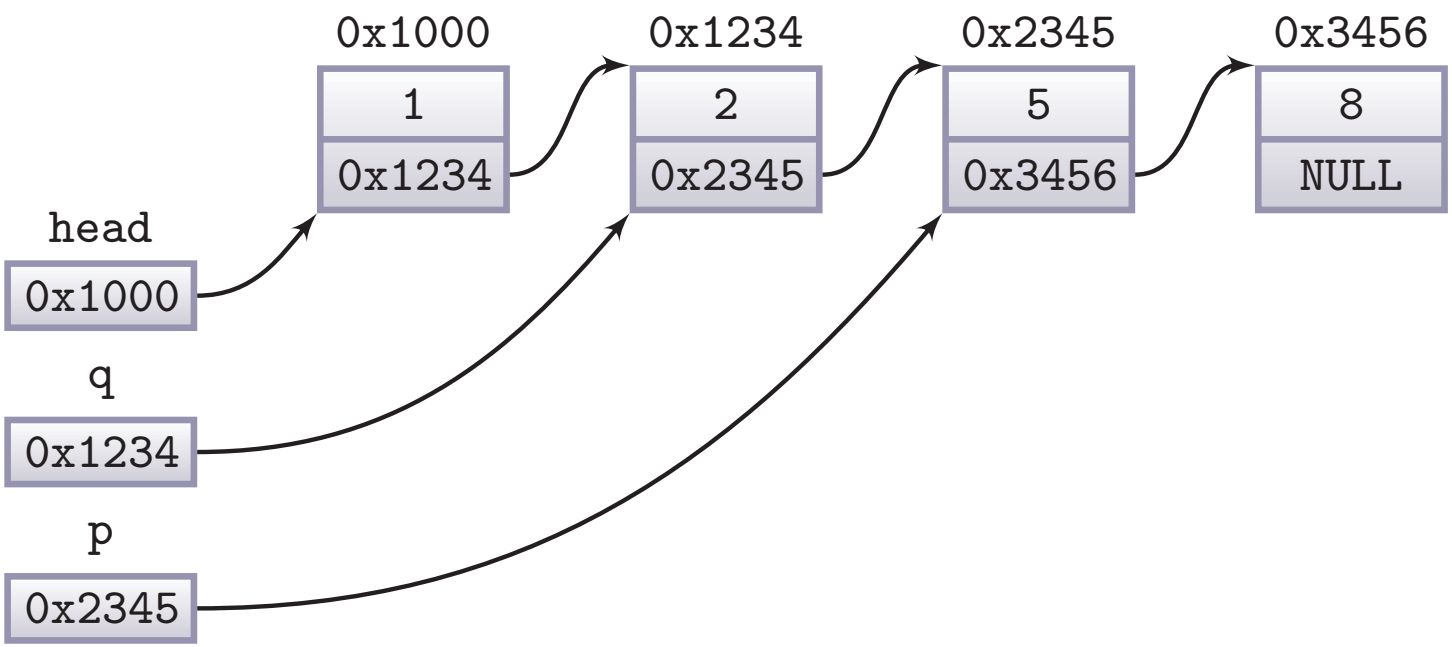
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

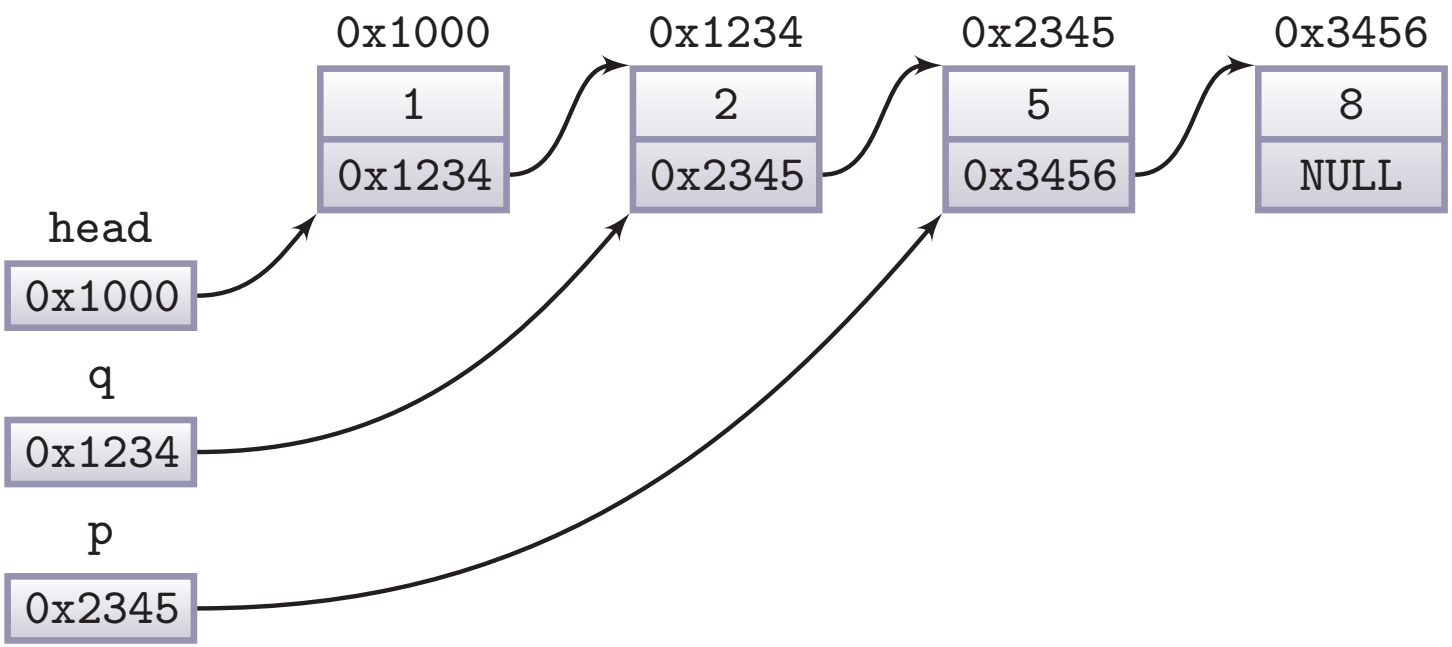
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

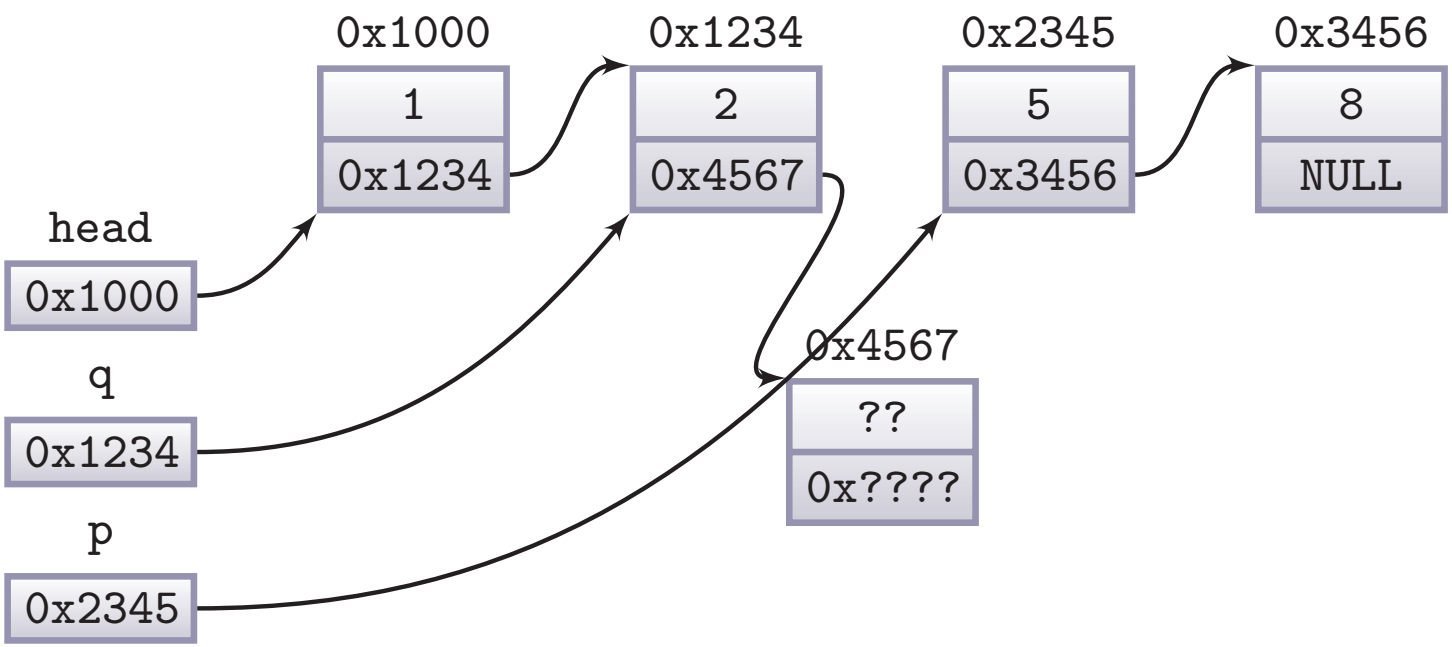
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

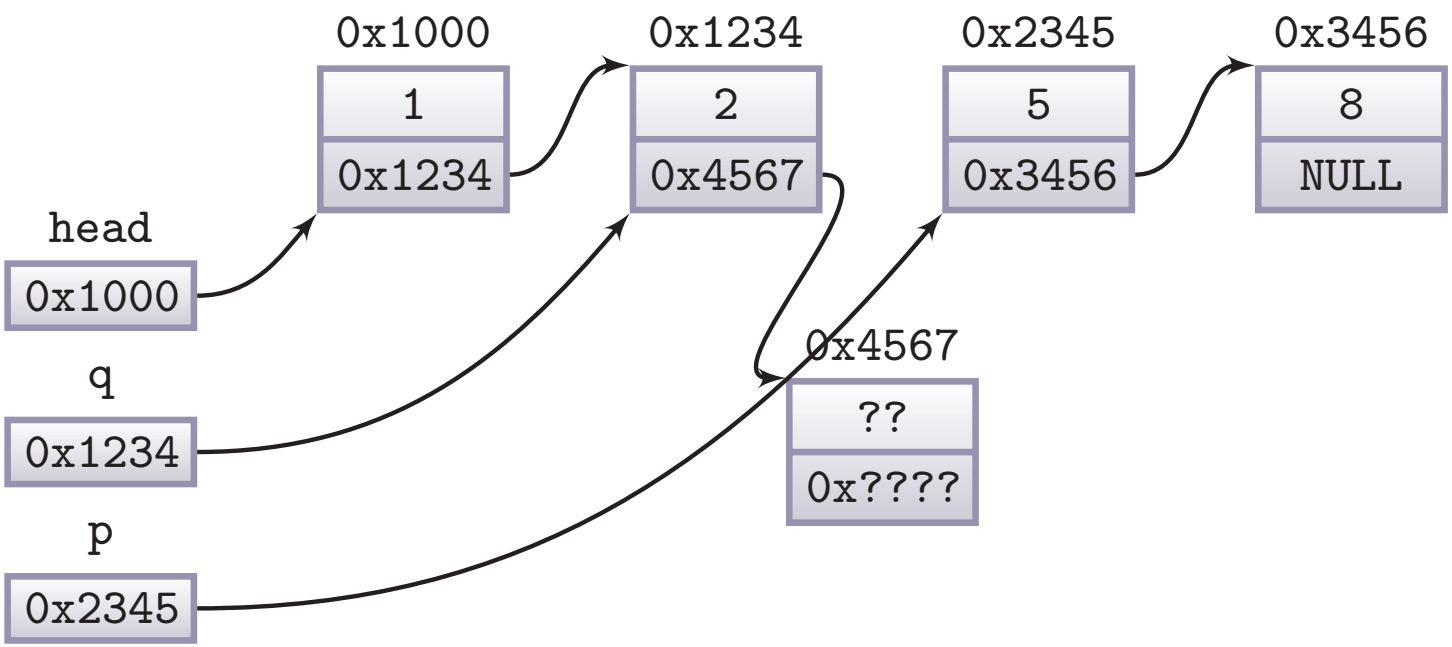
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

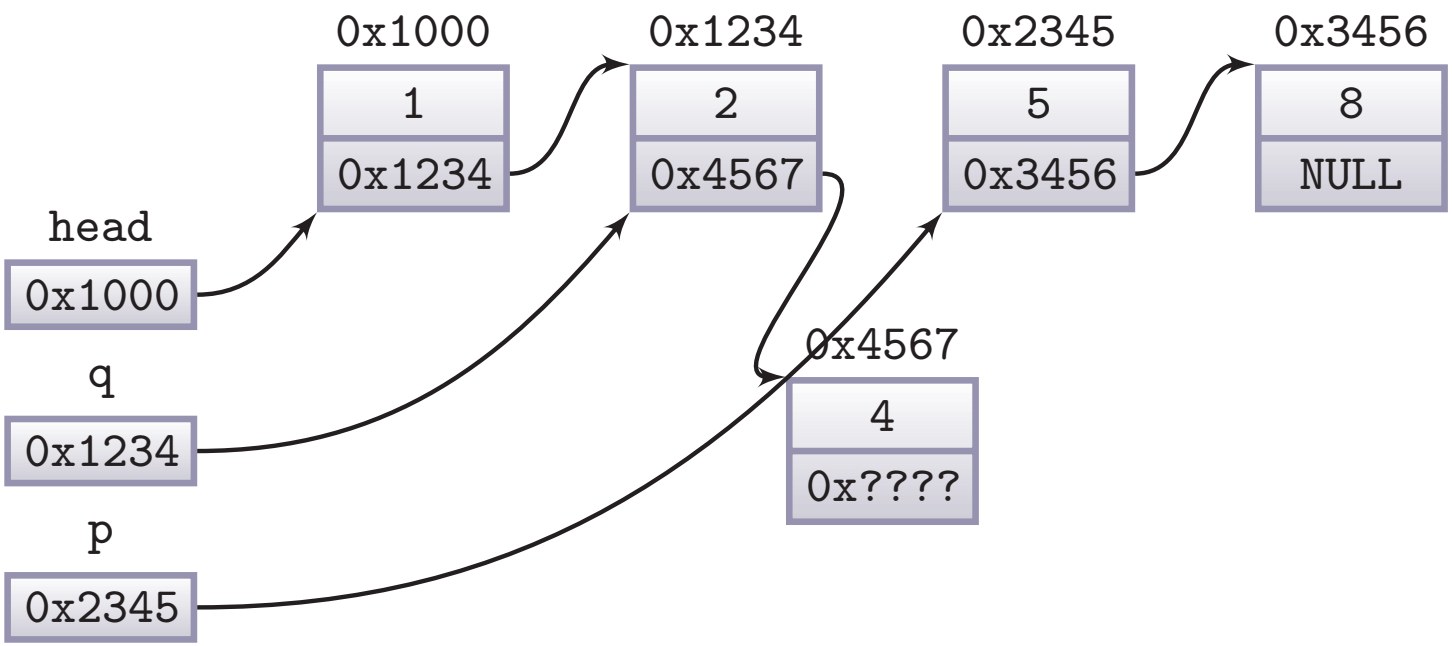
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

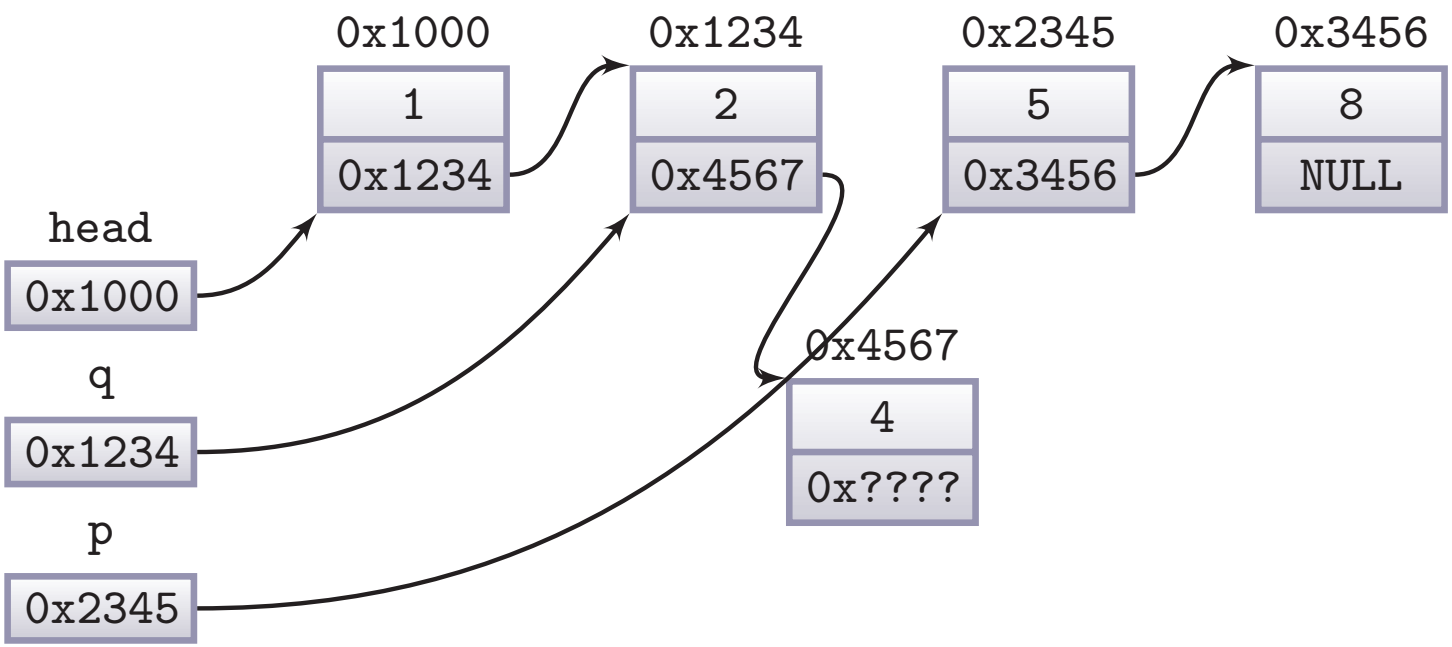
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

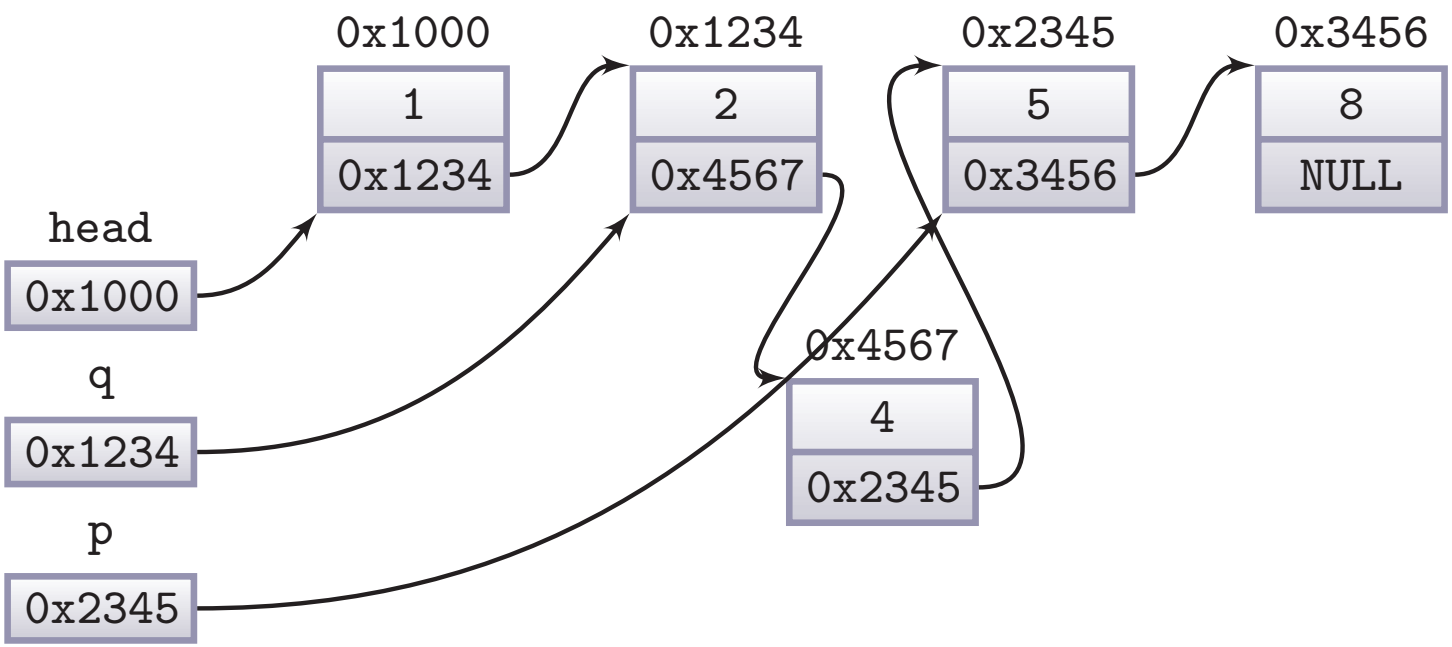
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

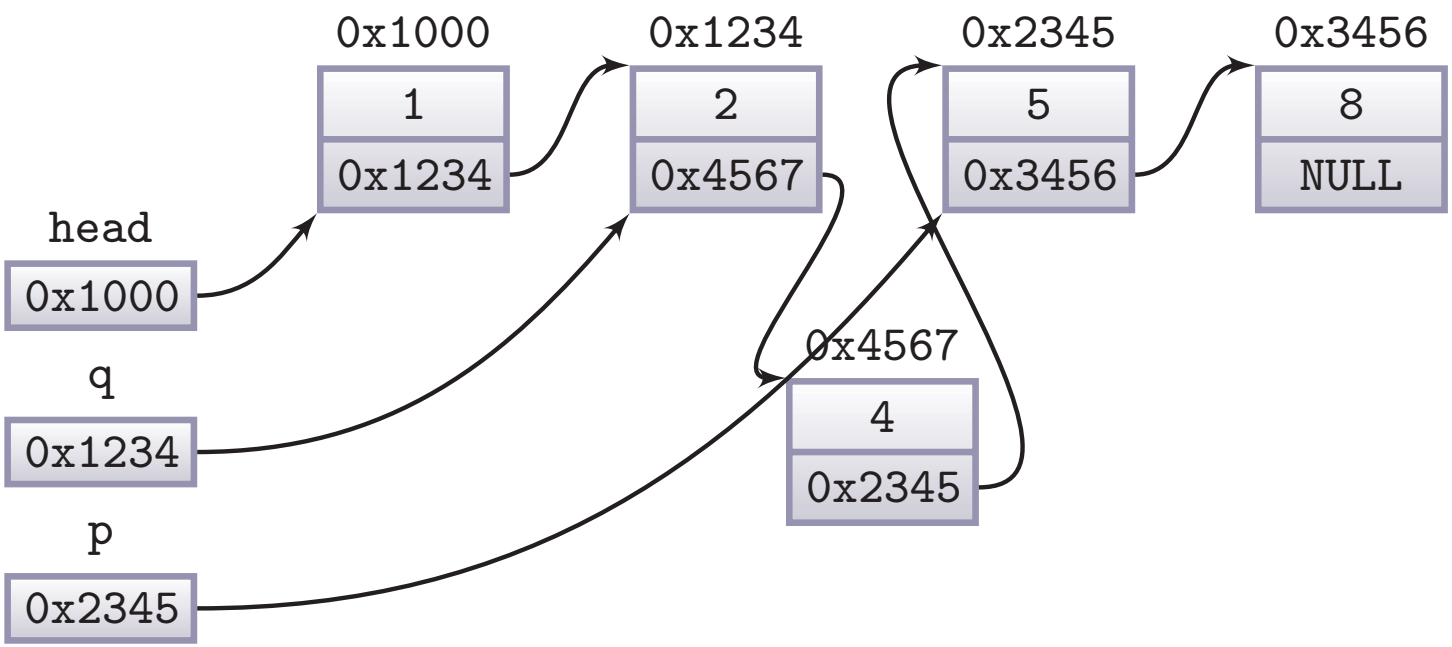
```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element (4) into a sorted list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  q = head;  p = q->next;
2  while (p != NULL && p->data <= data) { /* shortcut */
3    q = p;  p = p->next;
4  }
5  q->next = (listelem*)malloc(sizeof(listelem));
6  q->next->data = 4;
7  q->next->next = p;
```



Inserting element into a sorted list with a functio

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

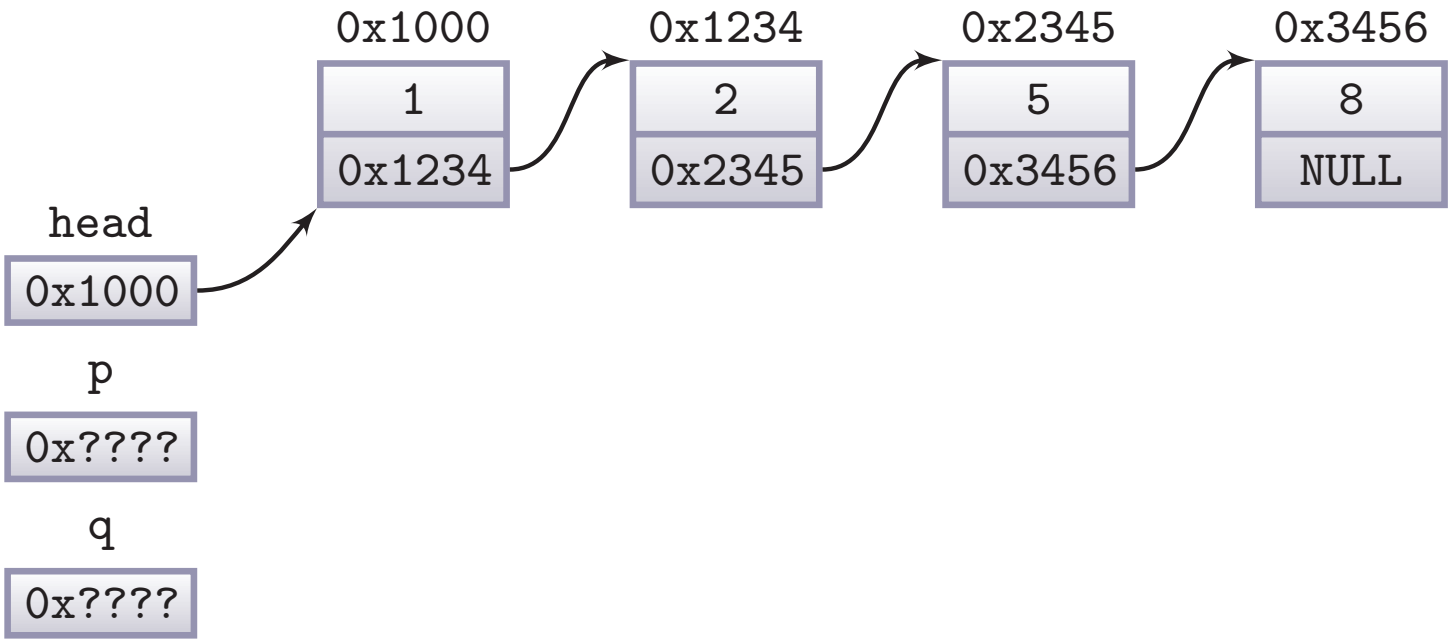
```
1  listelem *insert_sorted(listelem *head, int d)
2  {
3    listelem *p, *q;
4
5    if (head == NULL || head->data > d) /* shortcut */
6      return push_front(head, d);
7
8    q = head;
9    p = q->next;
10   while (p != NULL && p->data <= d) /* shortcut */ {
11     q = p; p = p->next;
12   }
13   q->next = (listelem*)malloc(sizeof(listelem));
14   q->next->data = d;
15   q->next->next = p;
16   return head;
17 }
```

link

Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

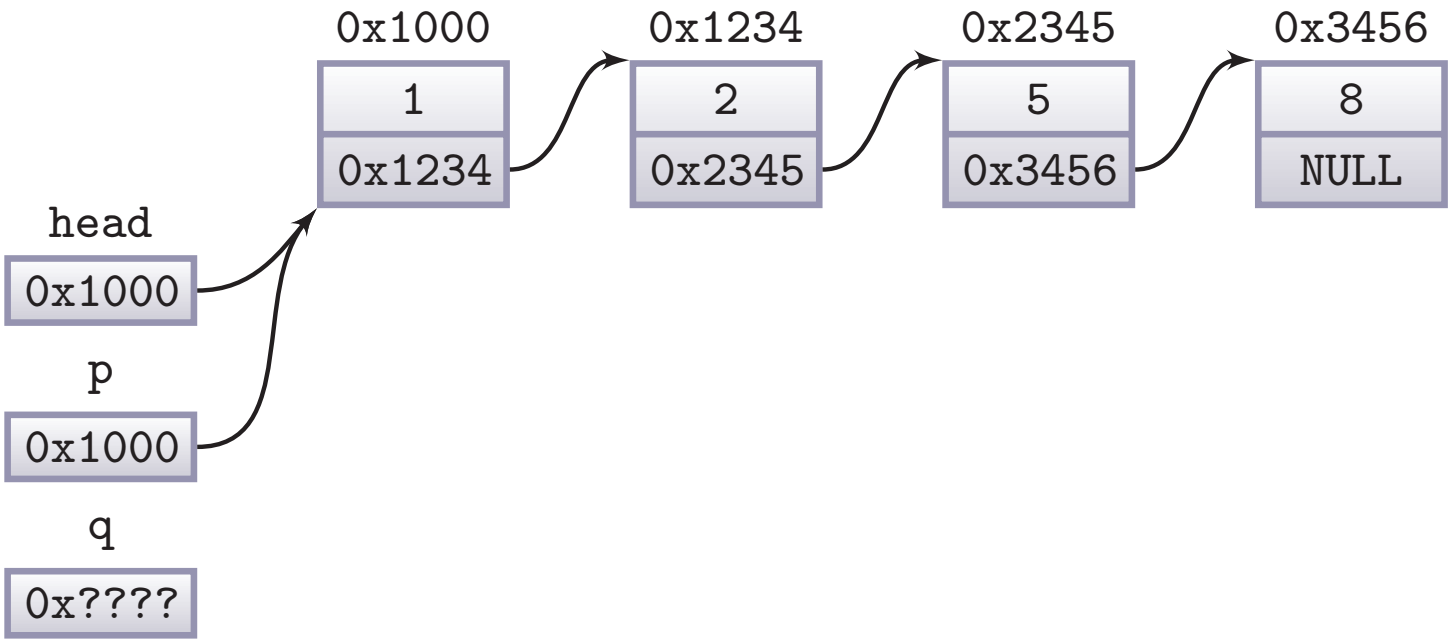
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

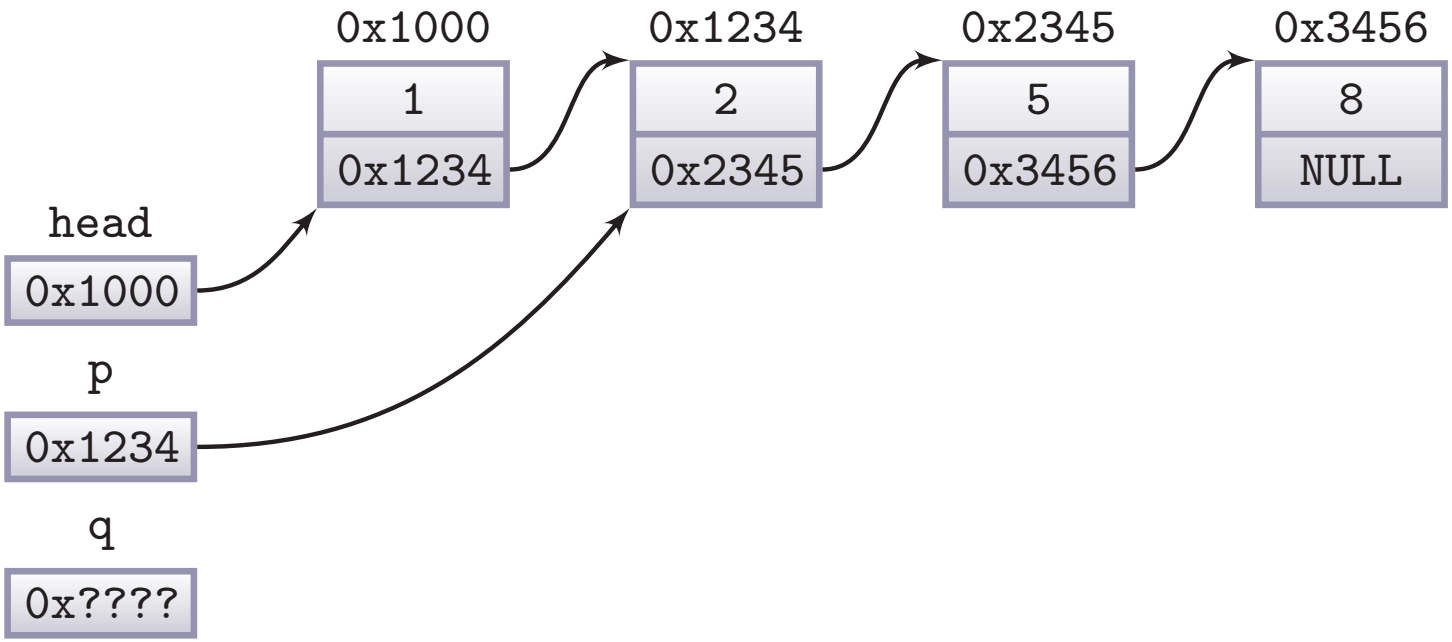
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

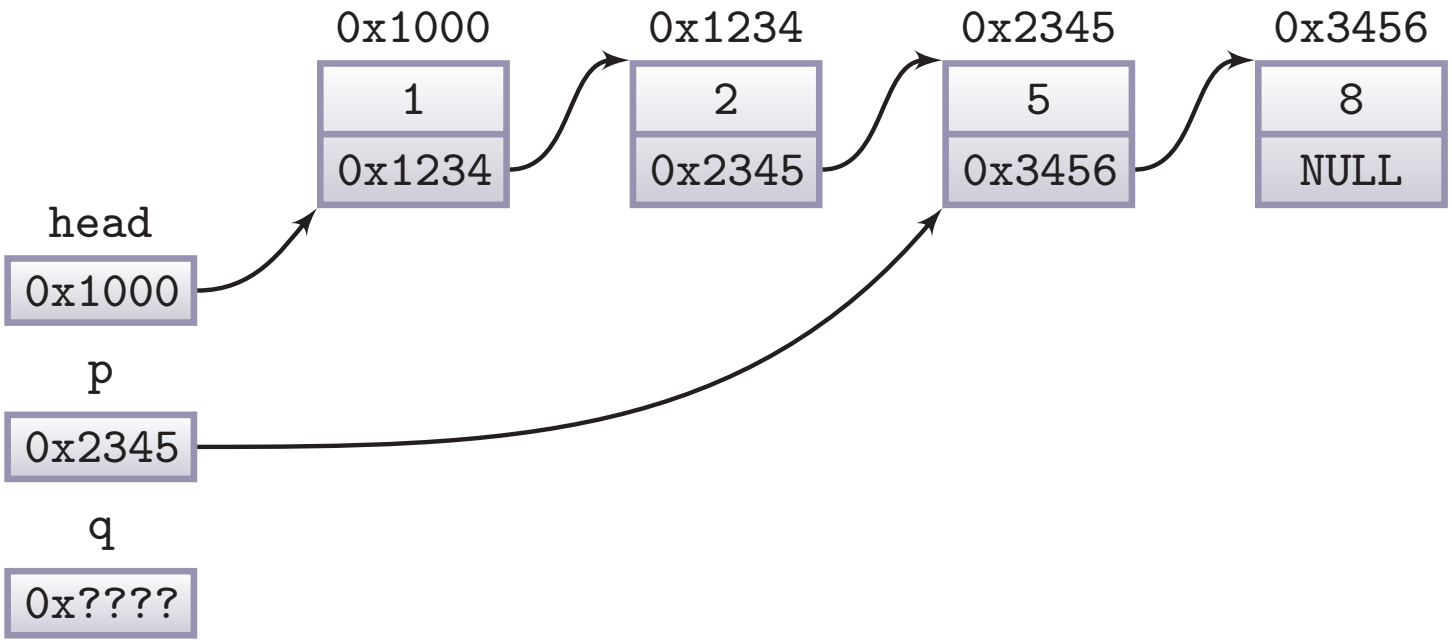
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

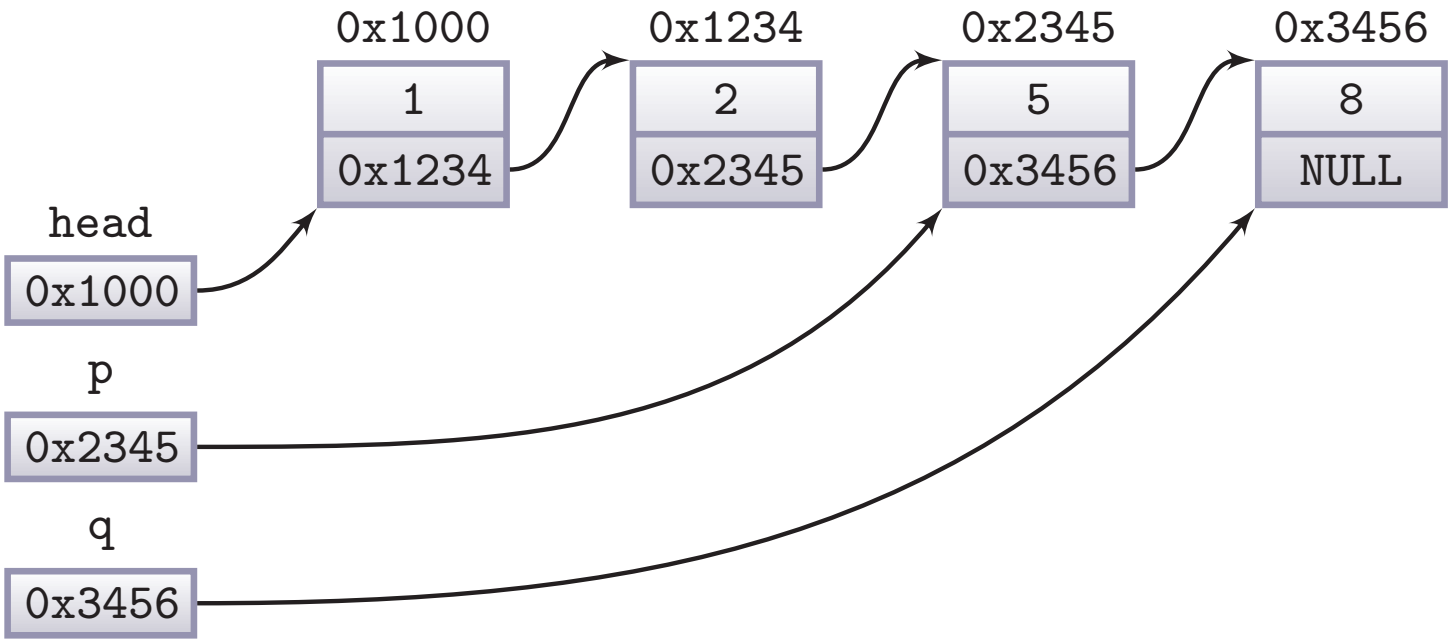
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

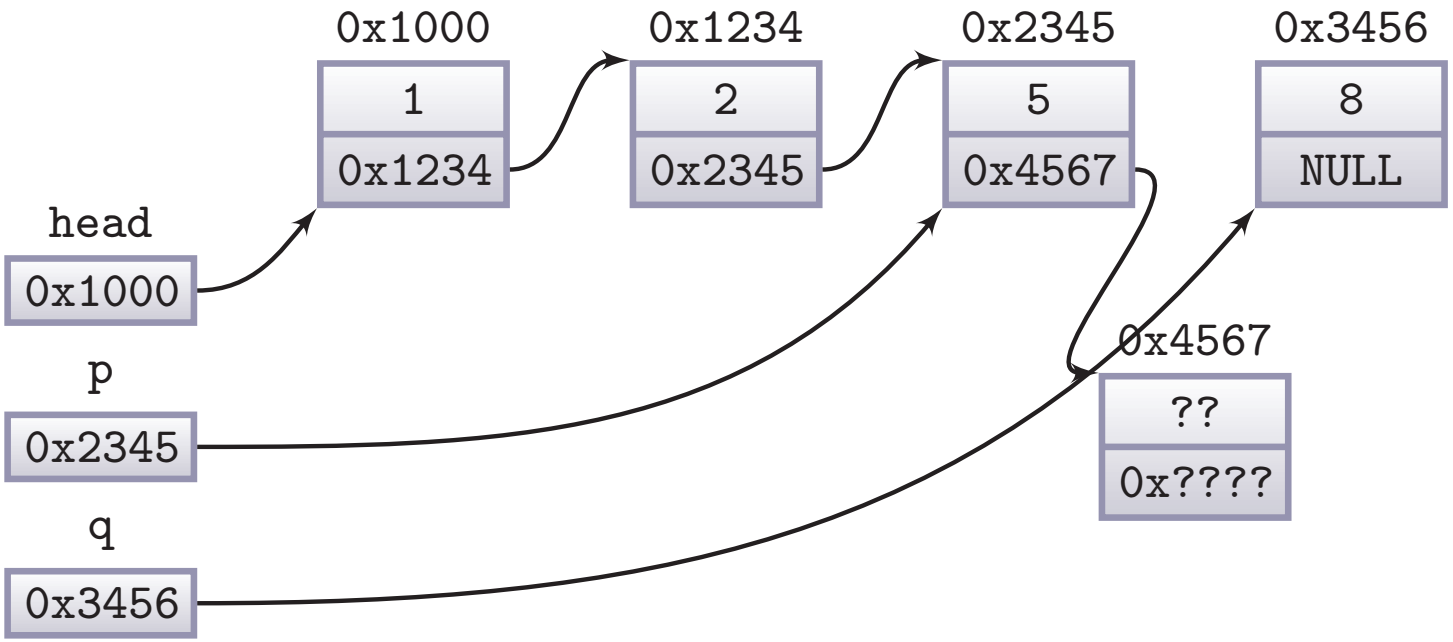
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

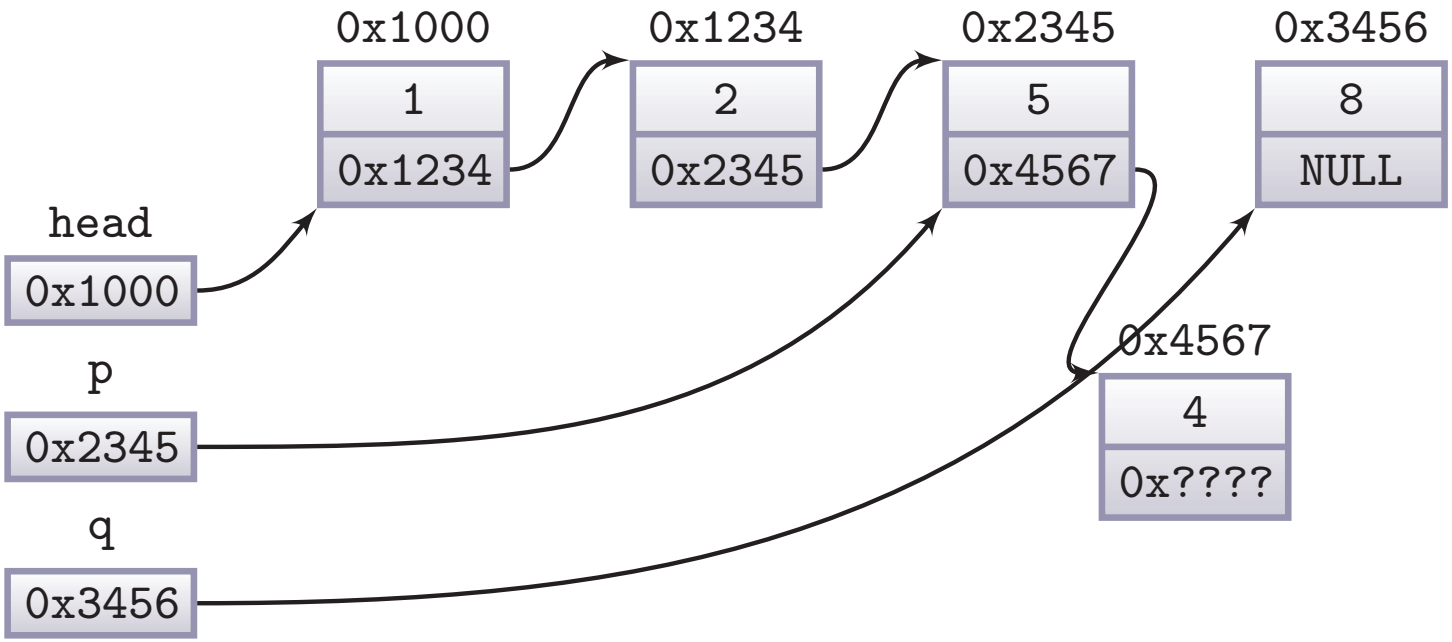
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

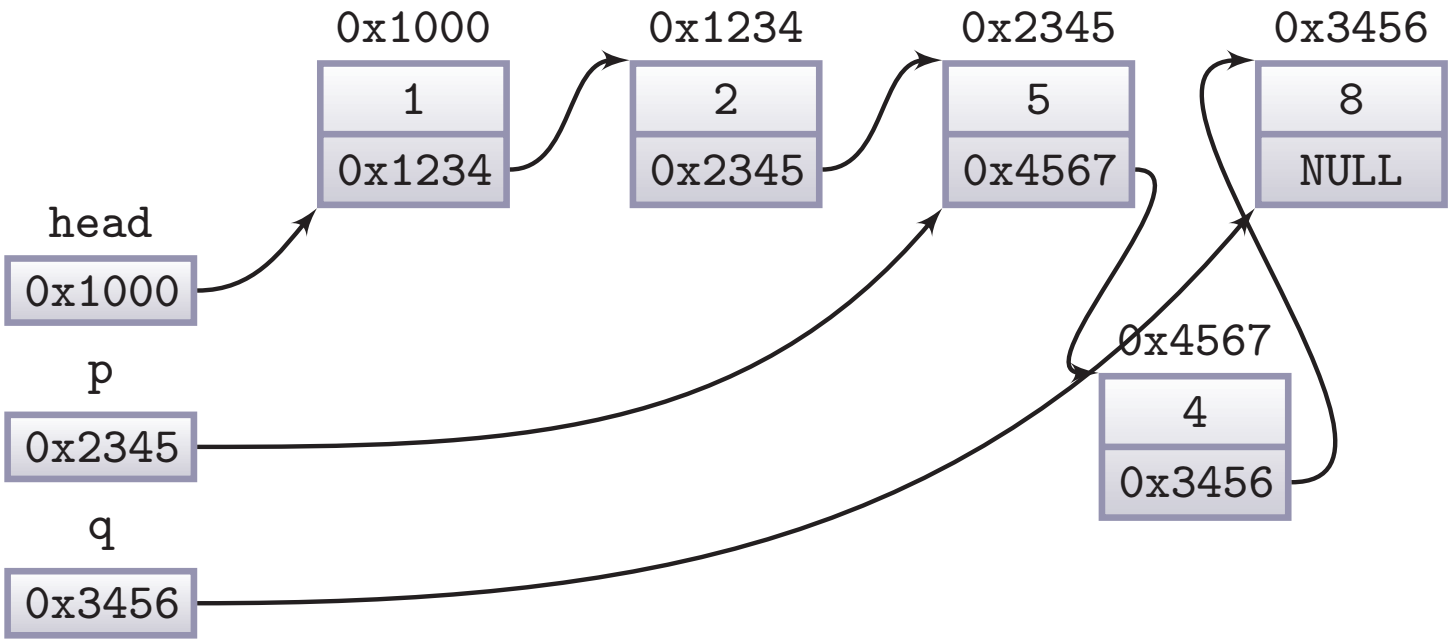
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

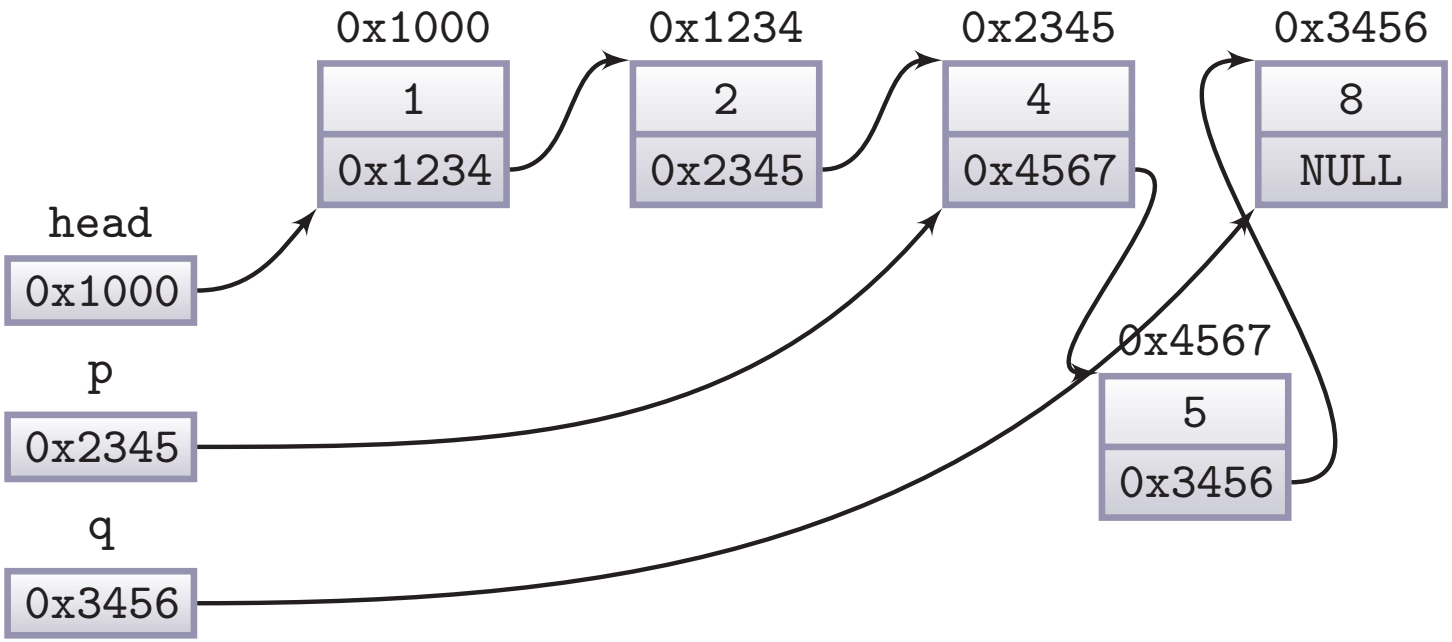
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

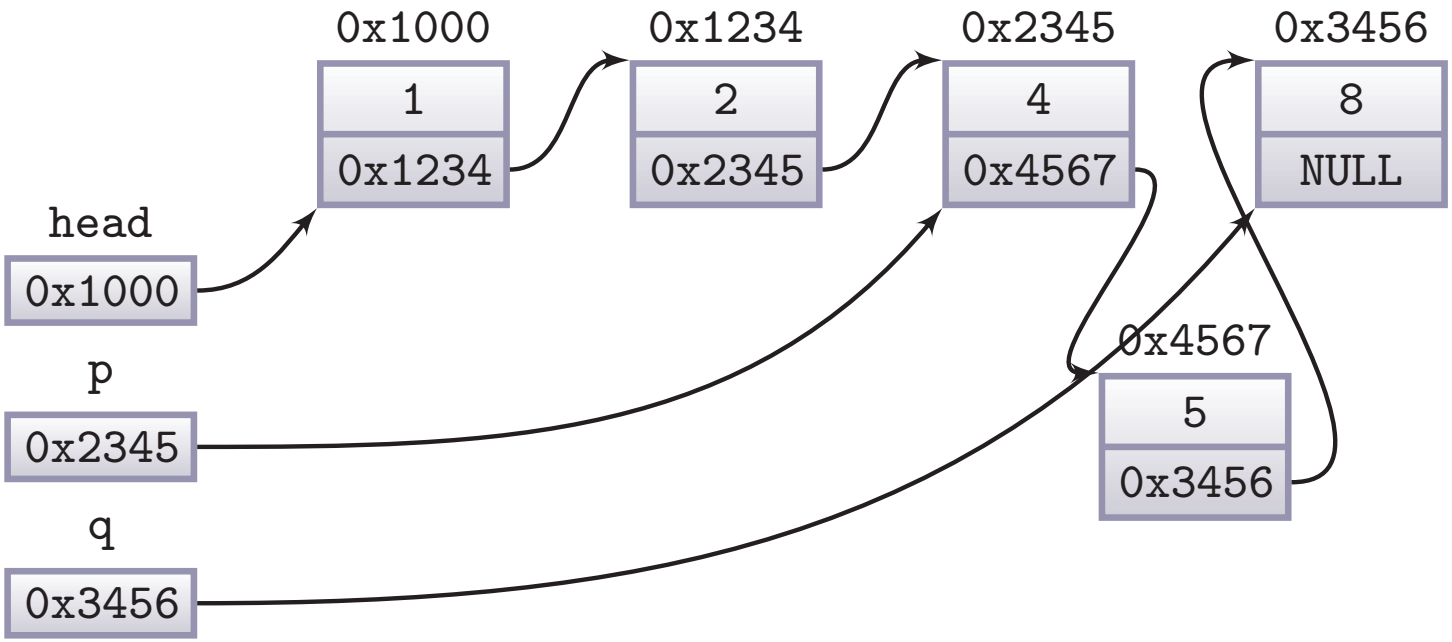
- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.



Inserting element (4) into a sorted list by replacement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- The delayed pointer can be saved (omitted), if we insert behind the selected element, and after that we replace the data.

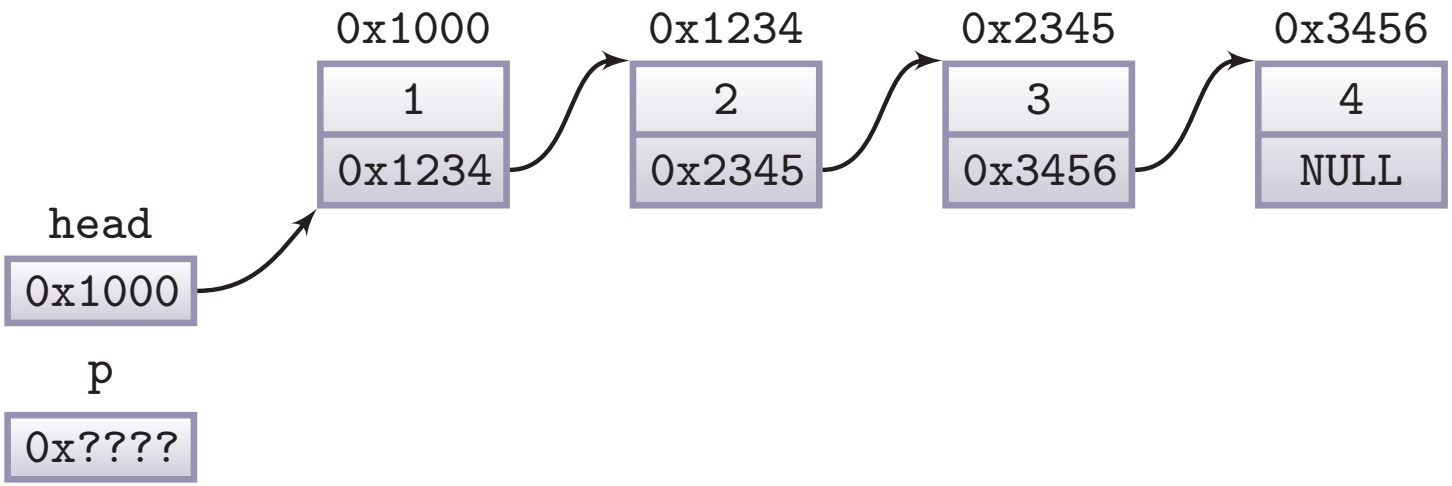


- This algorithm can be used only if we may modify the existing part of the list – others do not refer to it. But in many times this is not like that!

Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

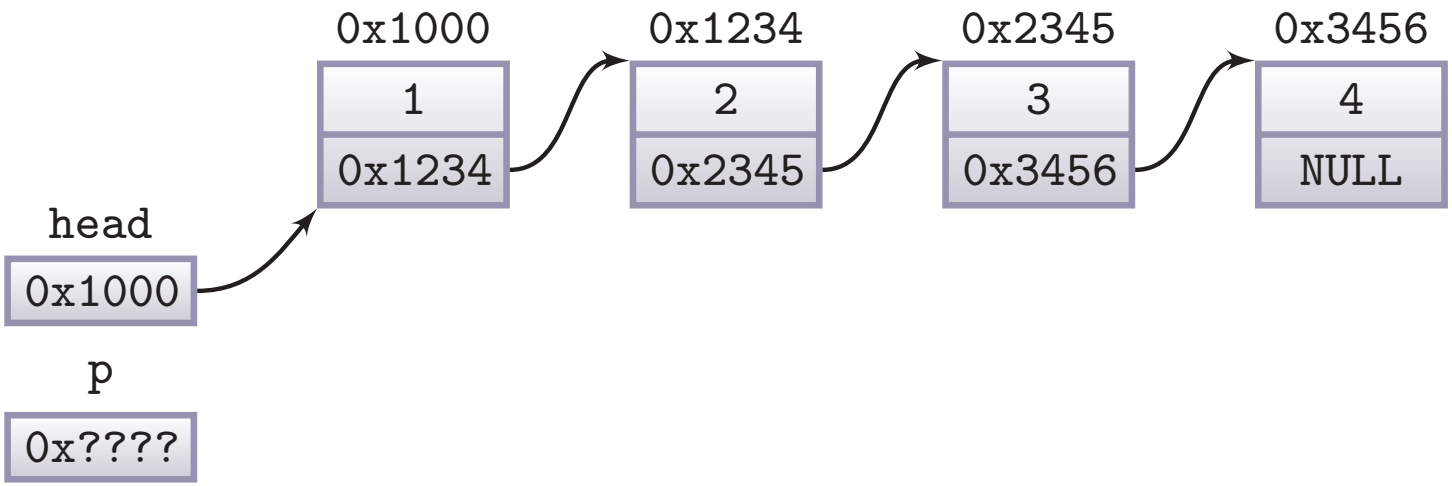
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

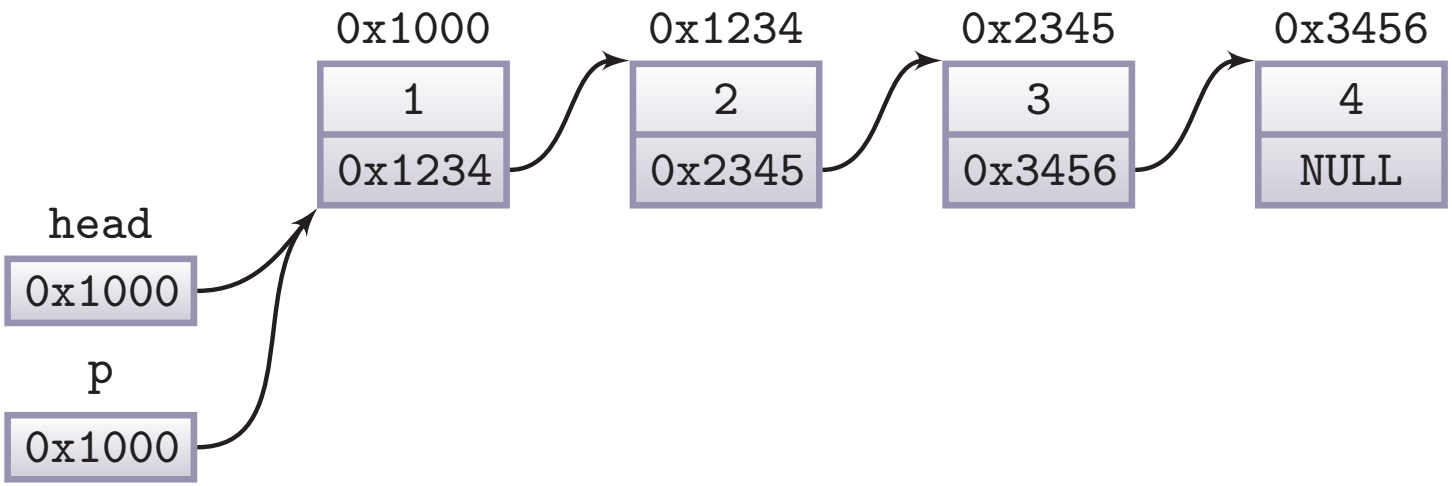
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

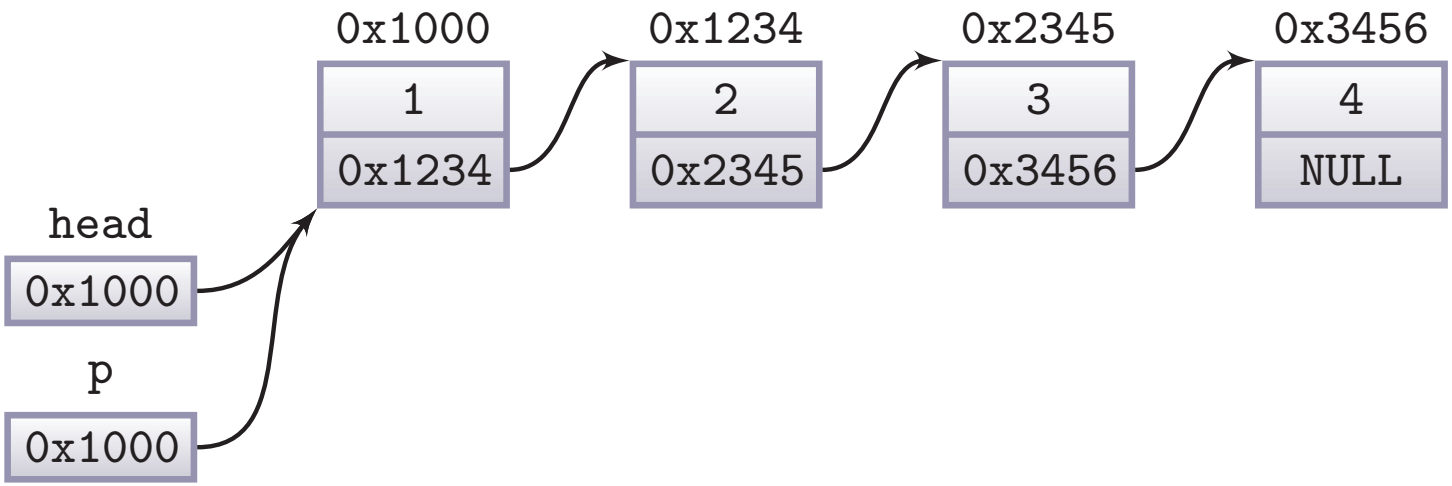
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

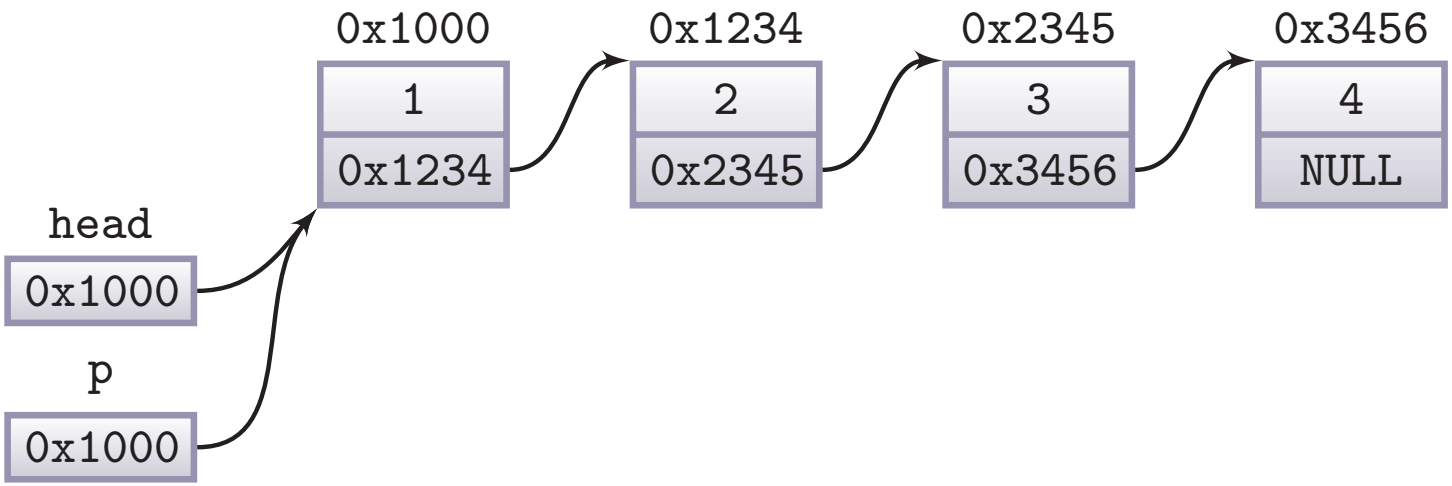
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

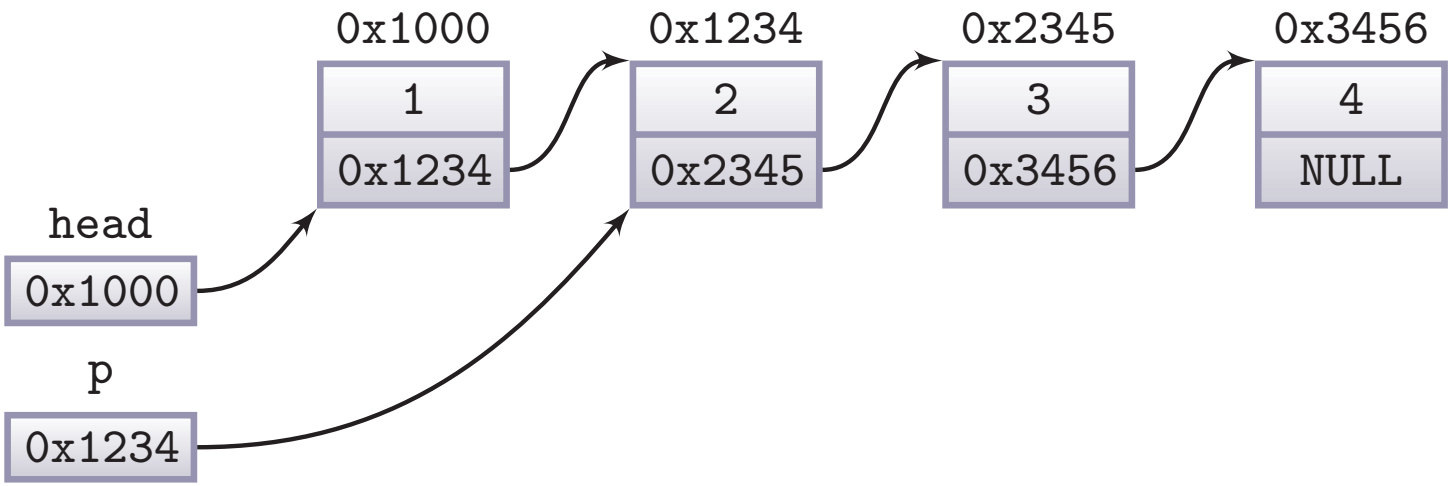
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

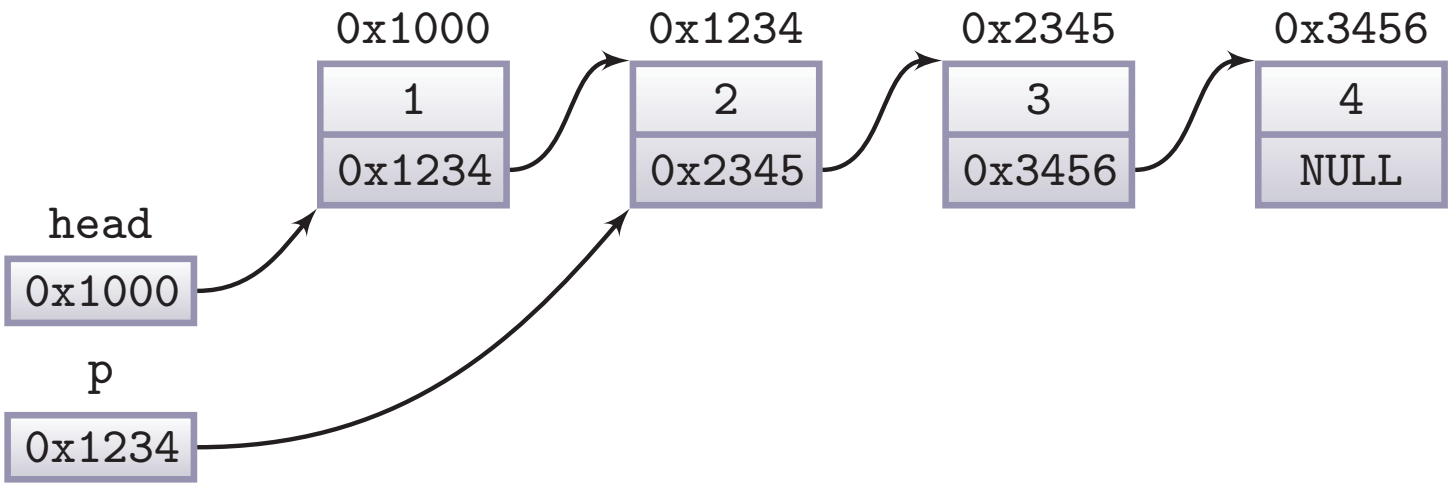
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

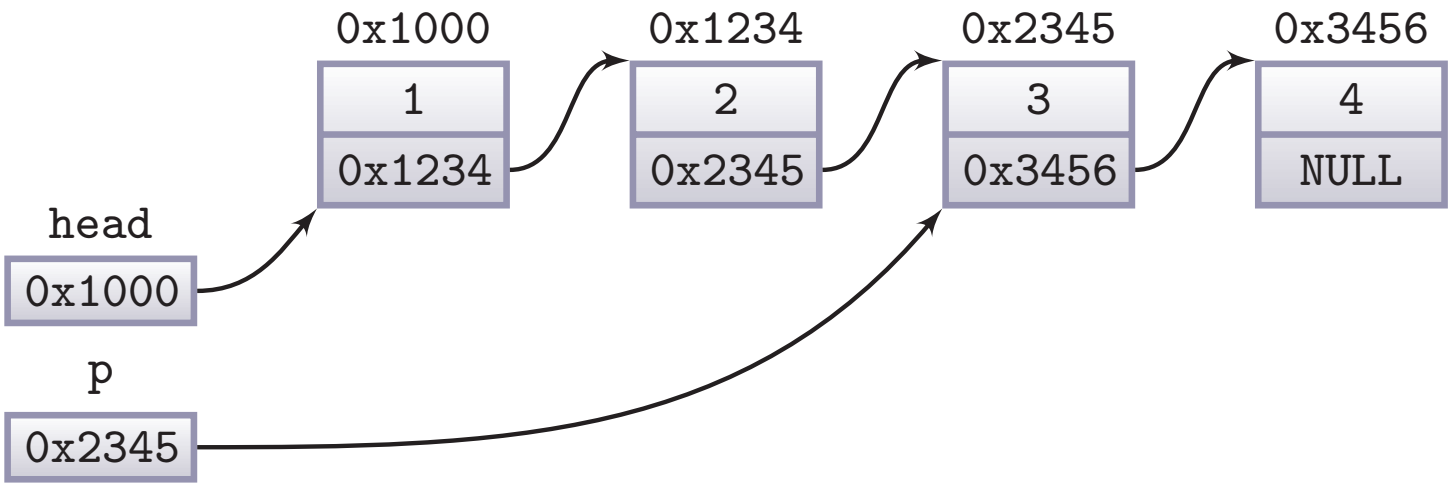
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

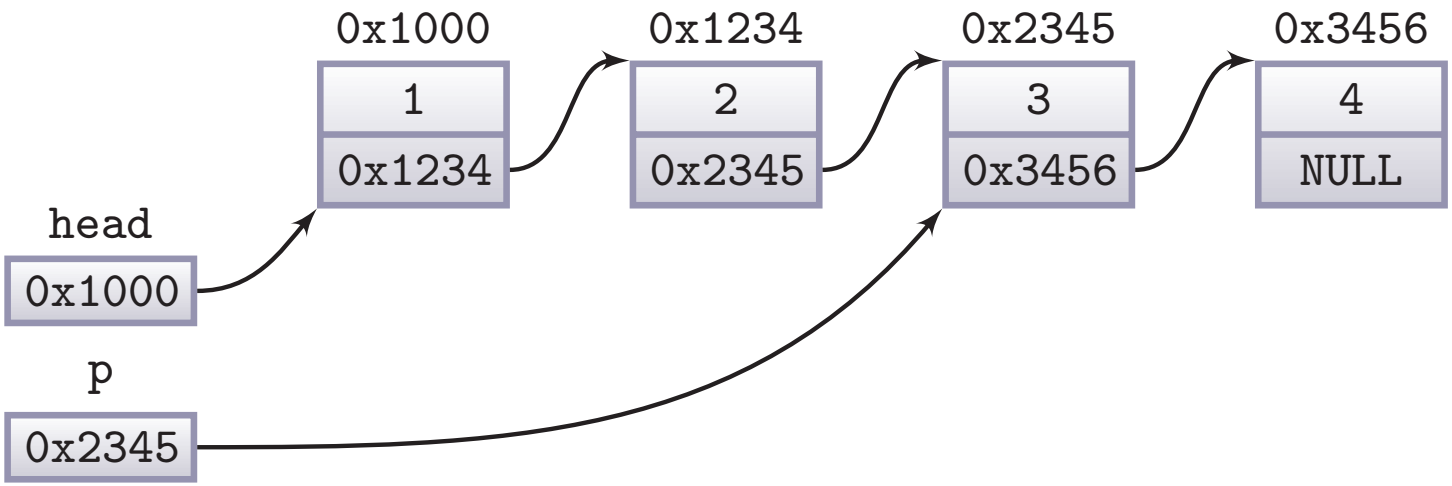
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

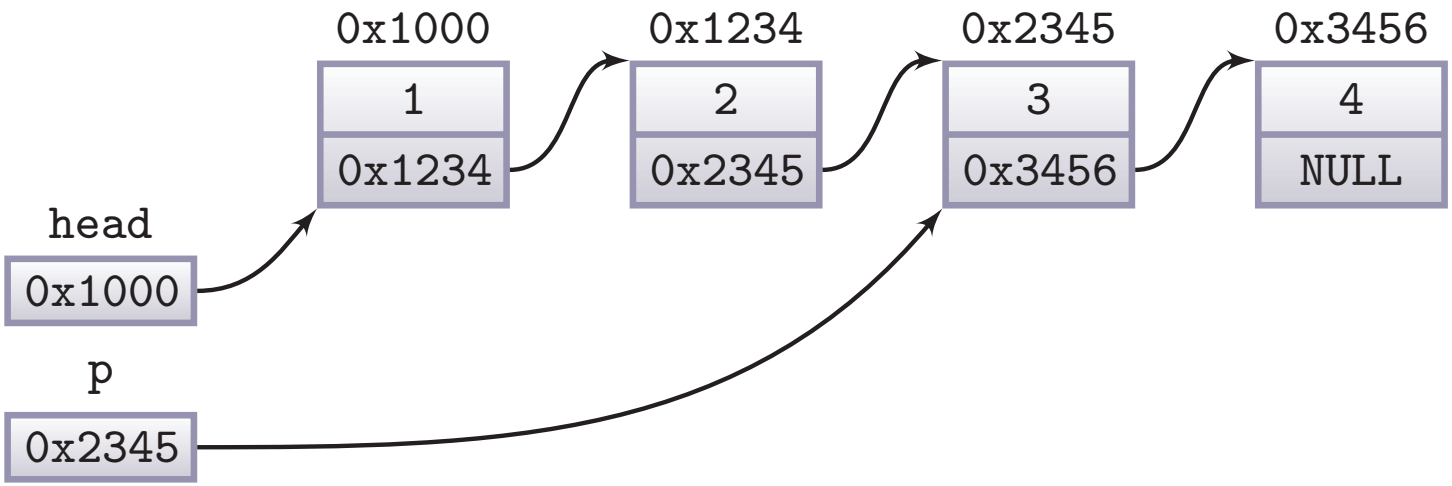
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

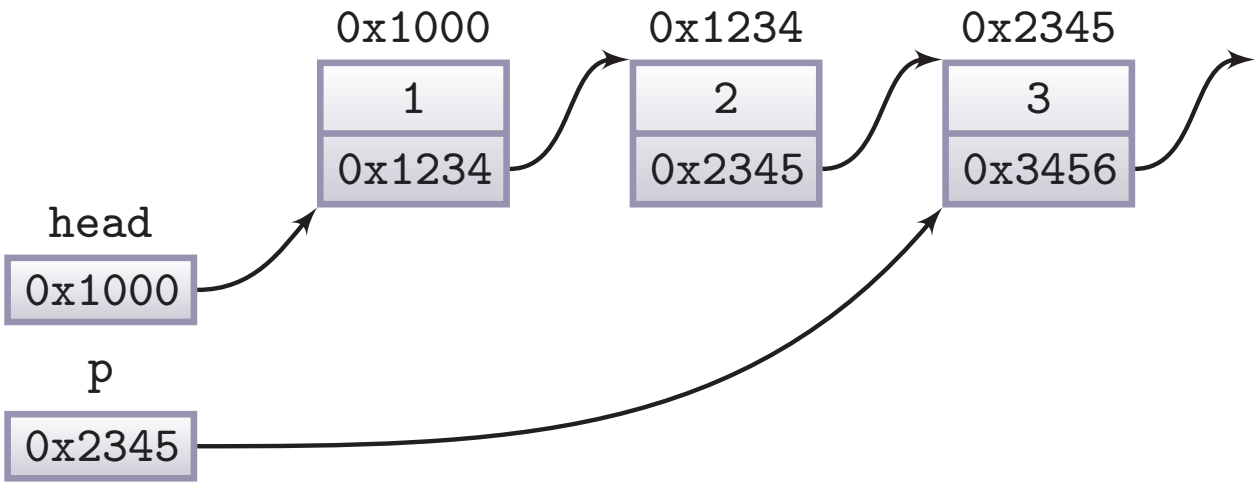
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

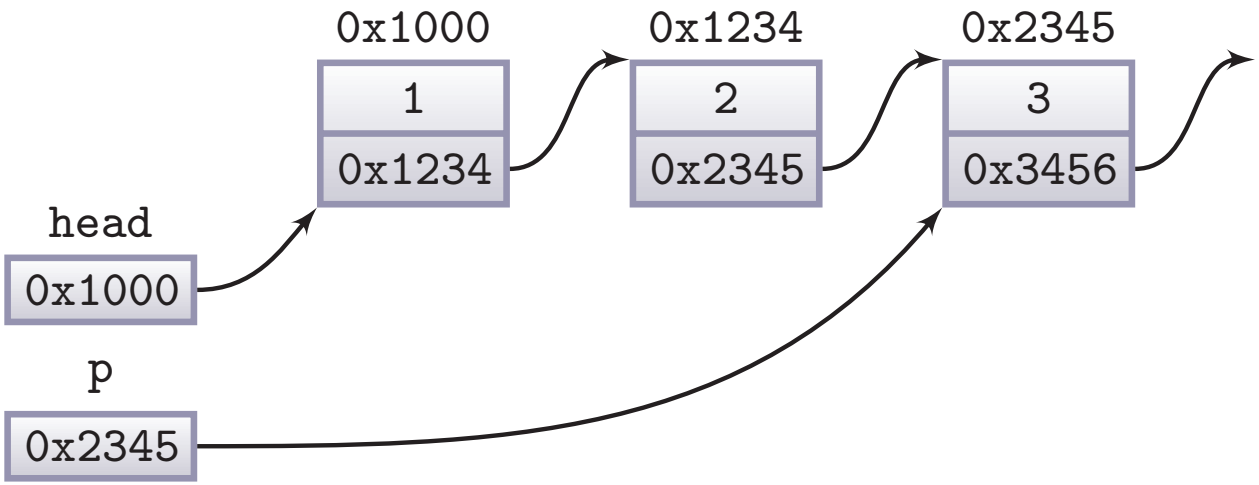
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

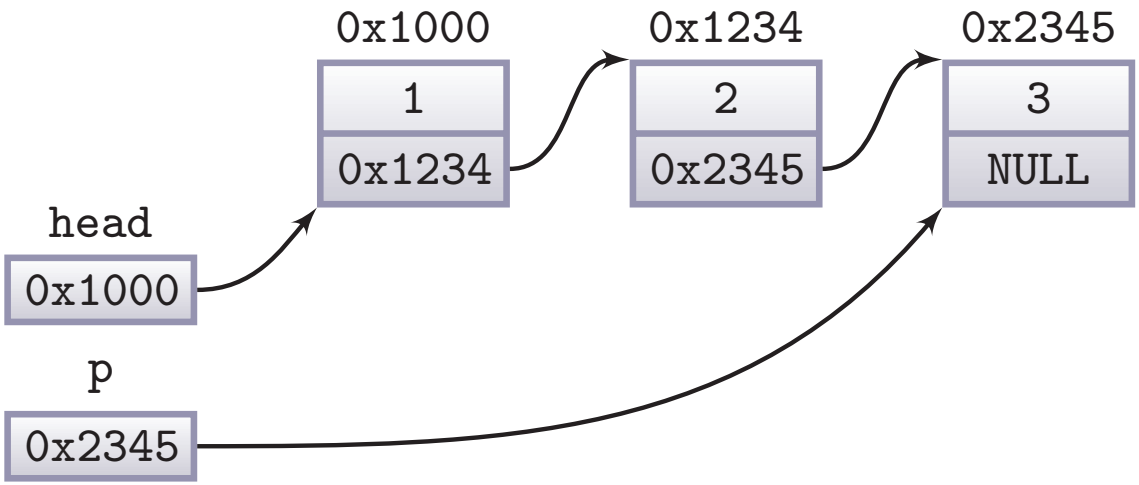
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

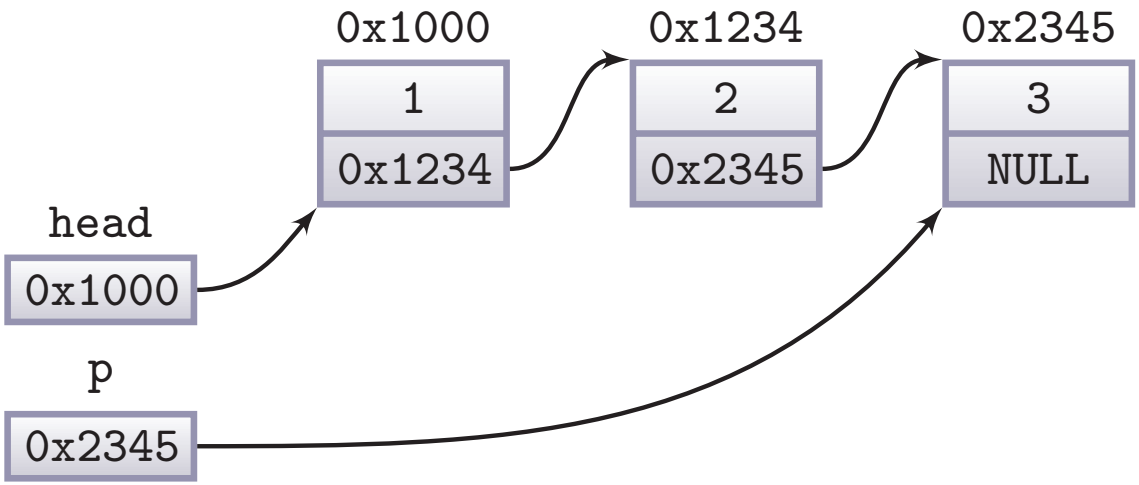
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

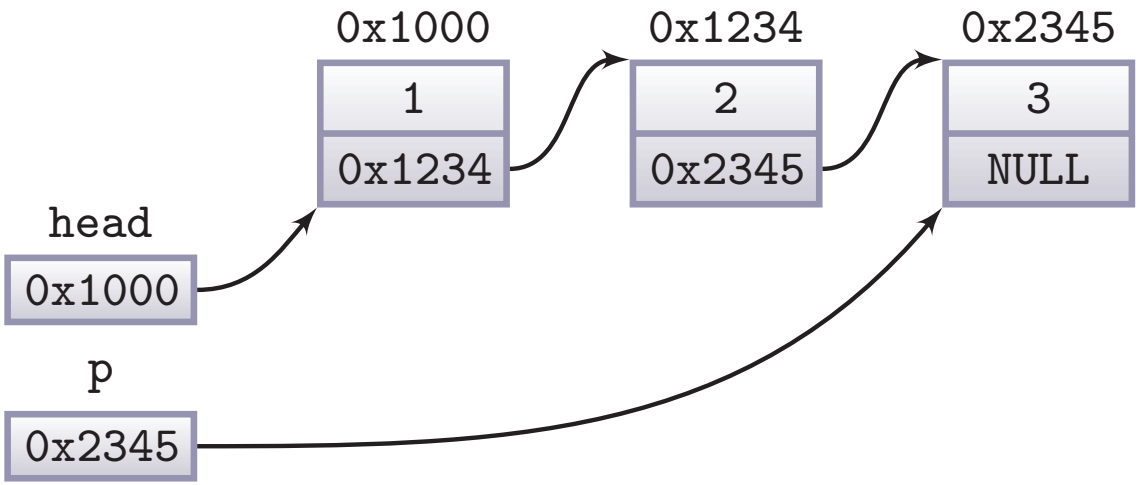
```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



Deleting element from the end of the list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  p = head;
2  while (p->next->next != NULL)
3    p = p->next;
4  free(p->next);
5  p->next = NULL;
```



- If the list is empty or it contains only one element, the expression `p->next->next` doesn't make any sense.

Deleting element from the end of the list with a function

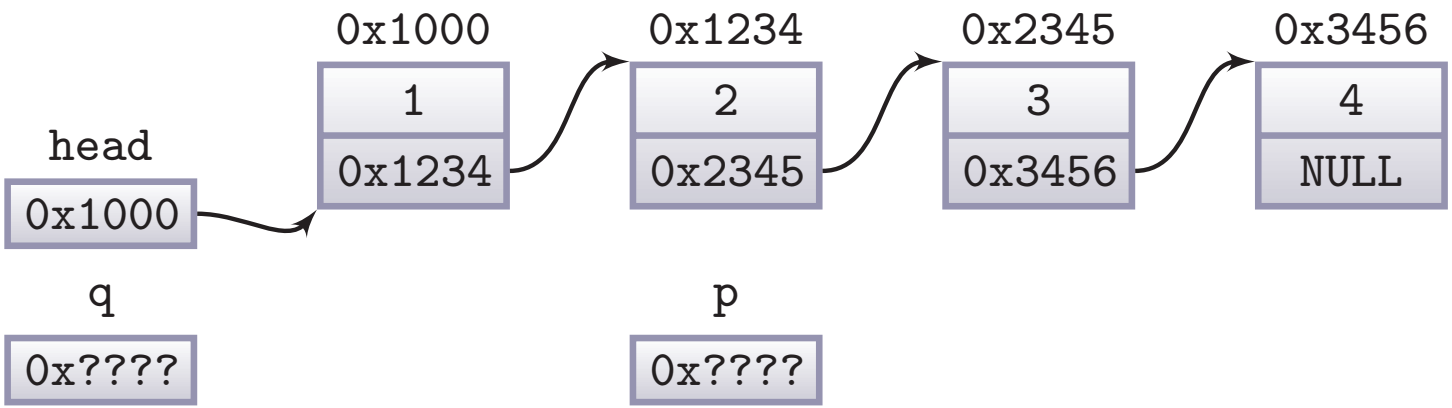

```
1  listelem *pop_back(listelem *head)
2  {
3    listelem *p;
4
5    if (head == NULL ) /* empty */
6      return head;
7
8    if (head->next == NULL) /* only one element */
9      return pop_front(head);
10
11   for (p = head; p->next->next != NULL; p = p->next);
12   free(p->next);
13   p->next = NULL;
14   return head;
15 }
```

link

Deleting a given element from list

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

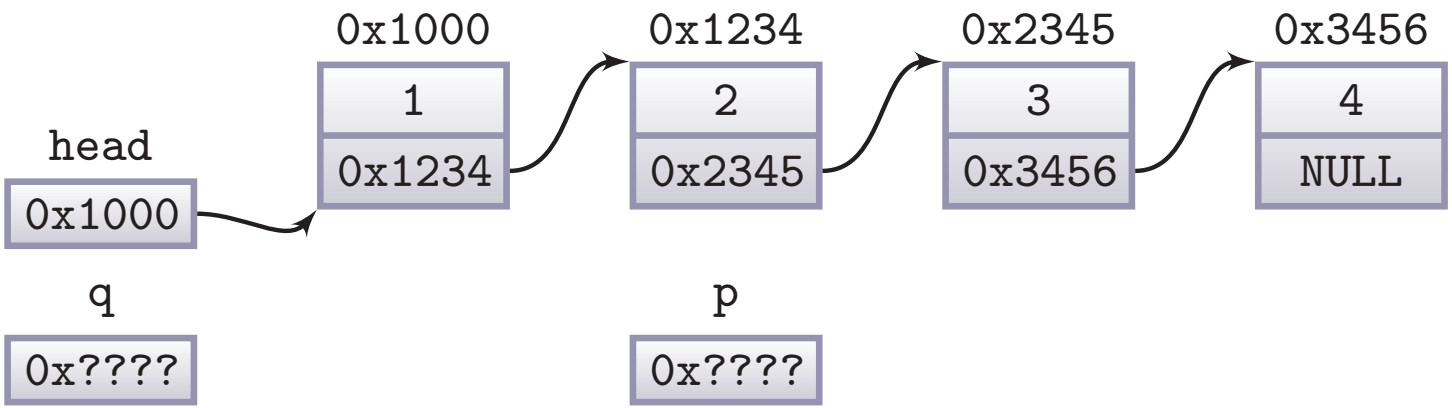


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

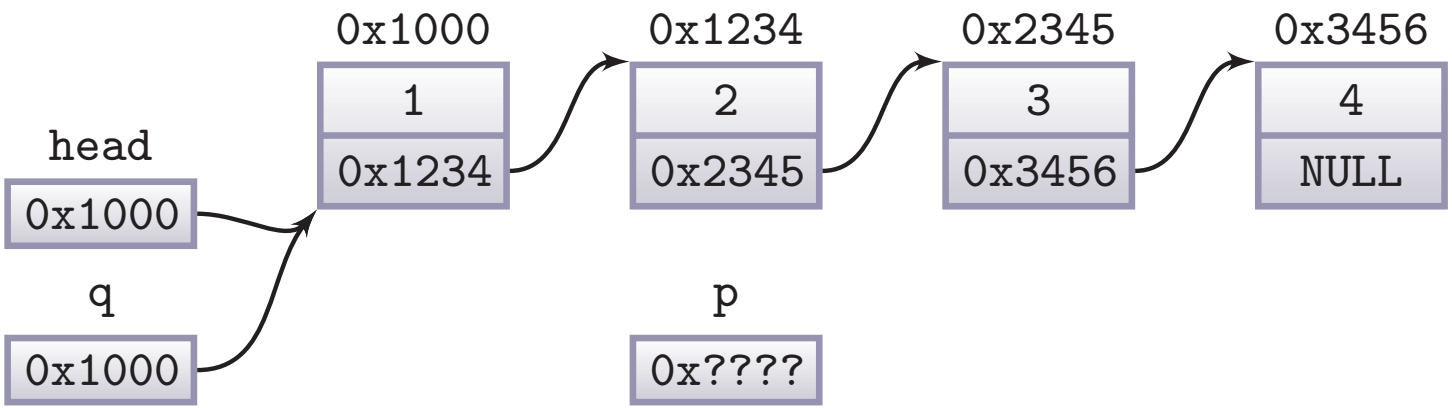


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

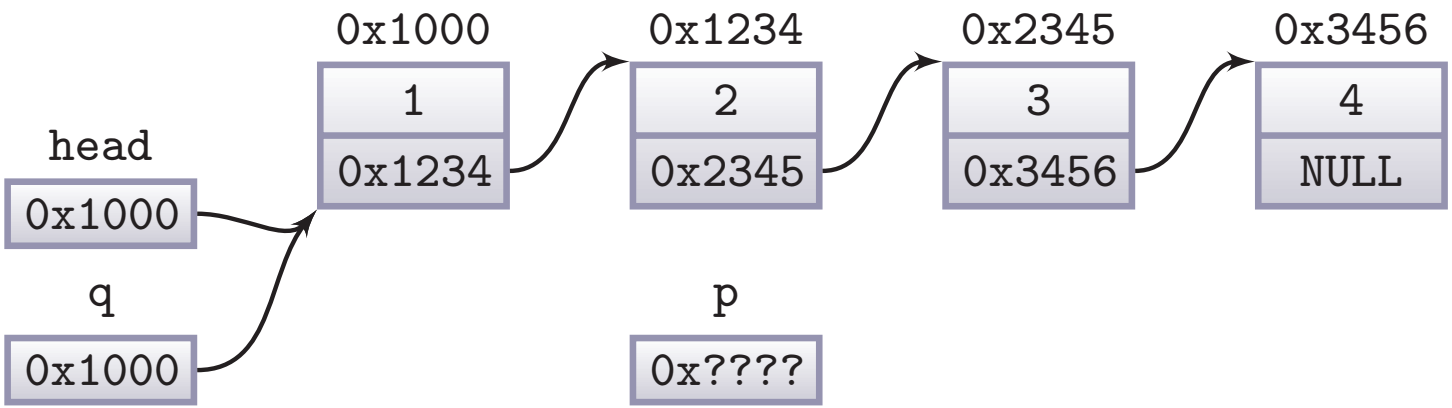


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

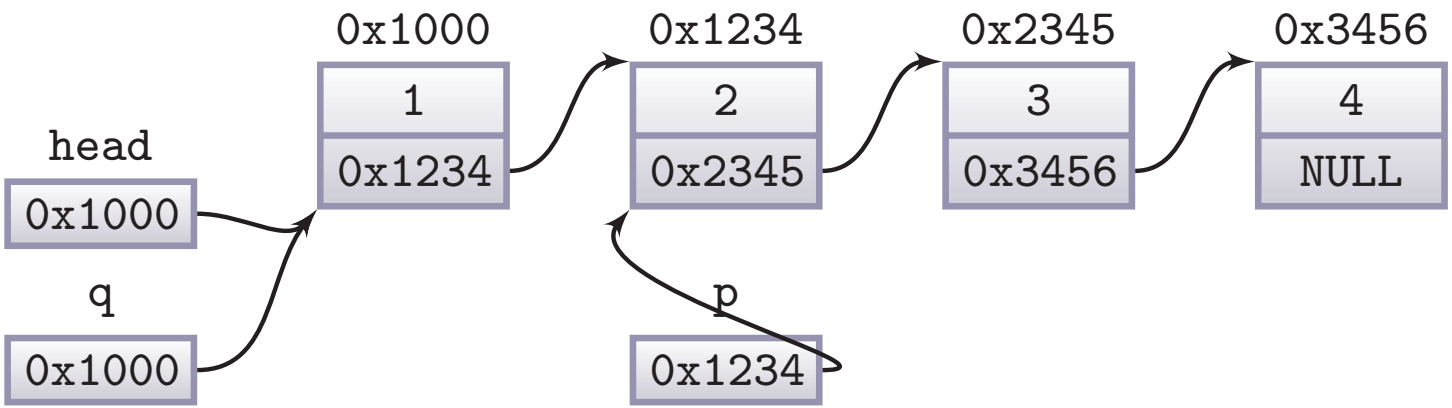


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

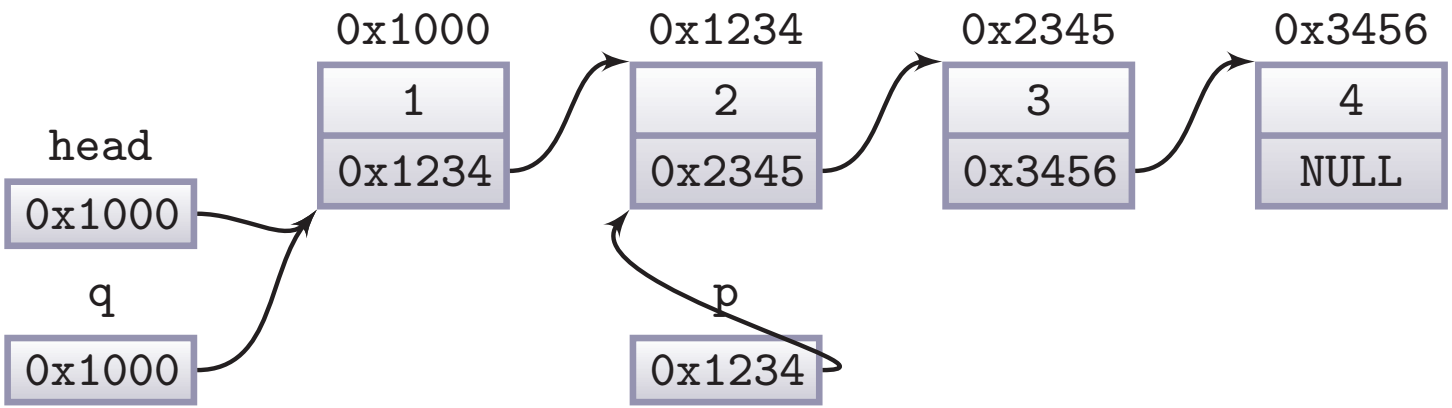


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

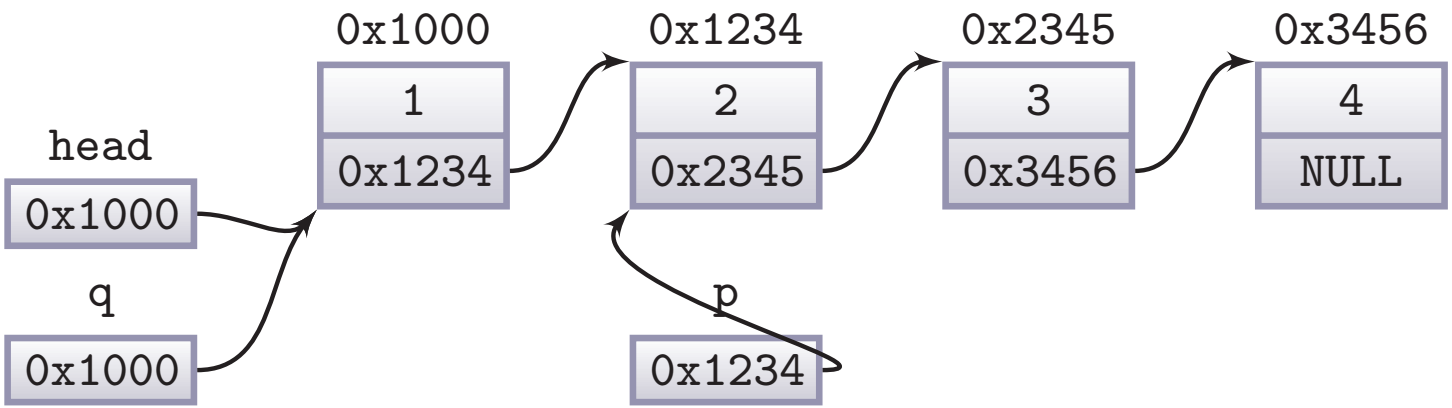


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

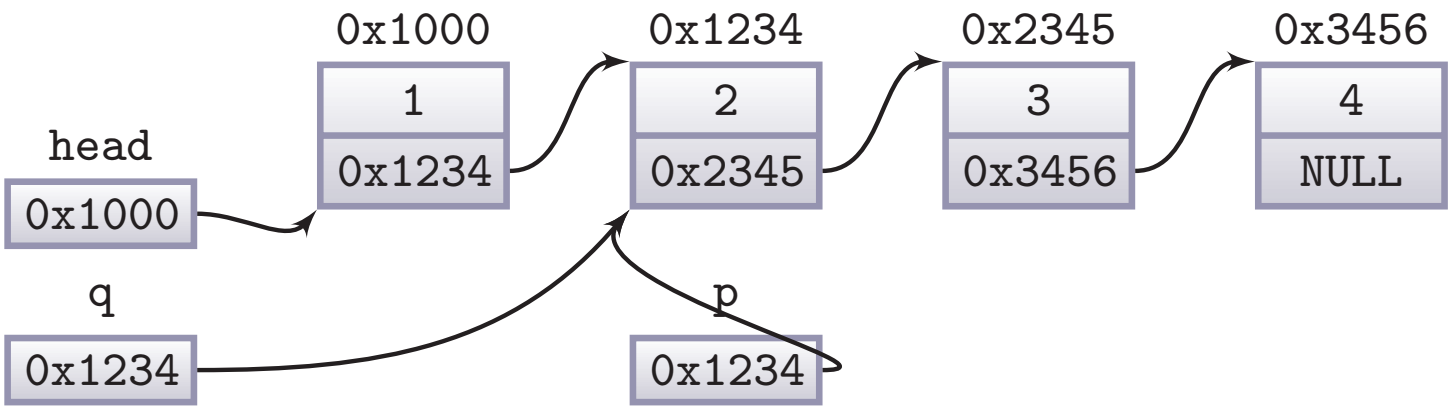


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

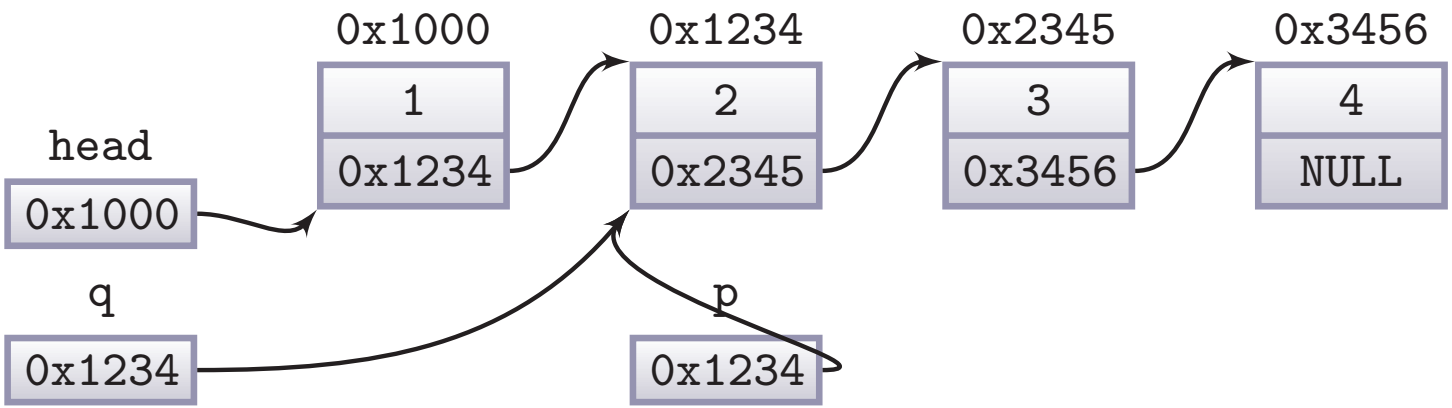


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

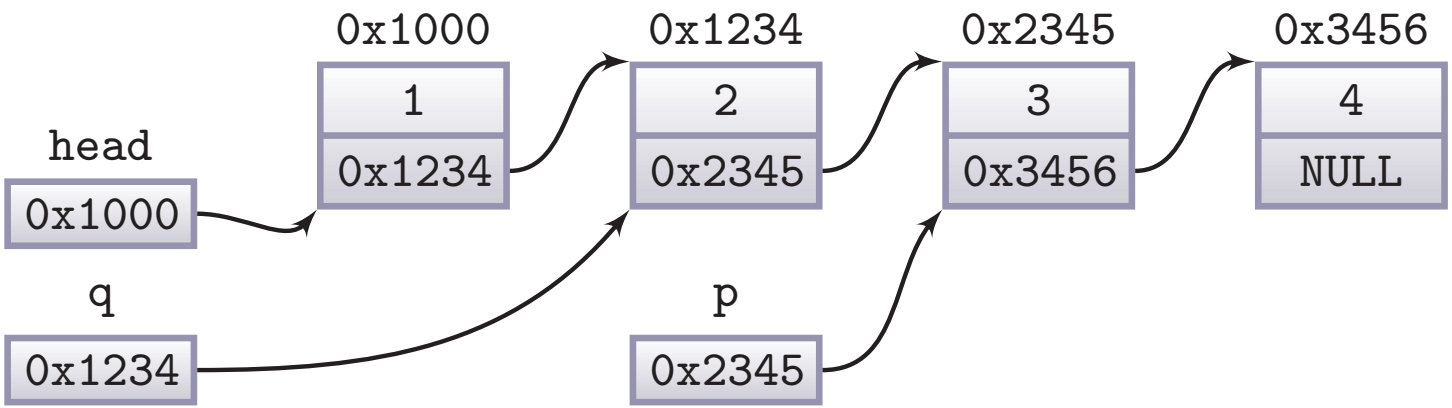


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

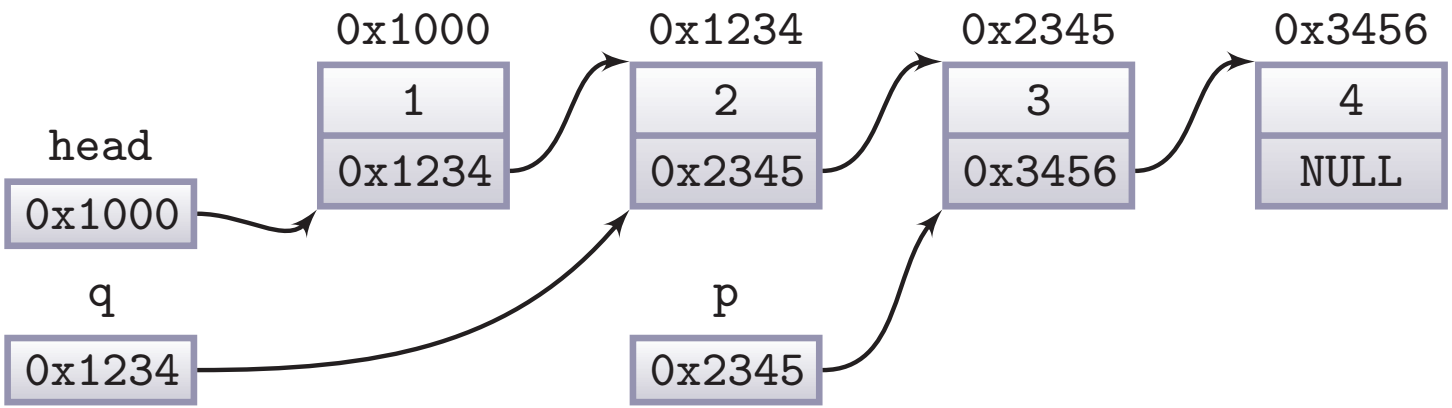


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

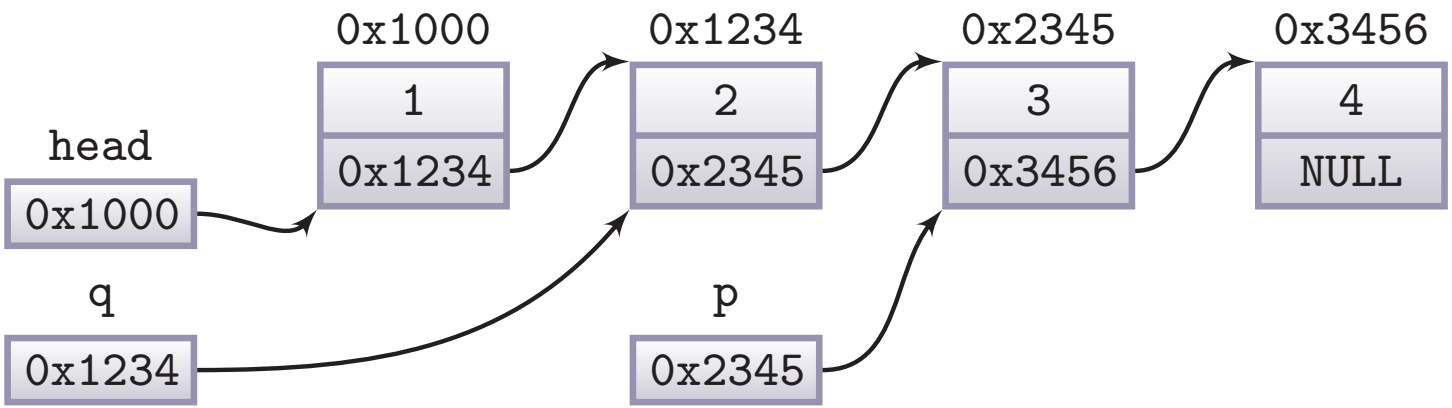


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

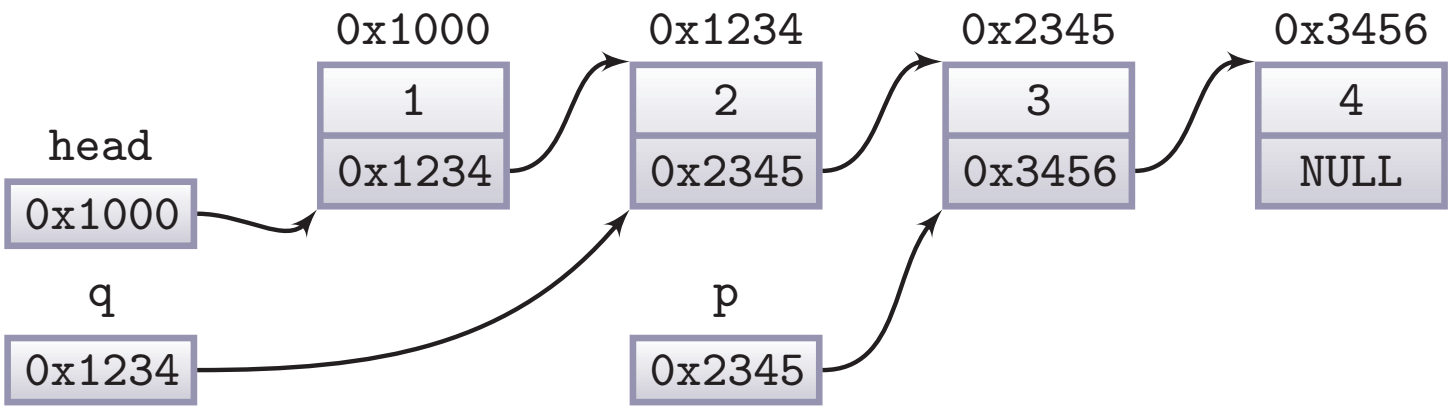


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

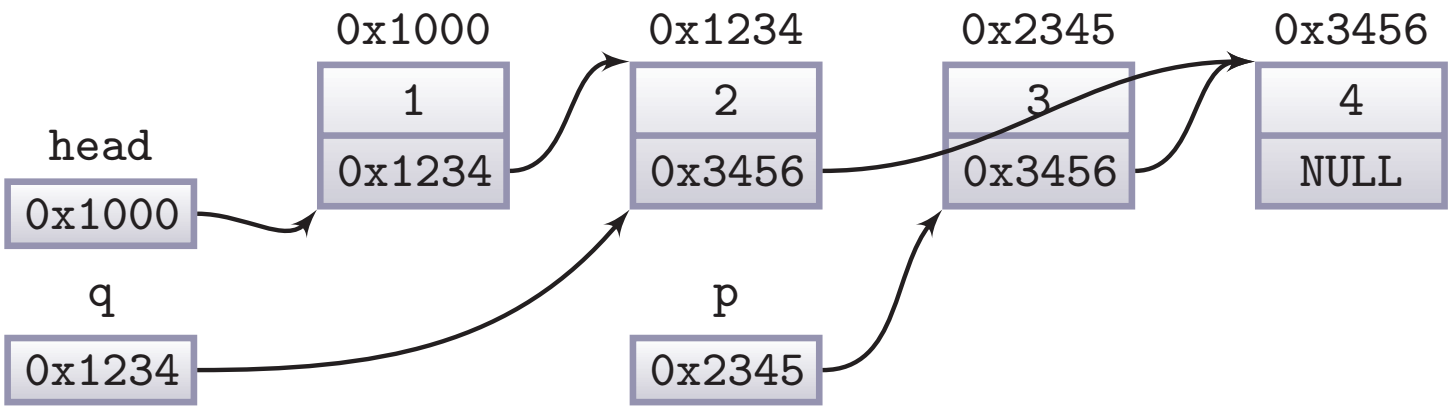


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

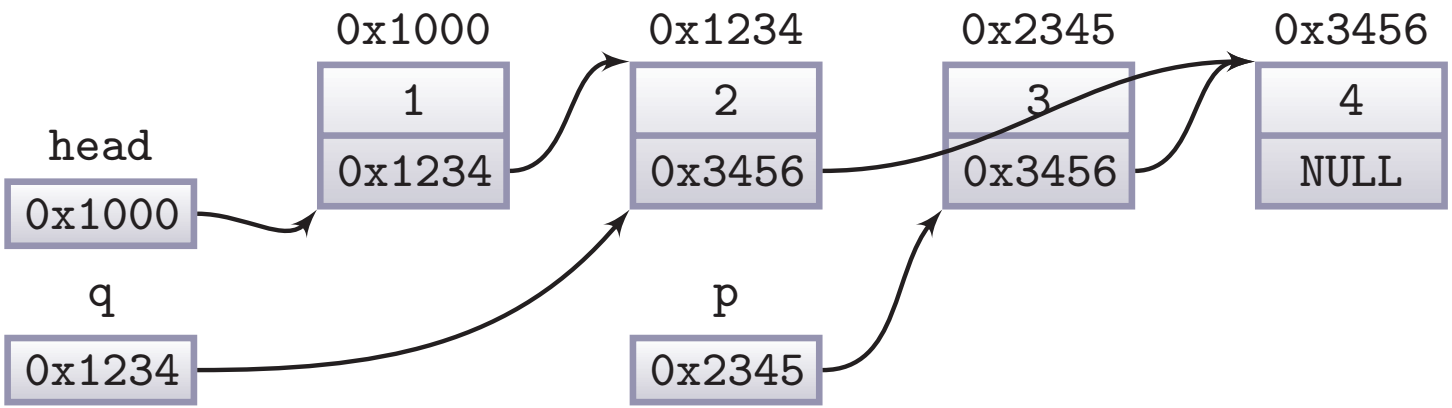


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

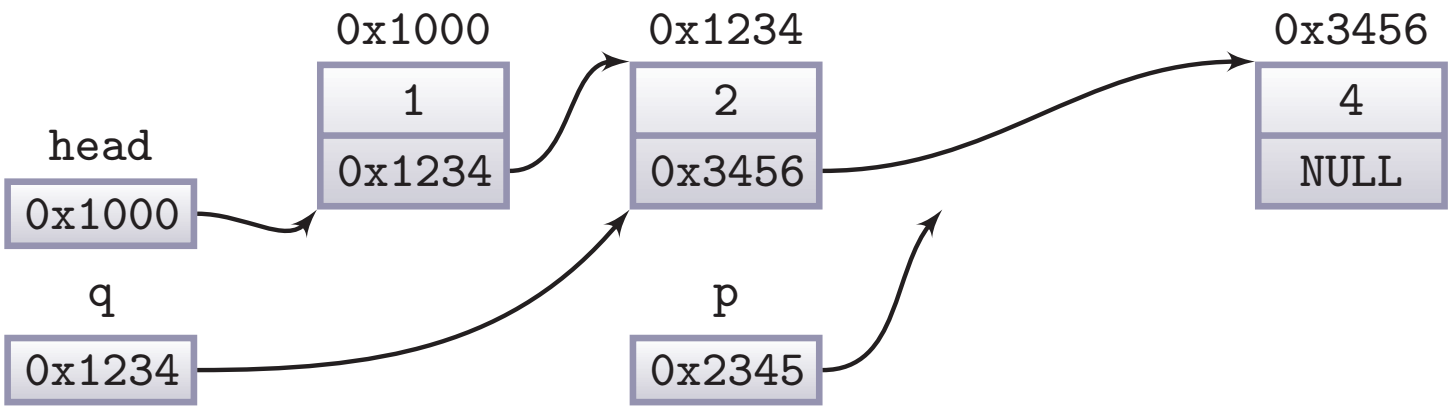


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```

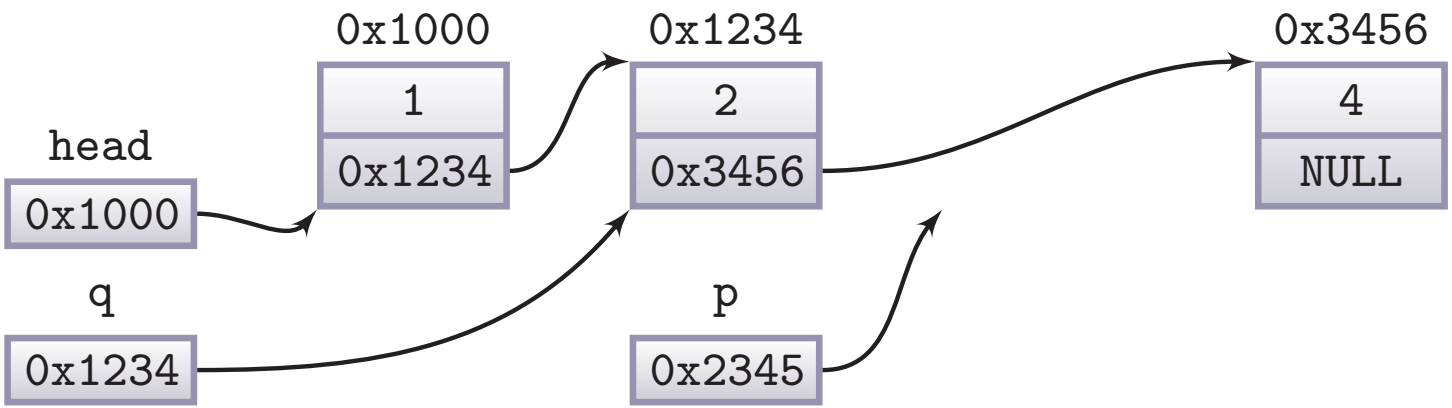


Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Deleting the `data = 3` element

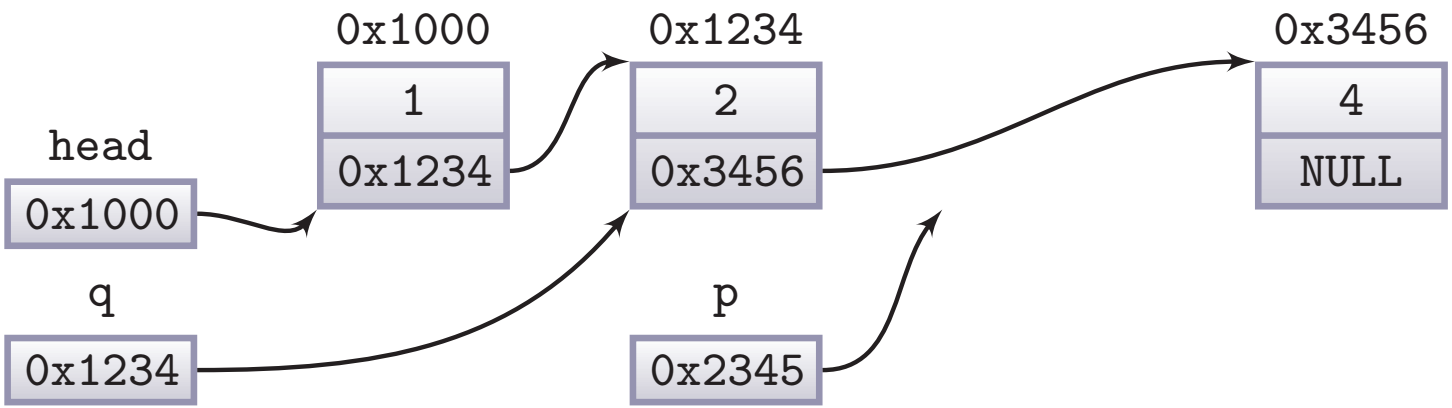
```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```



Deleting a given element from list

- Deleting the `data = 3` element

```
1  q = head; p = head->next;
2  while (p != NULL && p->data != data) {
3    q = p; p = p->next;
4  }
5  if (p != NULL) { /* now we have it */
6    q->next = p->next;
7    free(p);
8  }
```



- If the list is empty, or we have to delete the first element, this does not work

Deleting a given element from list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  listelem *delete_elem(listelem *head, int d)
2  {
3    listelem *p = head;
4
5    if (head == NULL) return head;
6
7    if (head->data == d) return pop_front(head);
8
9    while (p->next != NULL && p->next->data != d)
10     p = p->next;
11   if (p->next != NULL)
12   {
13     listelem *q = p->next;
14     p->next = q->next;
15     free(q);
16   }
17   return head;
18 }
```

link

Deleting an entire list

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void dispose_list(listelem *head)
2  {
3    while (head != NULL)
4      head = pop_front(head);
5  }
```

link

Summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We have everything we need, but it was really cumbersome, because

Summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We have everything we need, but it was really cumbersome, because
  - we can insert element only after (behind) an element

Summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We have everything we need, but it was really cumbersome, because
  - we can insert element only after (behind) an element
  - we can delete only an element behind another element

Summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We have everything we need, but it was really cumbersome, because
  - we can insert element only after (behind) an element
  - we can delete only an element behind another element
  - empty lists and lists with only one element must be handled separately when inserting or deleting

Thank you for your attention.