AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

# 23CSE203

# DATA STRUCTURES & ALGORITHMS

Refresher of Data Structures - Abstract Data Types and Data Structures - Principles, and Patterns. Basic complexity analysis – Best, Worst, and Average Cases - Asymptotic Analysis -Analyzing Programs – Space Bounds, recursion- linear, binary, and multiple recursions. -Sorting and Selection – Linear Sorting –Divide and Conquer based sorting – Analysis using Recurrence Tree based Method - Merge Sort - Quick Sort - Studying Sorting through an Algorithmic Lens. Arrays, Linked Lists and Recursion: Using Arrays - Lists - Array based List Implementation – Linked Lists – LL ADT – Singly Linked List – Doubly Linked List – **Circular Linked List** Stacks and Queues: Stack ADT - Array based Stacks, Linked Stacks – Implementing Recursion using Stacks, Stack Applications. Queues - ADT, Array based Queue, Linked Queue, Double-ended queue, Circular queue, applications.

## Course Outcome:

| COs | Course Outcome Description | BTL |
|---|---|---|
| CO1 | Understand the concept and functionalities of Data Structures and be able to implement them efficiently | L3 |

J.UMA, AP-CSE

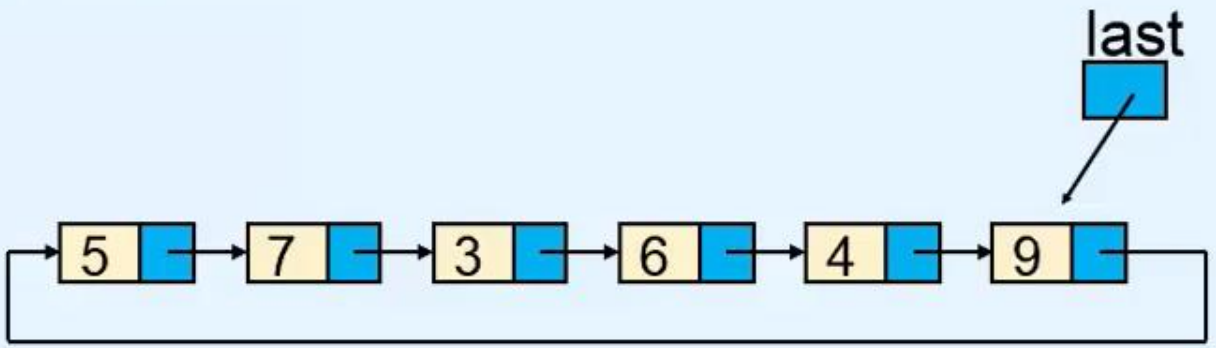
Amrita School of Computing

# Circular Linked List in DS using Python

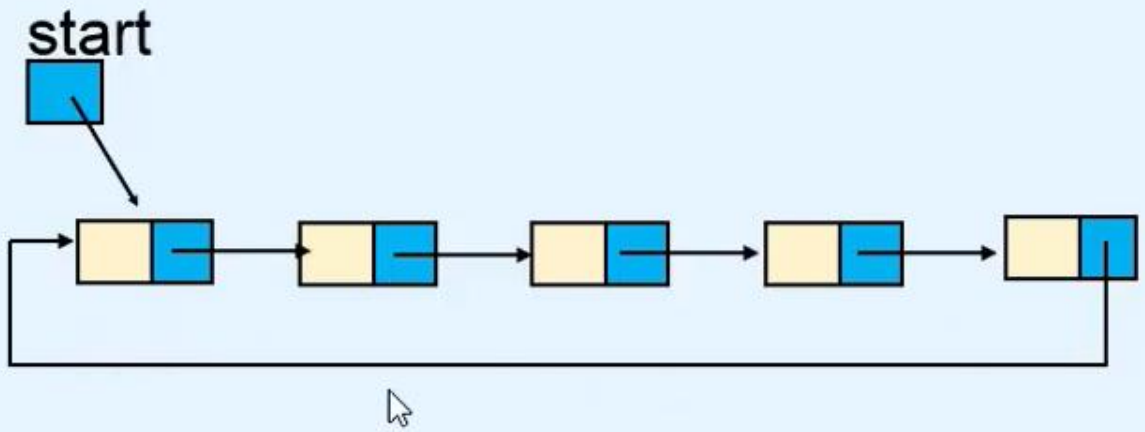
Dr.J.Uma

AP-CSE

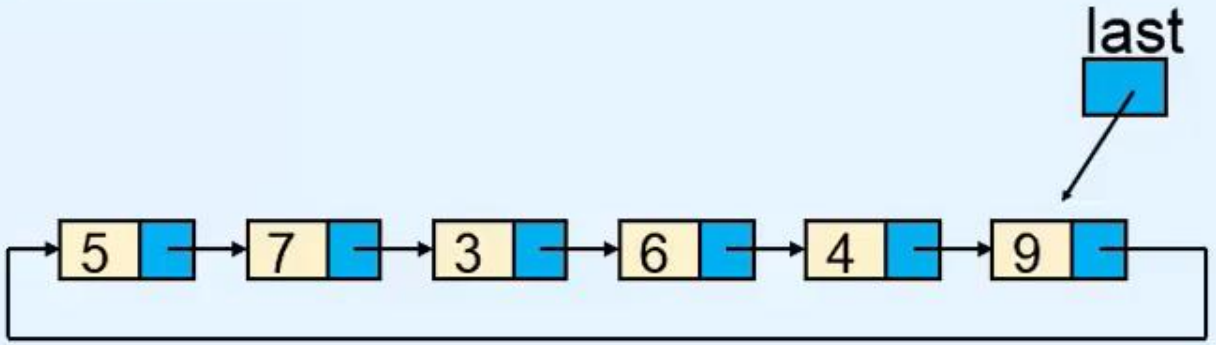
# Circular linked list




last.link Refers to first node of list



# Circular linked list

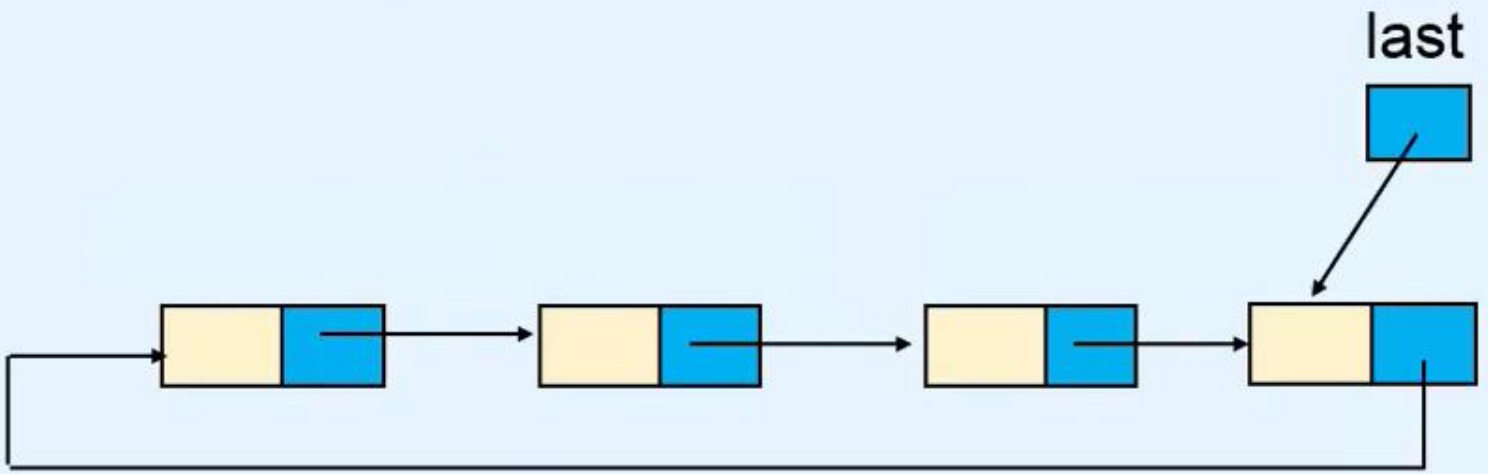



last.link Refers to first node of list

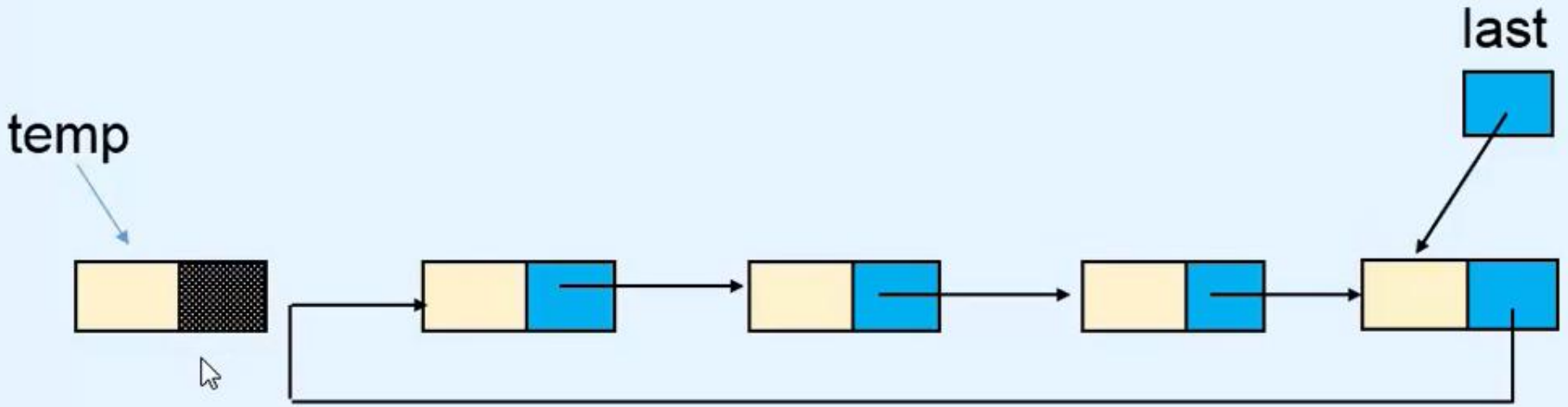
## Implementation of Queue

```
class Node(object):

    def __init__(self,value):
        self.info = value
        self.link = None


class CircularLinkedList(object):

    def __init__(self):
        self.last = None
```

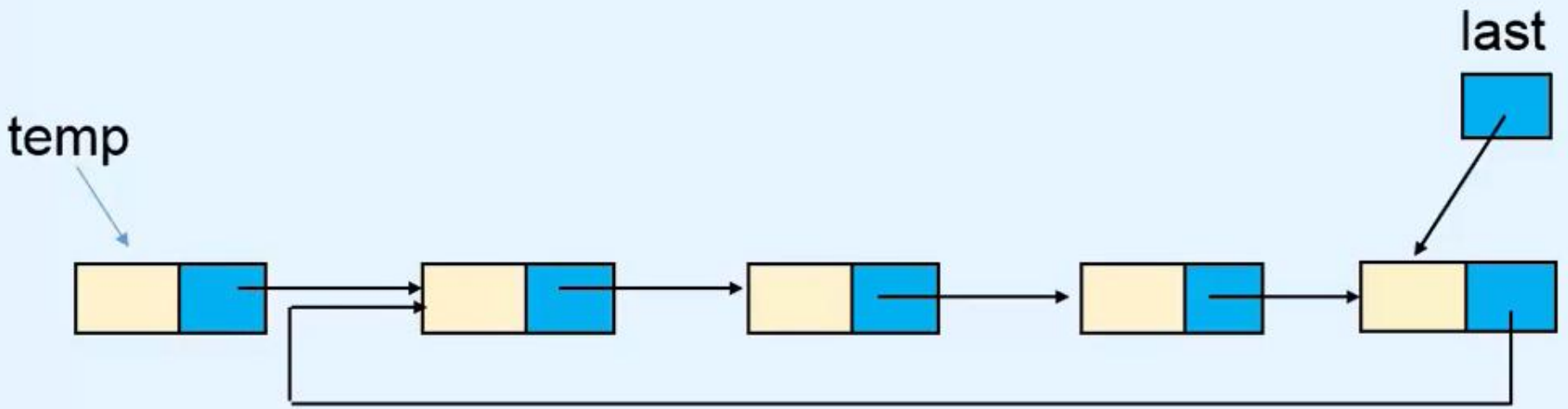
# Insertion in the beginning of the list



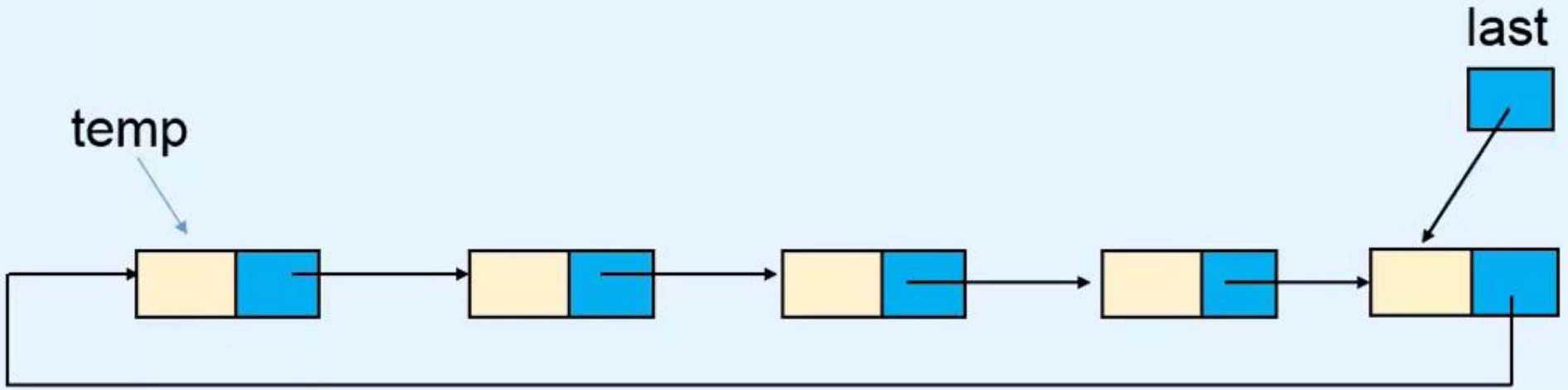
# Insertion in the beginning of the list



# Insertion in the beginning of the list



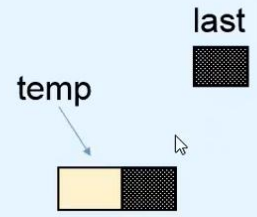
# Insertion in the beginning of the list




```
temp.link = self.last.link

self.last.link = temp
```

```
def insert_in_beginning(self,data):
    temp = Node(data)
    temp.link = self.last.link
    self.last.link = temp
```

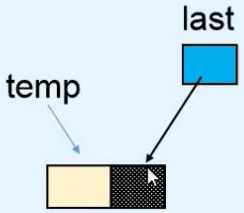
## Insertion in an empty list




1

## Insertion in an empty list

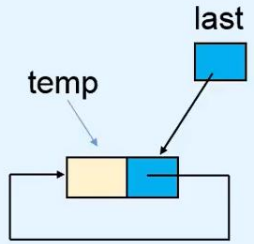
2




```
def insert_in_empty_list(self,data):
    temp = Node(data)
    self.last = temp
    self.last.link = self.last
```

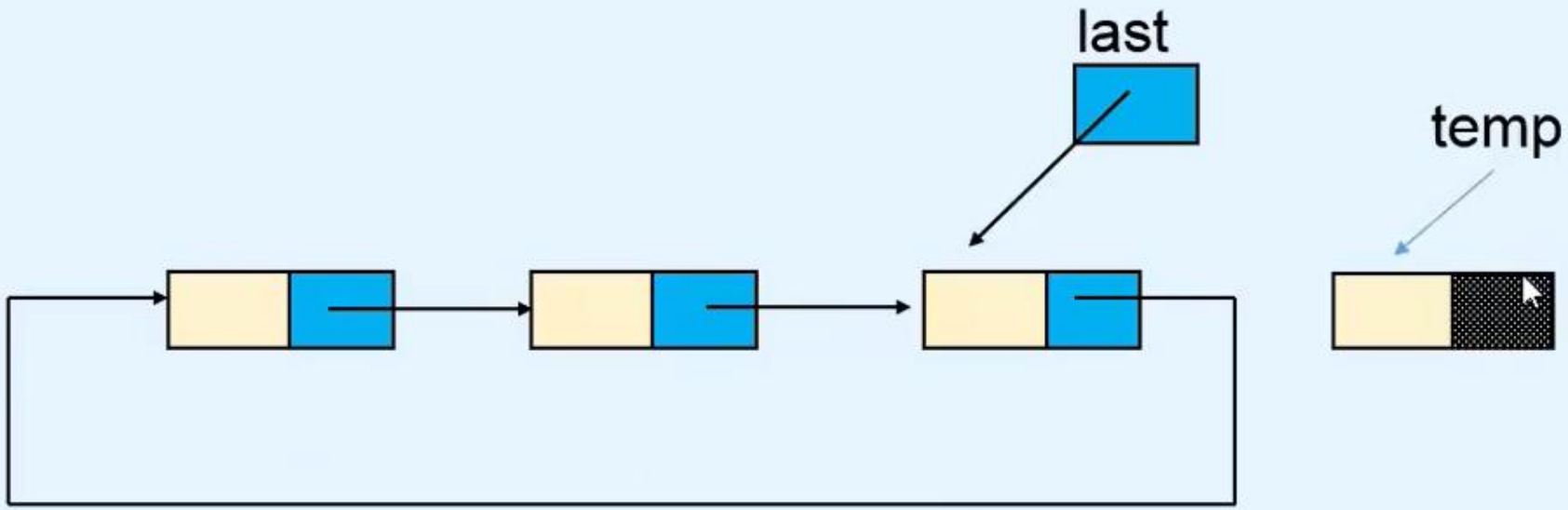
## Insertion in an empty list

3

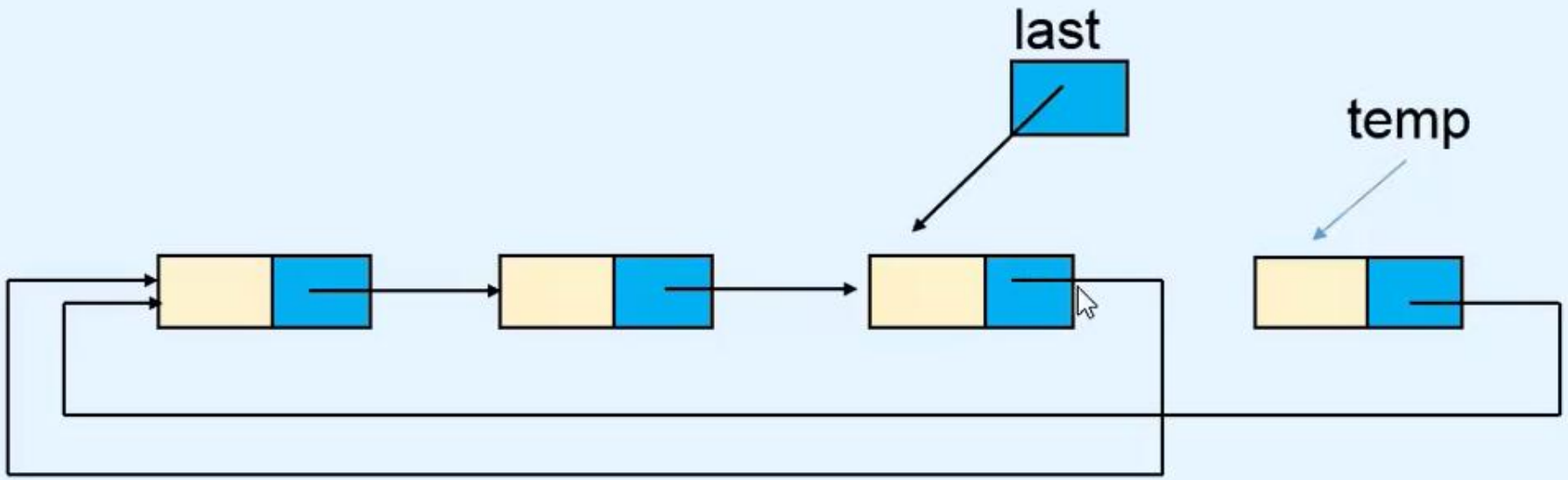



```
self.last = temp

self.last.link = self.last
```

# Insertion at the end of the list

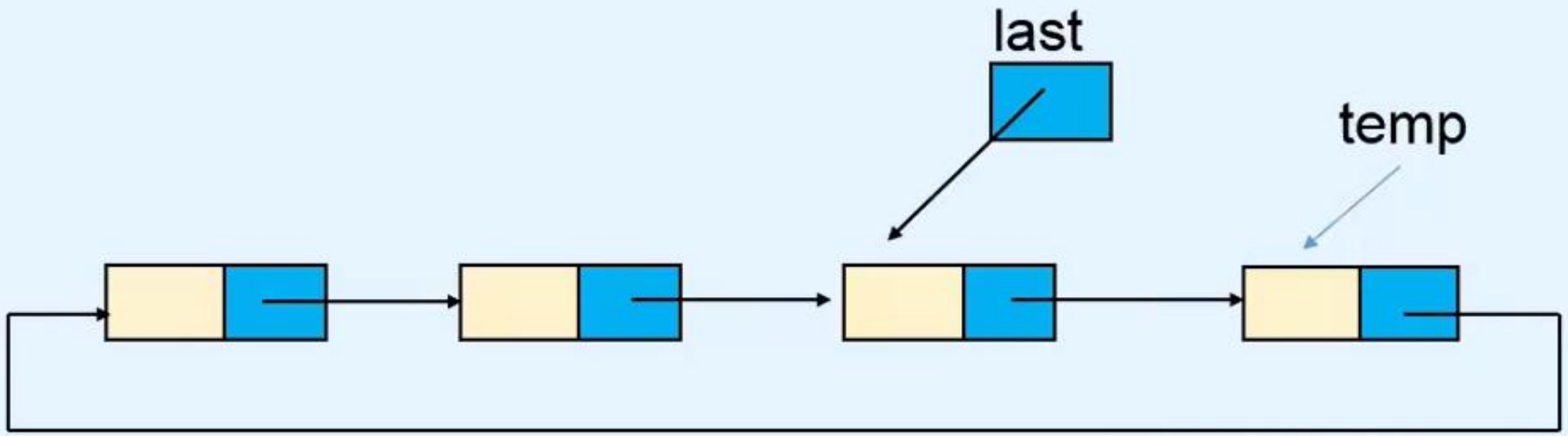


# Insertion at the end of the list




```
temp.link = self.last.link
```

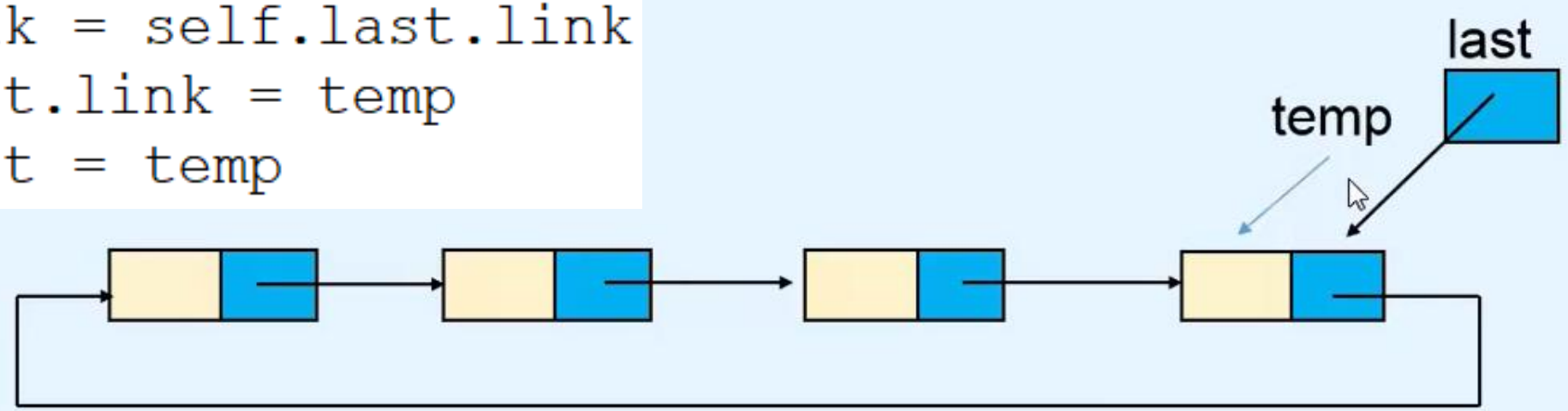
# Insertion at the end of the list




```
temp.link = self.last.link

self.last.link = temp
```

# Insertion at the end of the list

```
def insert_at_end(self,data):
    temp = Node(data)
    temp.link = self.last.link
    self.last.link = temp
    self.last = temp
```

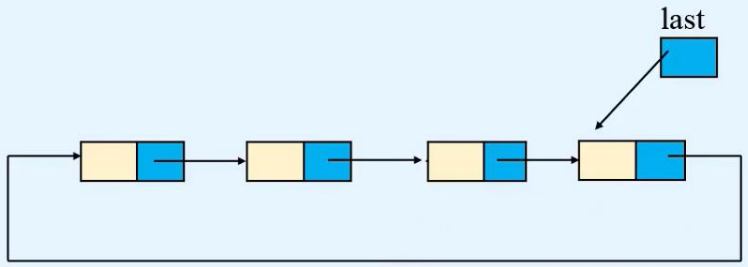



```
temp.link = last.link

last.link = temp

last = temp
```

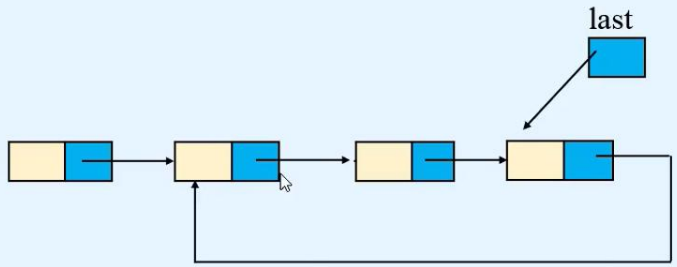
```
def insert_after(self,data,x):
    p = self.last.link

    while True:
        if p.info == x:
            break
        p = p.link
        if p == self.last.link:
            break

    if p == self.last.link and p.info != x:
        print(x , " not present in the list")
    else:
        temp = Node(data)
        temp.link = p.link
        p.link = temp
        if p == self.last:
            self.last = temp
```

# Deletion in circular linked list

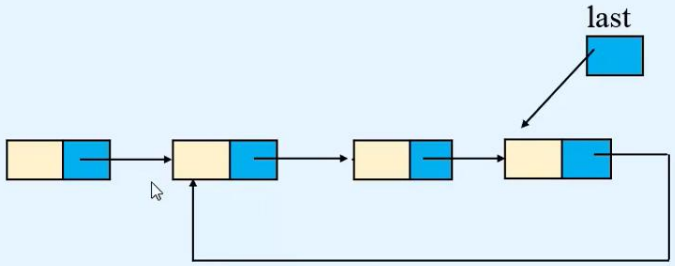
## Deletion of the first node




## Deletion of the first node



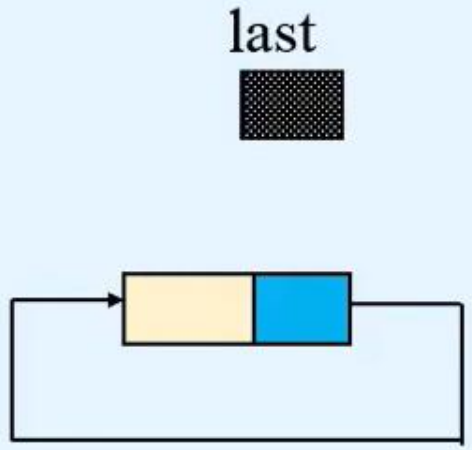

## Deletion of the first node




`self.last.link = self.last.link.link`

```
def delete_first_node(self):
    if self.last is None: # List is empty
        return
    if self.last.link == self.last: # List has only one node
        self.last = None
        return
    self.last.link = self.last.link.link
```

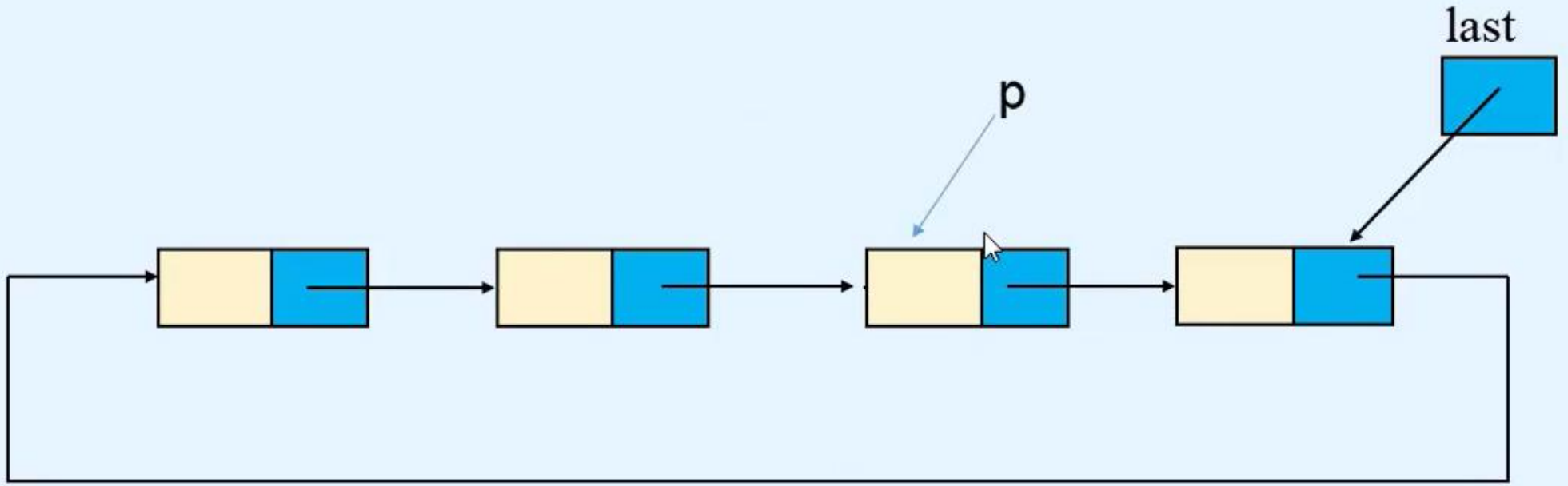
# Deletion of the only node



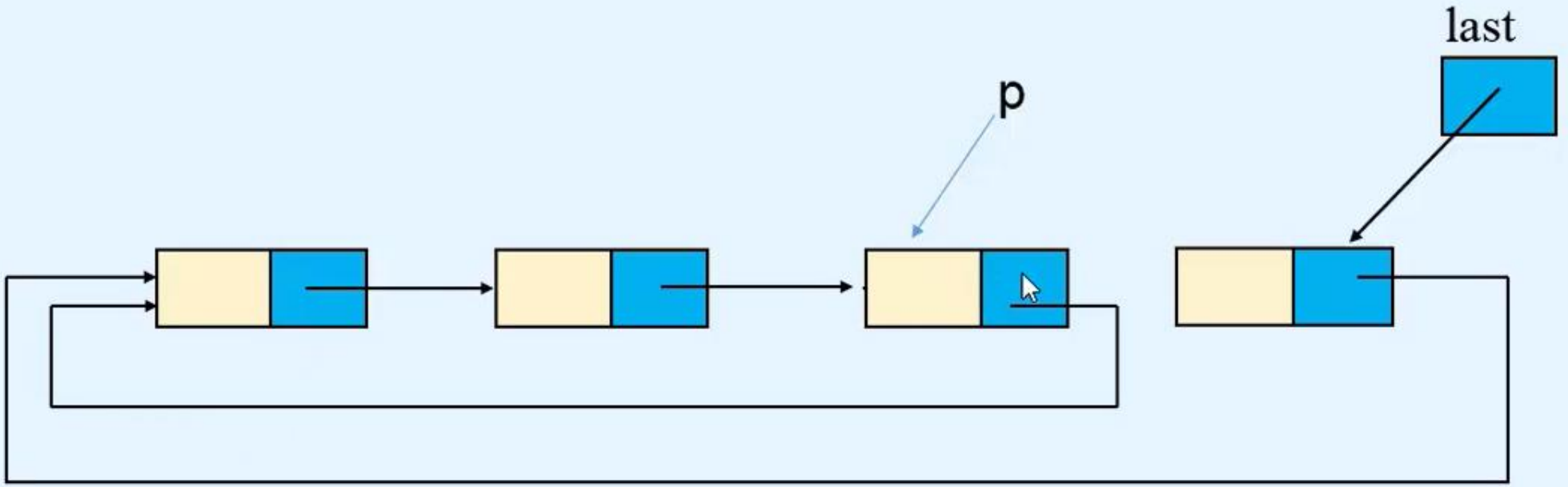

**self.last = None**

```
def delete_node(self, x):

    if self.last is None: # List is empty
            return
    if self.last.link == self.last and self.last.info == x:
       # Deletion of only node
        self.last = None
        return
```

# Deletion at the end of the list

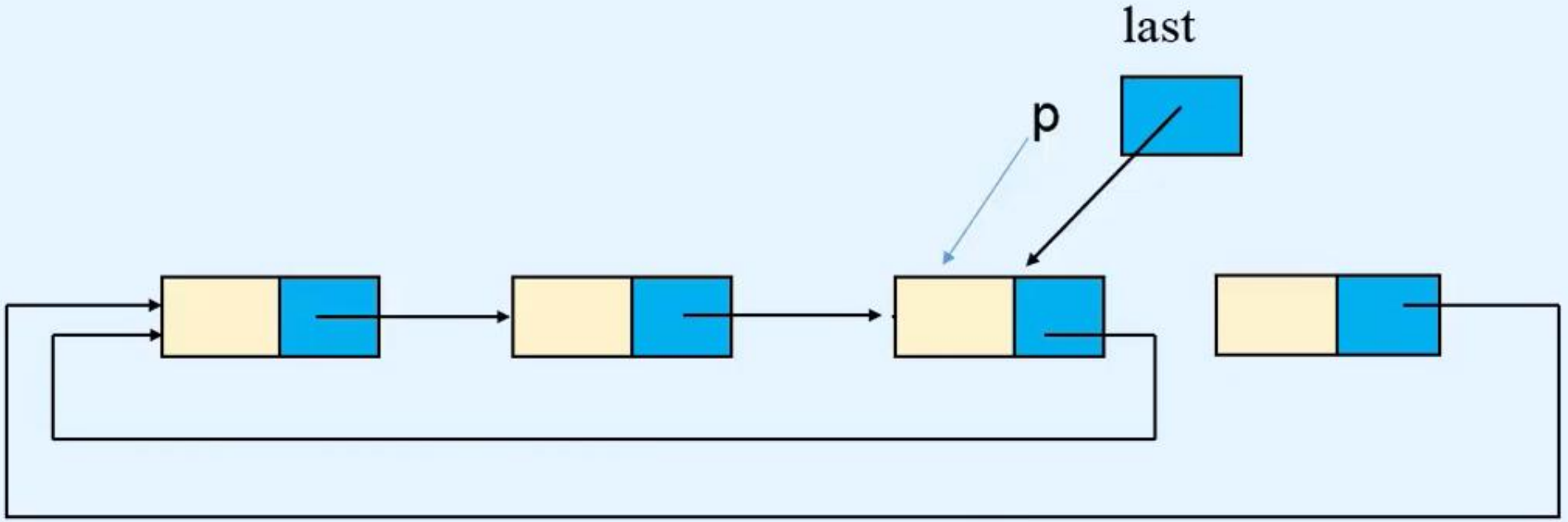


# Deletion at the end of the list



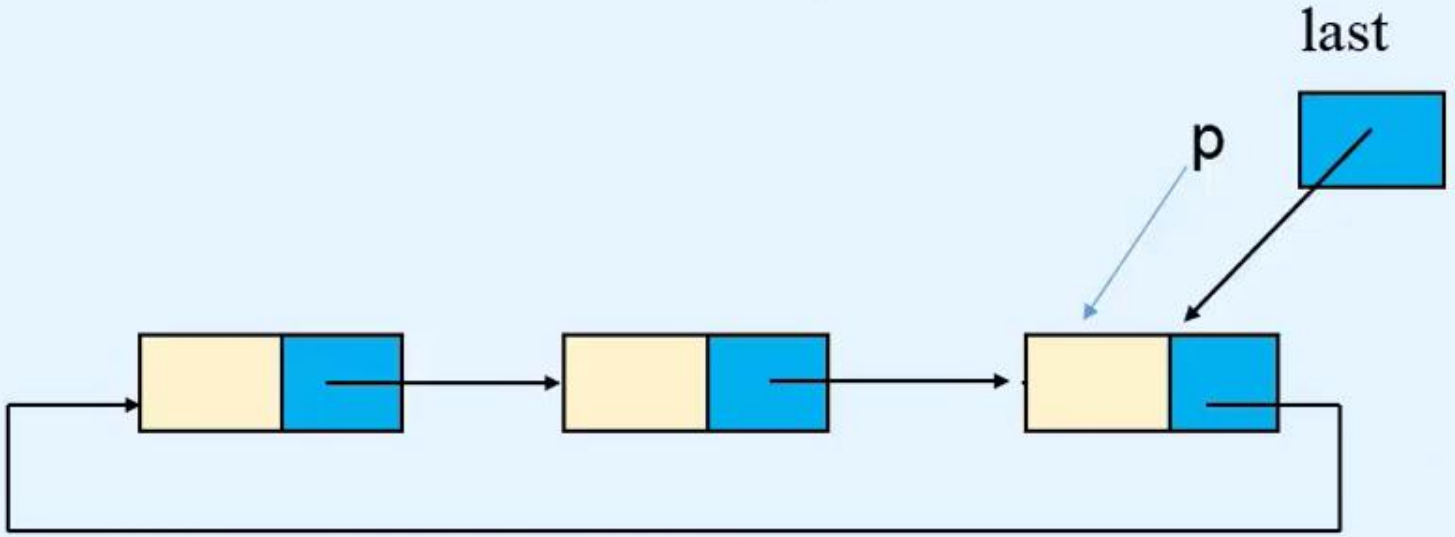

```
p.link = self.last.link
```

# Deletion at the end of the list




```
p.link = self.last.link

self.last = p
```

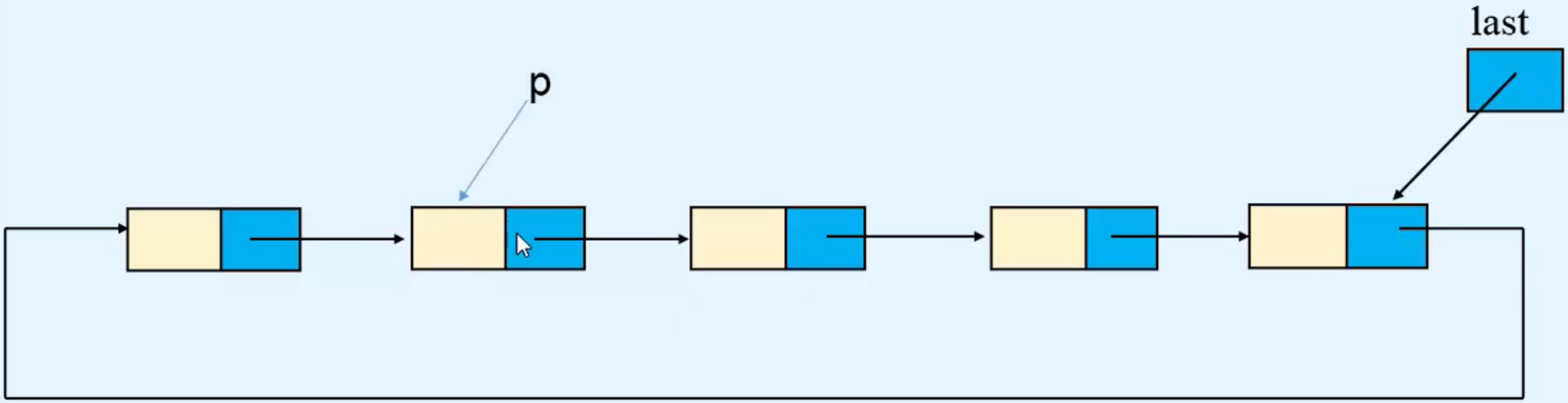
# Deletion at the end of the list



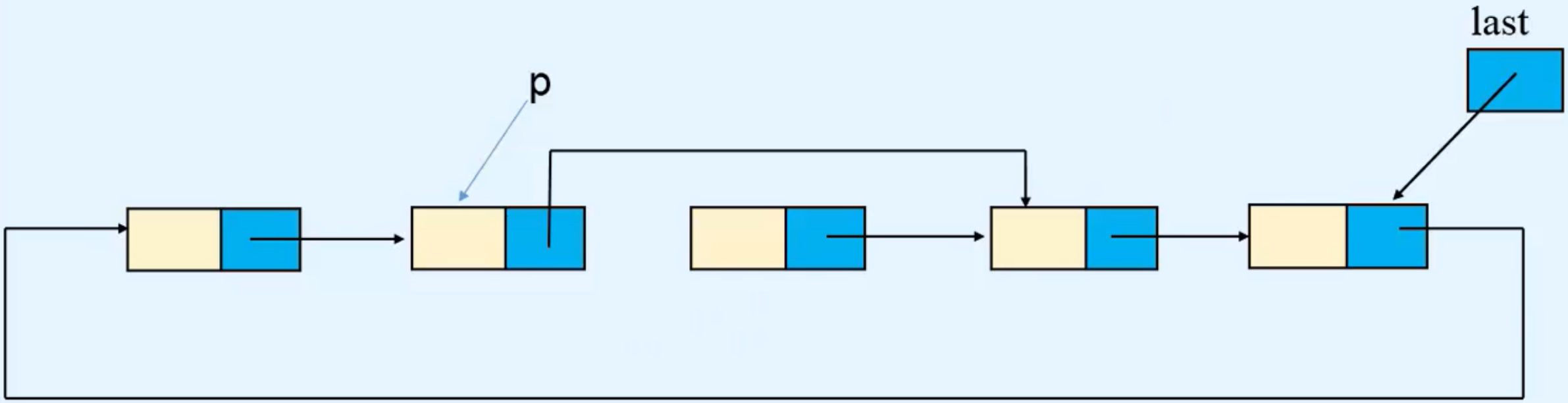

```
p.link = self.last.link

self.last = p
```

```
def delete_last_node(self):
    if self.last is None: # List is empty
        return
    if self.last.link == self.last: # List has only one node
        self.last = None
        return

    p = self.last.link
    while p.link != self.last:
        p = p.link
    p.link = self.last.link
    self.last = p
```

# Deletion in between the list

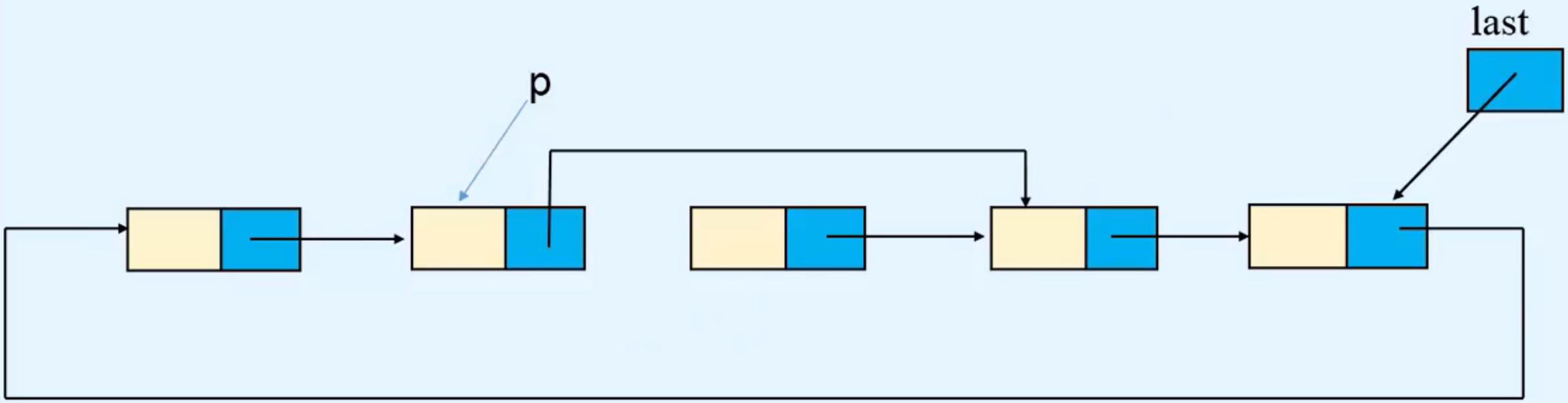


# Deletion in between the list



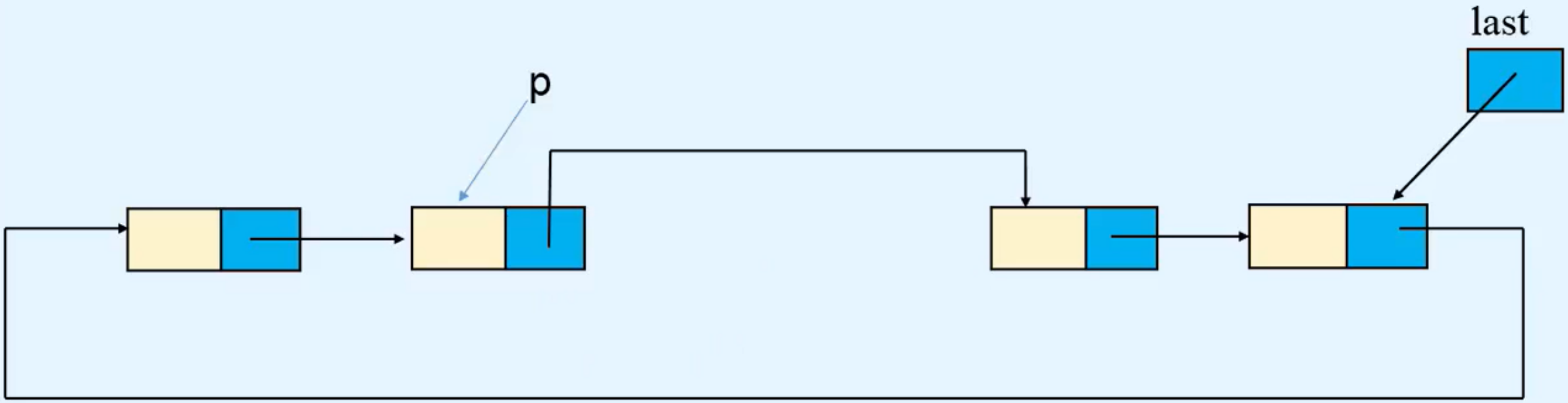

```
p.link = p.link.link
```

# Deletion in between the list




```
p.link = p.link.link
```

# Deletion in between the list




```
p.link = p.link.link
```

1 3 5

2 4 ?

This may not be a number.
Think of a situation in everyday life where these numbers appear.

*Let's give you a hint...*

*It's a lever in the car with numbers on it.*

*You guessed the answer now, right?*

*You guessed it, right?*

*Exactly!* ***It’s a gear stick****.*

*So the question mark will be replaced by the letter ‘****R****’.*