Data Structures & Algorithms I

Roshani Wijesuriya

Session 01

Outline of the syllabus

- Why Data Structures and Algorithms?
- Data Structures
  - Array, Linked lists, Stacks, Queues
- Sorting Algorithms
  - Insertion, Bubble & Selection sort
- Recursion
- Applications of Data Structures and Algorithms
- Algorithm Complexity

Course logistics and Details

- Final Examination (70%)
- Continuous Assessments (30%)

What are Data Structures?

Data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

Characteristics of a Data Structure

- **Correctness** − Data structure implementation should implement its interface correctly.
- **Time Complexity** − Running time or the execution time of operations of data structure must be as small as possible.
- **Space Complexity** − Memory usage of a data structure operation should be as little as possible.

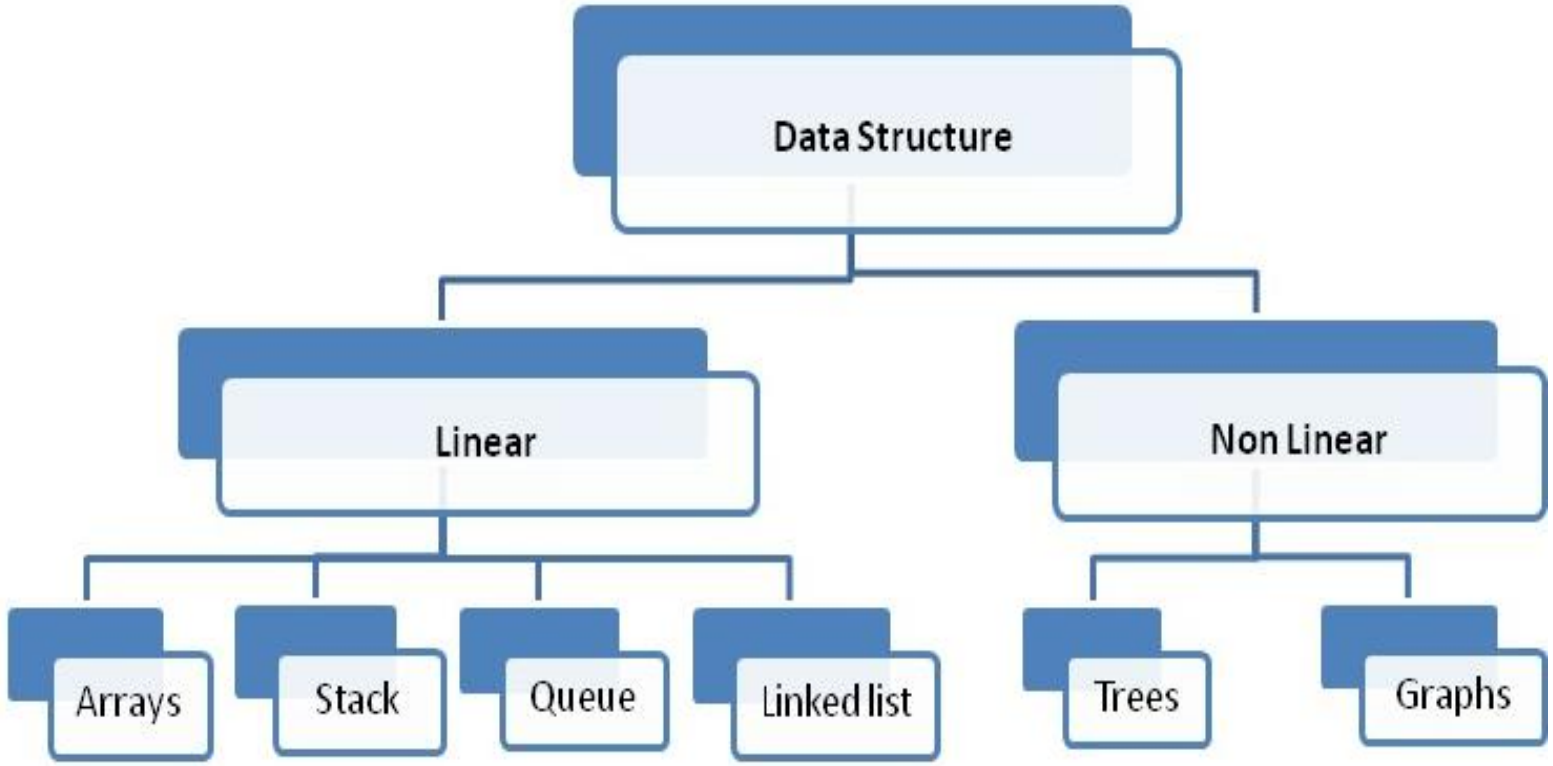
Why to learn Data Structure and Algorithms?

- **Data search** - Consider an inventory of 1 million(10^6) items of a store. If the application is to search an item, it has to search an item in 1 million(10^6) items every time slowing down the search. As data grows, search will become slower.

- **Processor Speed** - Processor speed although being very high, falls limited if the data grows to billion records.

- **Multiple requests** - As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

Data Structure



Content

- What is Array?
- Types of Arrays.
- Array Operations.

Array

Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

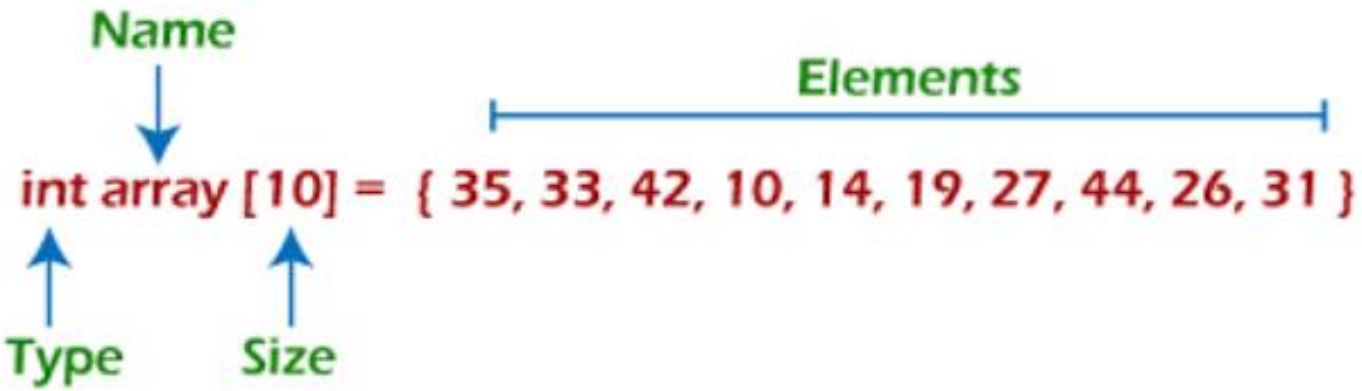
Types of Arrays

- One dimensional array
- Two-dimensional array
- Multi dimensional array

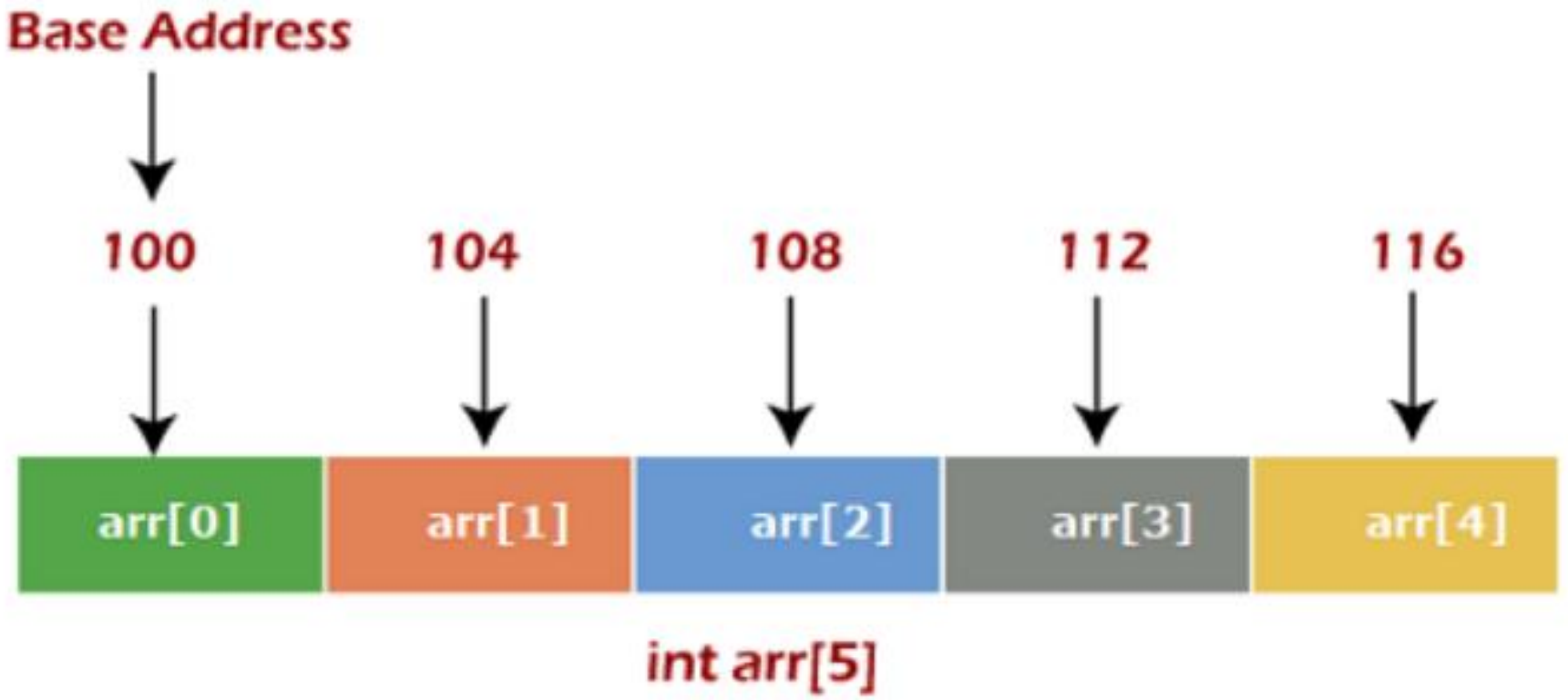
One dimensional array

- The one-dimensional array stores the data elements in a single row or column.
- Syntax:

<data type> <array name>[size] ={values};



Memory allocation of an array



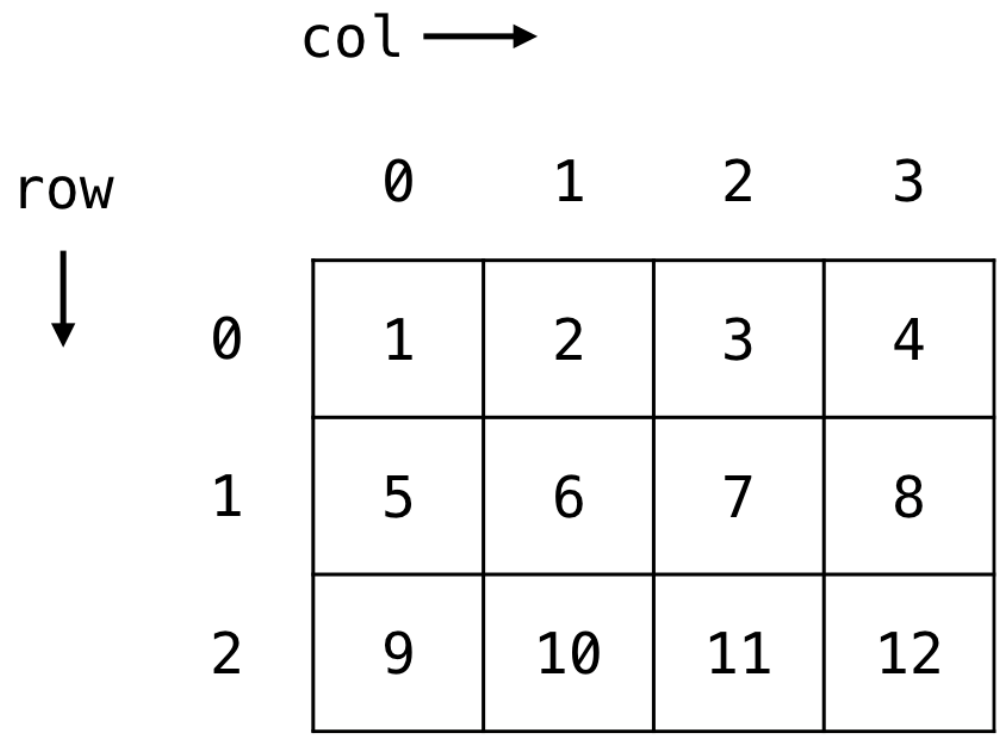
Two-dimensional array

- A two-dimensional array is a collection of elements placed in rows and columns.
- Syntax:

  <data type> <array name> [row size][column size];

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

```
int num[3][4] = {
  {1, 2,  3,  4},
  {5, 6,  7,  8},
  {9, 10, 11, 12}
};
```



Multi dimensional arrays

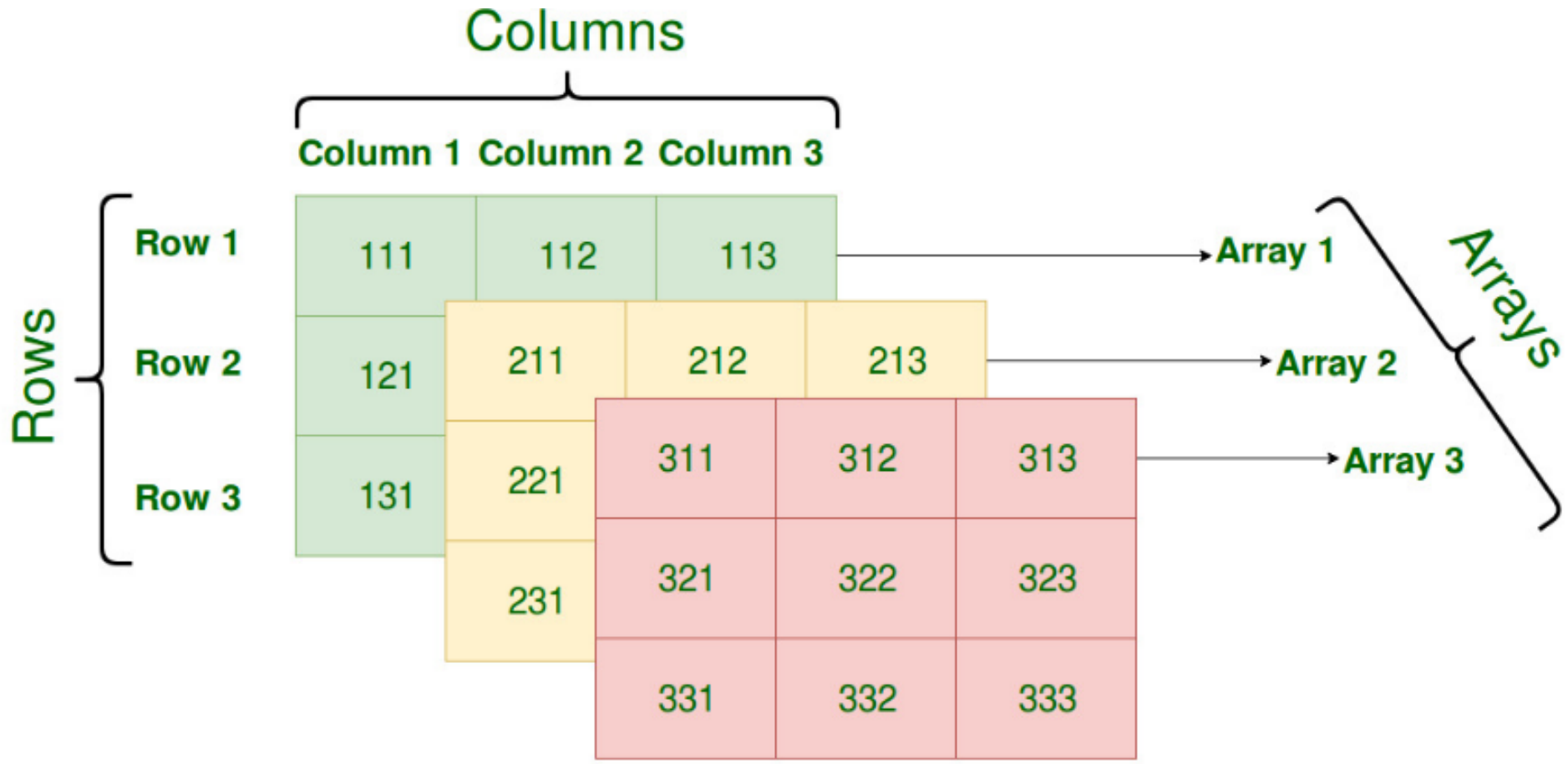
- An array haves 2 or more subscripts, that type of array is called multi dimensional array.
- Syntax for 3D array

<data type> <array name> [s1][s2][s3] = {values};

ex: int arr[3][3][3];

- **int** shows that the 3D array is an array of **type integer**.
- **arr** is the **name of array**.
- **first dimension** represents the **block size**(total number of 2D arrays).
- **second dimension** represents the **rows of 2D arrays**.
- **third dimension** represents the **columns of 2D arrays**.

Three-dimensional array

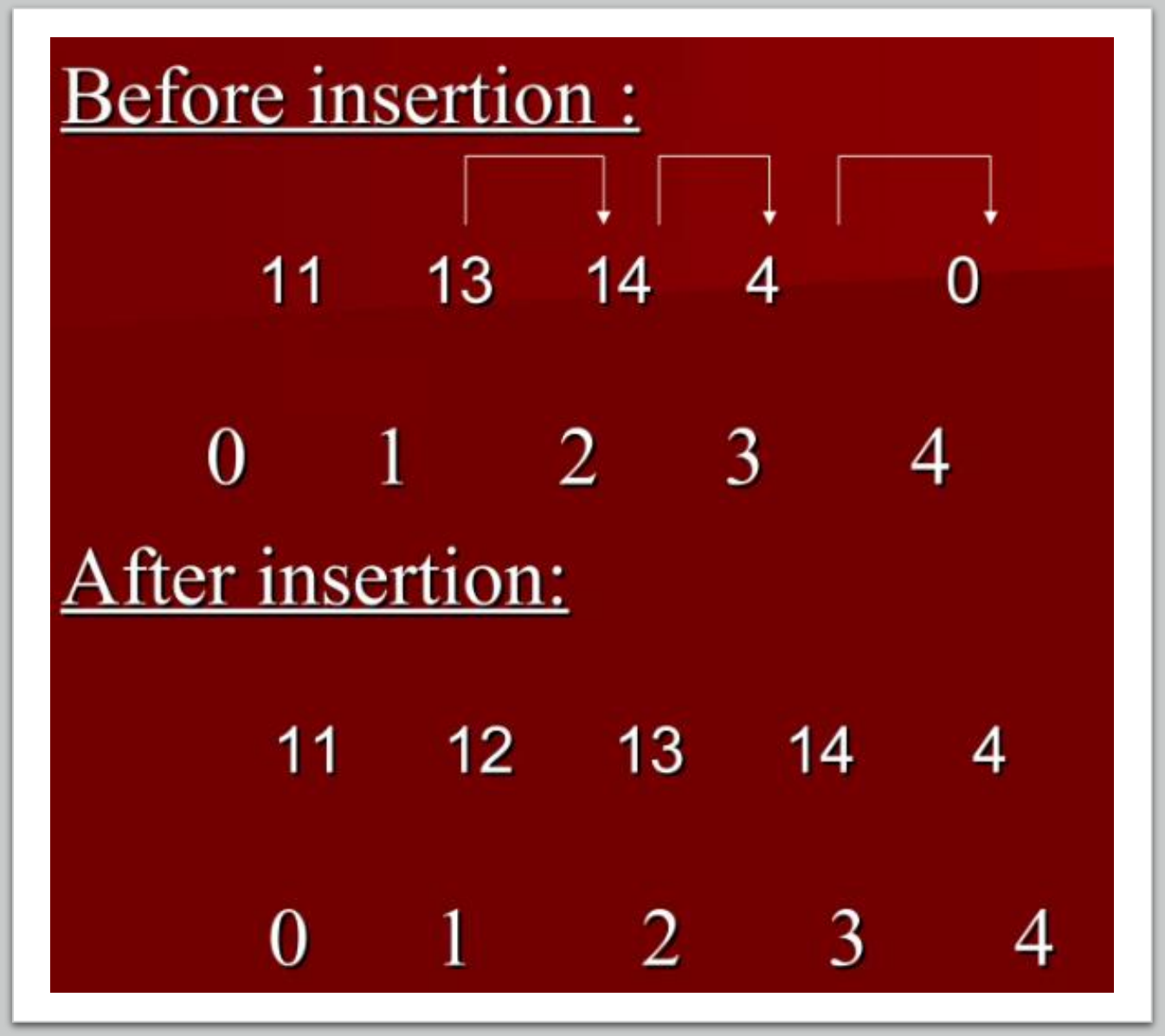


Basic operations

- Insertion - It is used to add an element at a particular index.
- Deletion - It is used to delete an element from a particular index.
- Search - It is used to search an element using the given index or by the value.
- Traversing: It prints all the array elements one after another.

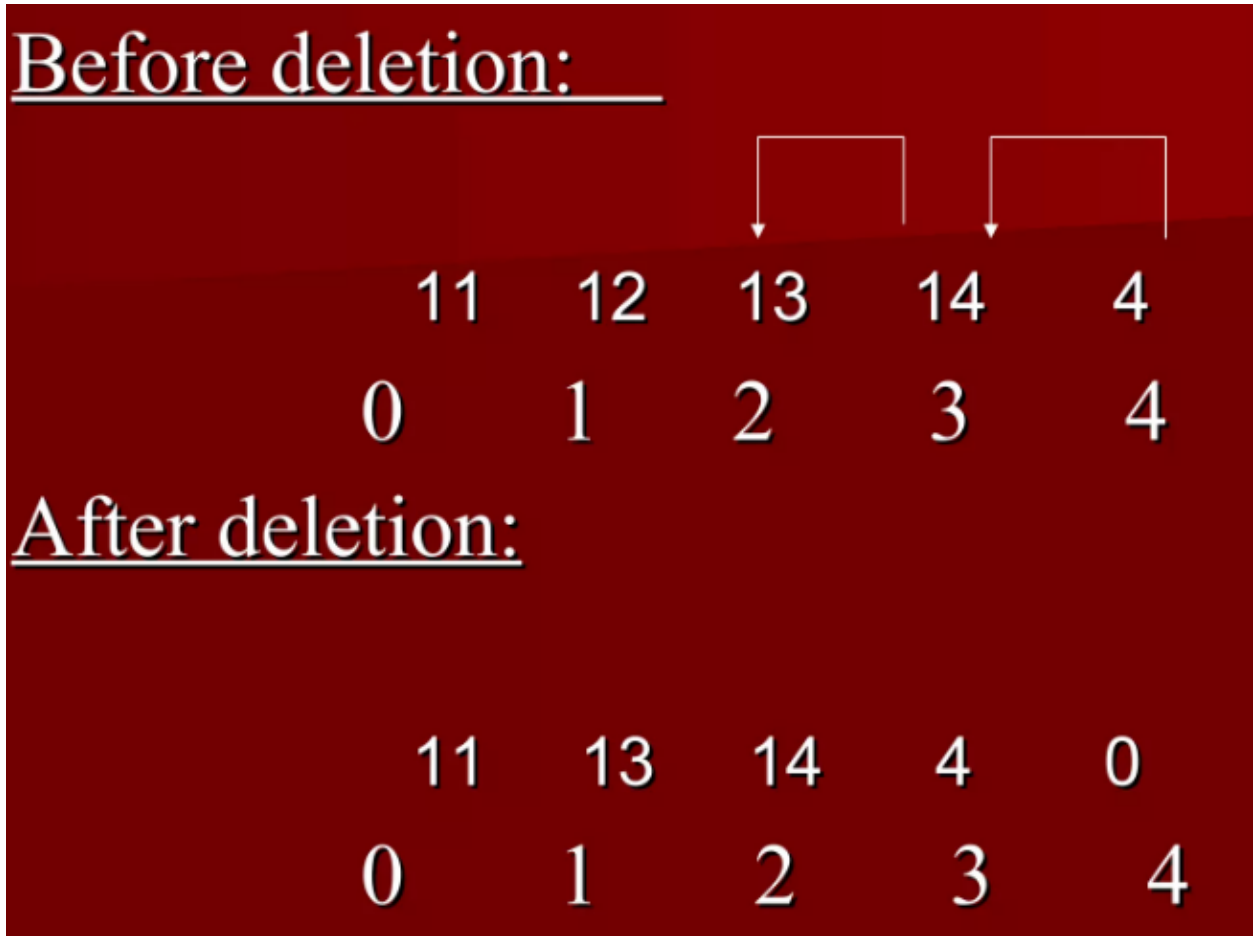
Insertion

- Insertion is adding a new element to an array.
- We have shifted the numbers, from the specified position, one place to the right of their existing position.
- Then we have placed the new number at the vacant place.



Deletion

- Deletion is process of remove an element from the array.
- We have shifted the numbers of placed after the position from where the number is to be deleted, one place to the left of their existing positions.



Arrays of pointers

- A pointer variable always contains an address.
- An array of pointer would be nothing but a collection of addresses.
- The address present in an array of pointer can be address of isolated variables or even the address of other variables.

Linked lists

Content

- Introduction to Linked List
- Insertion
- Deletion
- Types of Linked List
- Linked List vs Array

Need of Linked List

int x = 3;

000 001 002 ……

012

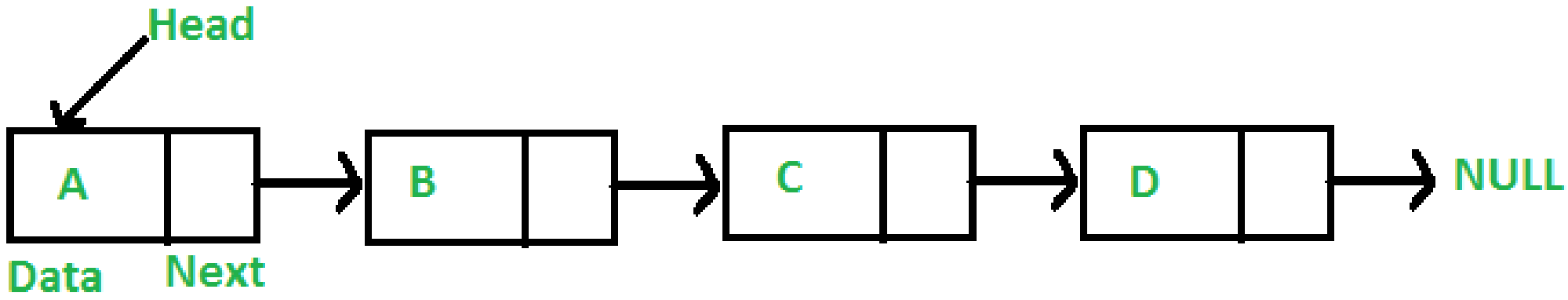
int y[3];

000 001 002 ……

012

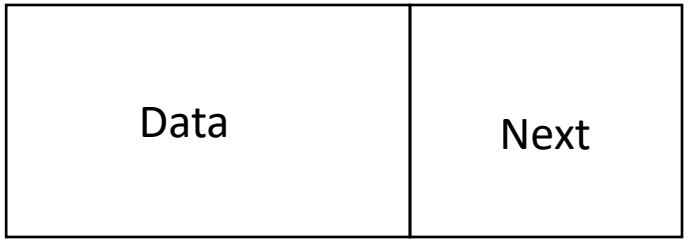
What is a Linked List?

- Like arrays, Linked List is a linear data structure.
- Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.



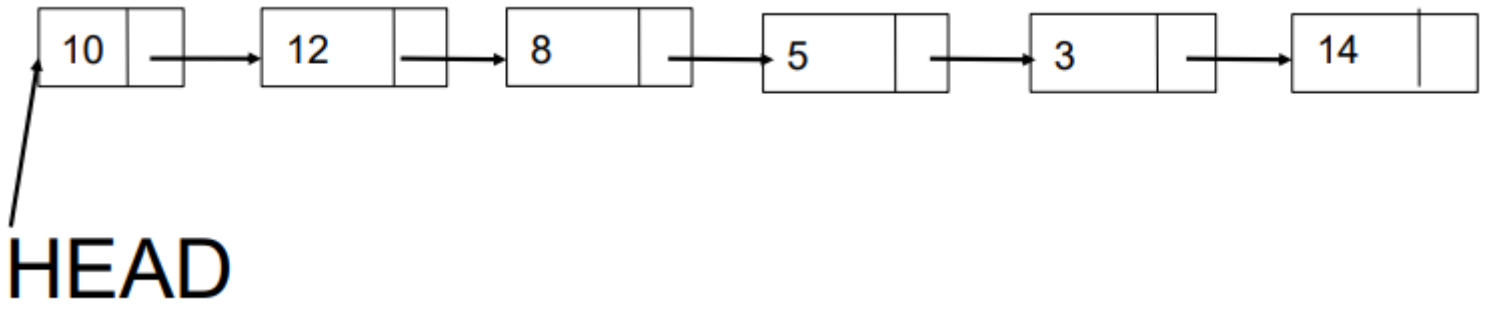
What is a Linked List?

- Each node in the list has some data and then also indicates via a **pointer** the location of the next node.
- Some languages call the pointer also as a **reference**.



How to access a Linked List?

- Via a pointer to the first node, normally called the **head**.



Node

- Each data item is embedded in a node.
- Each node has
  - An item
  - A reference to next node in the list.

```
public  class Node {
        public  int item ; // Data
        public  Node next ; // reference to next node
}
```

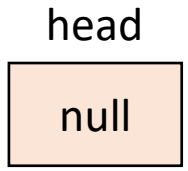
```
struct node
{
int data;
struct node *next;
}
```

Reference

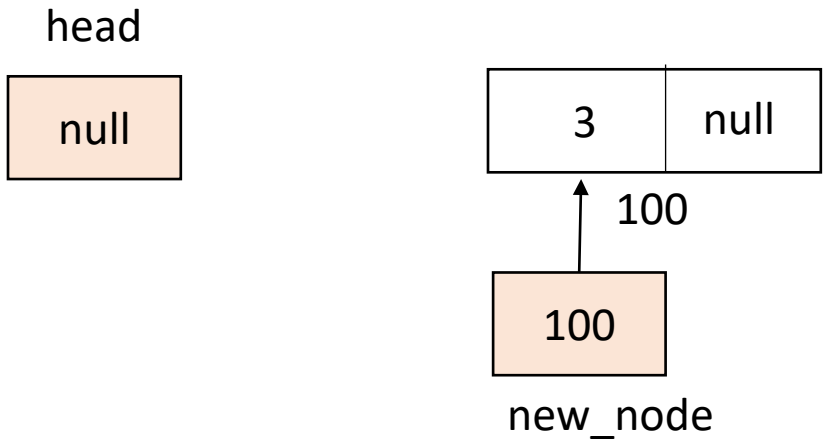
- A reference is a number that refers to an object.
- It is the object's address in the computer's memory, but you don't need to know its value. You just treat it as a number that tells you where the object is.

Create a Linked List

- Create a head pointer

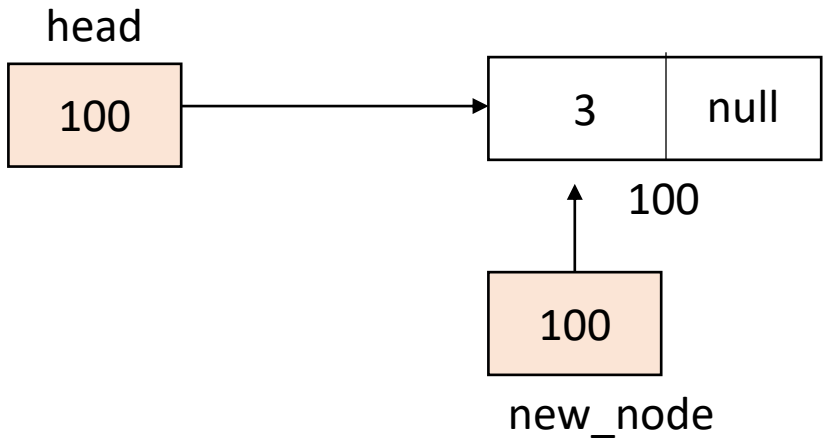


- Create a new node



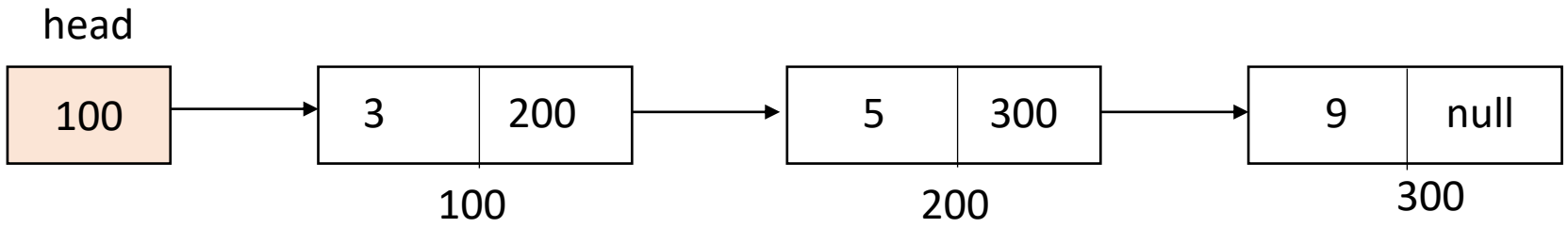
Create a Link List

- Change head pointer value to new node's memory address



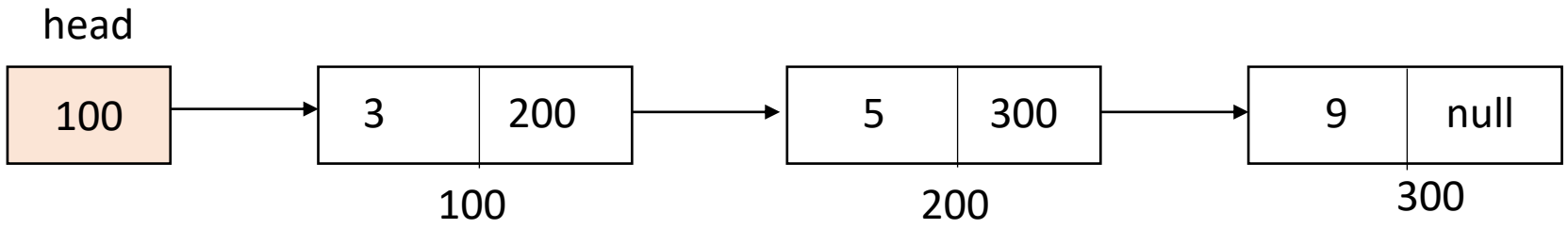
Searching

- Traverse



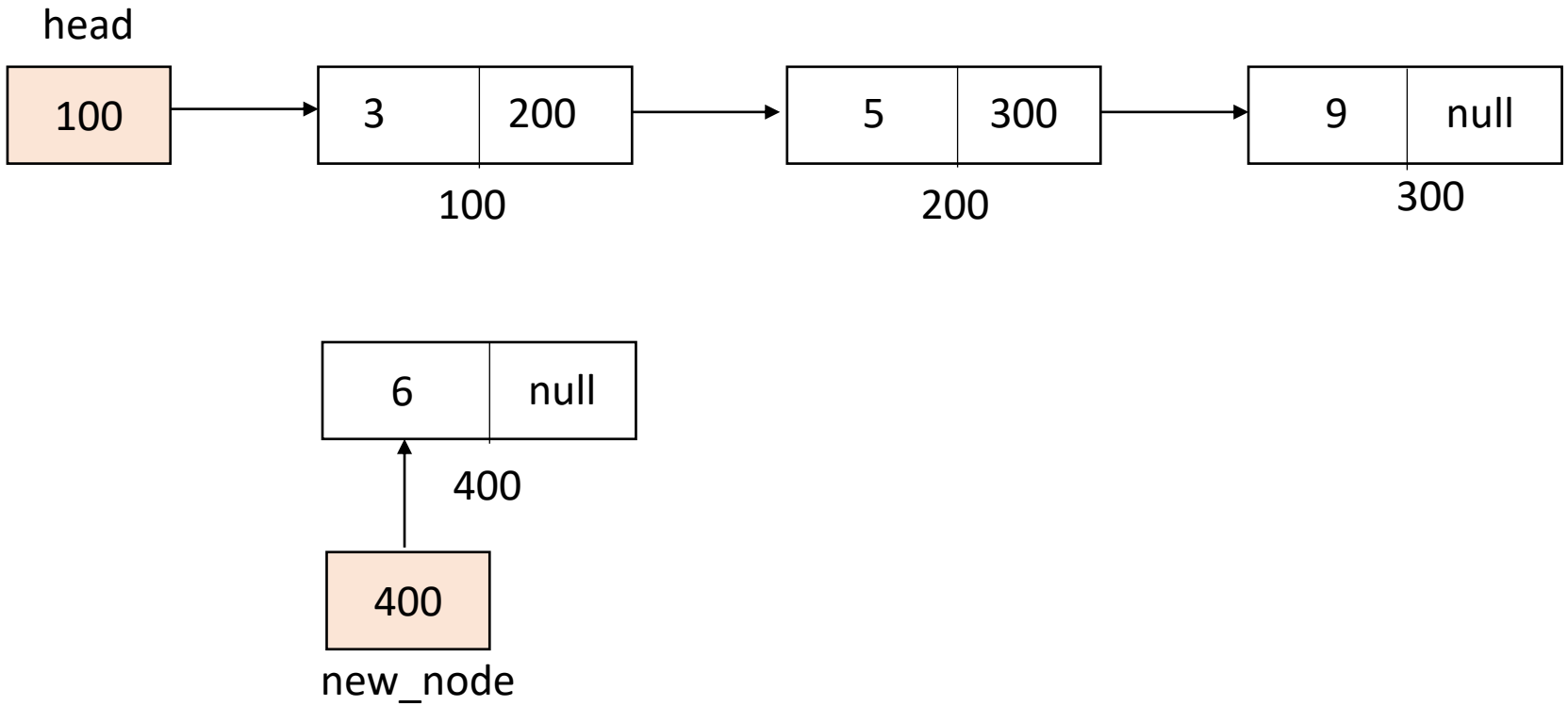
Insertion (at the beginning)

- Add a new node at the beginning



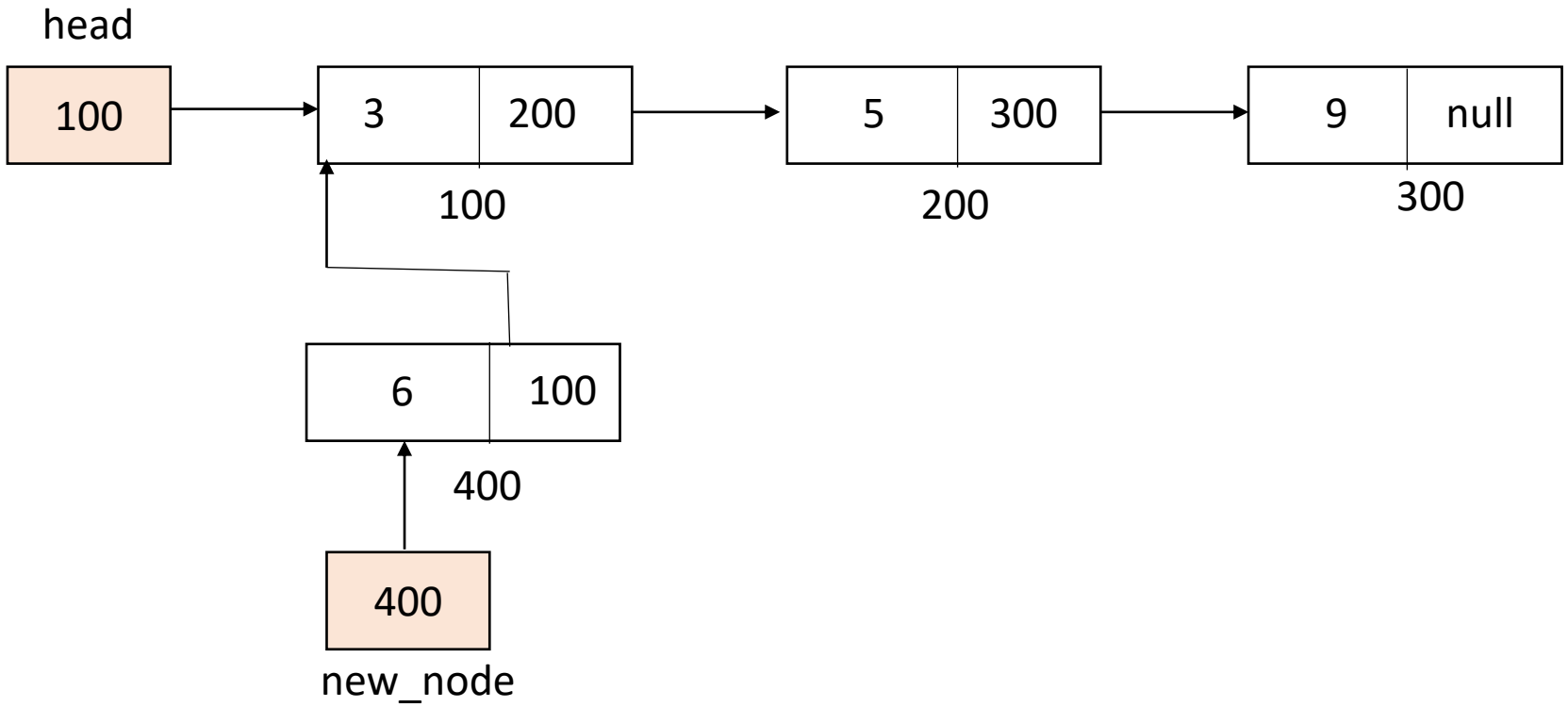
Insertion (at the beginning)

- Create a new node



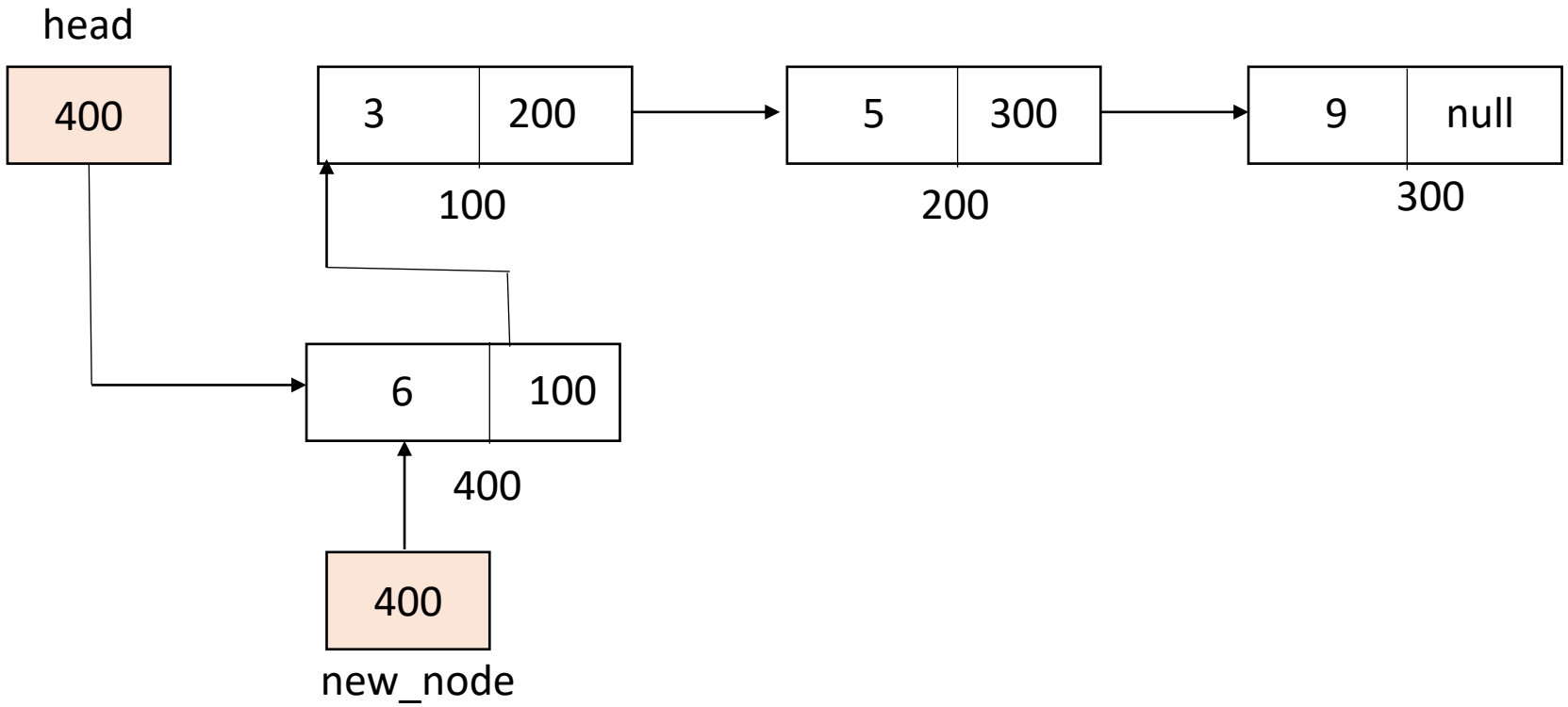
Insertion (at the beginning)

- Assign first node's address into new node's next link



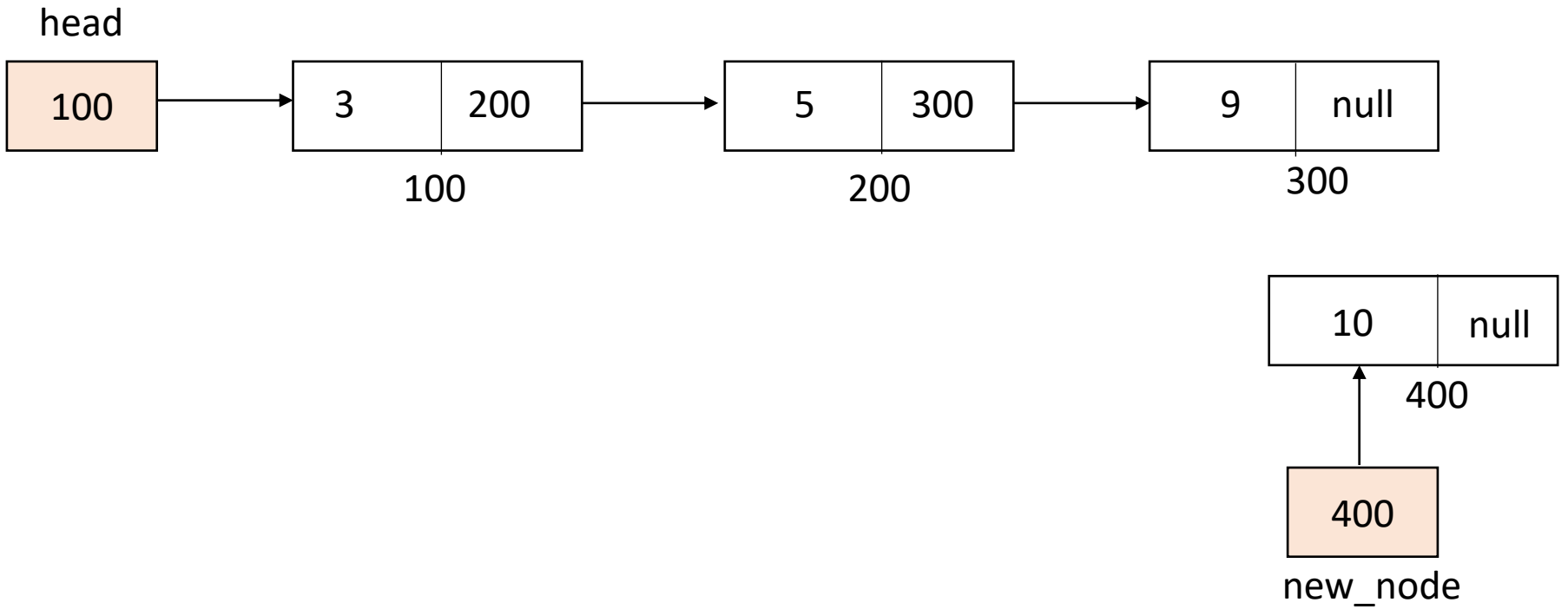
Insertion (at the beginning)

- Point head to the new node



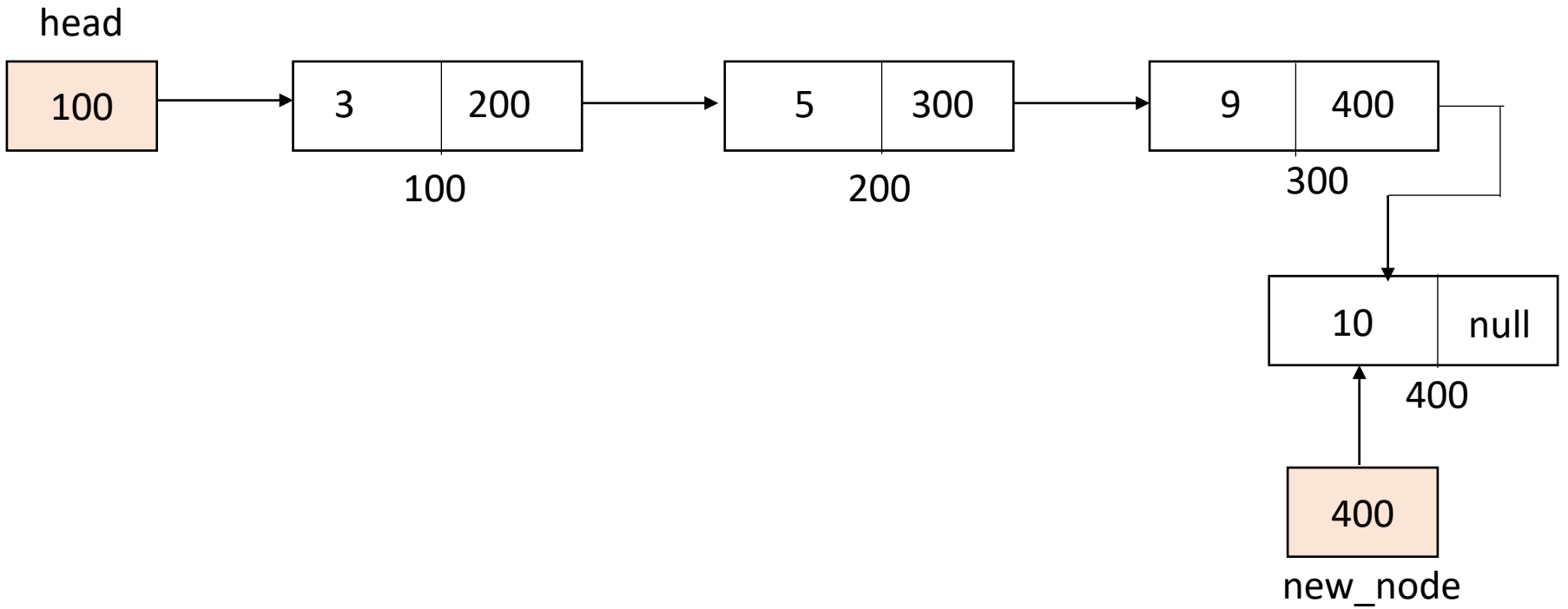
Insertion (at the end)

- Create a new node and traverse to the last node



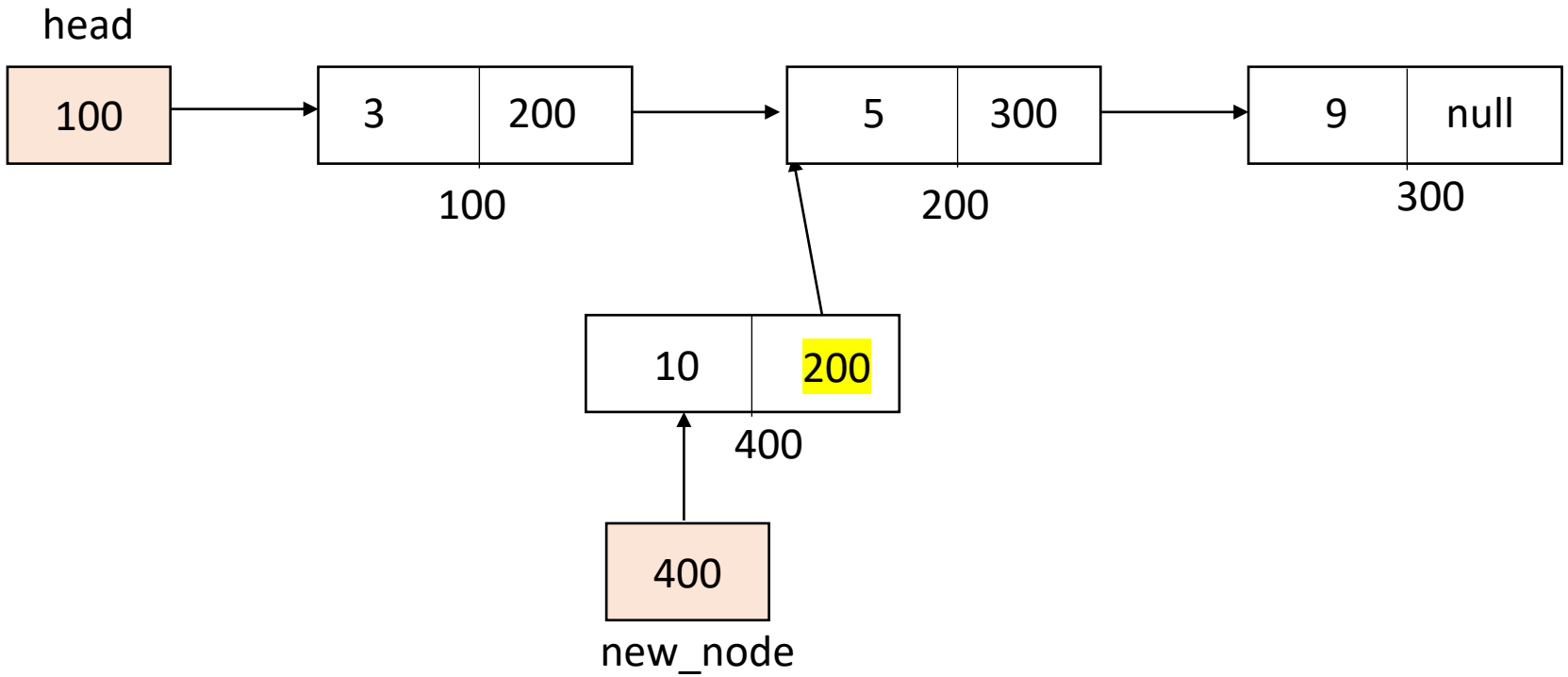
Insertion (at the end)

- Point new node to the last node



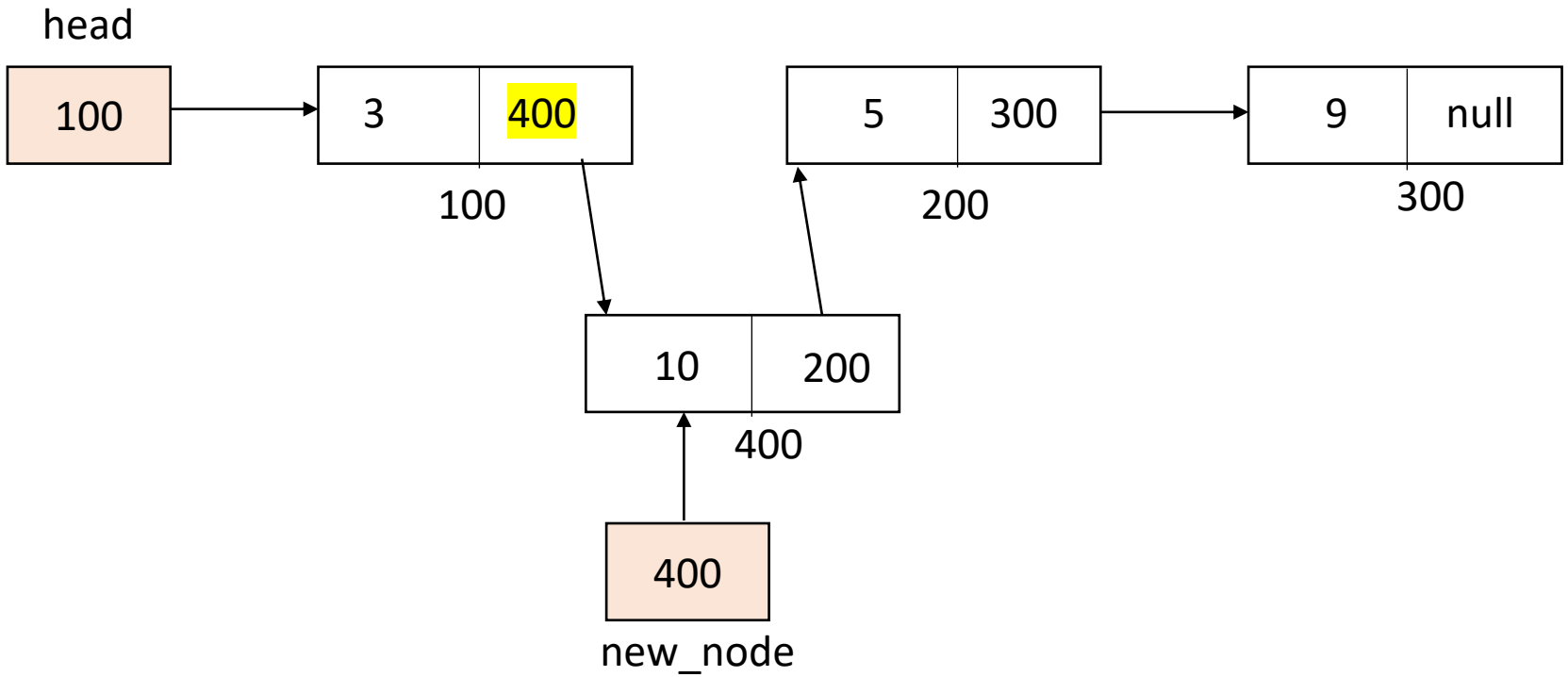
Insertion (after a given location)

- Create a new node and assign next node's address to new node's next value



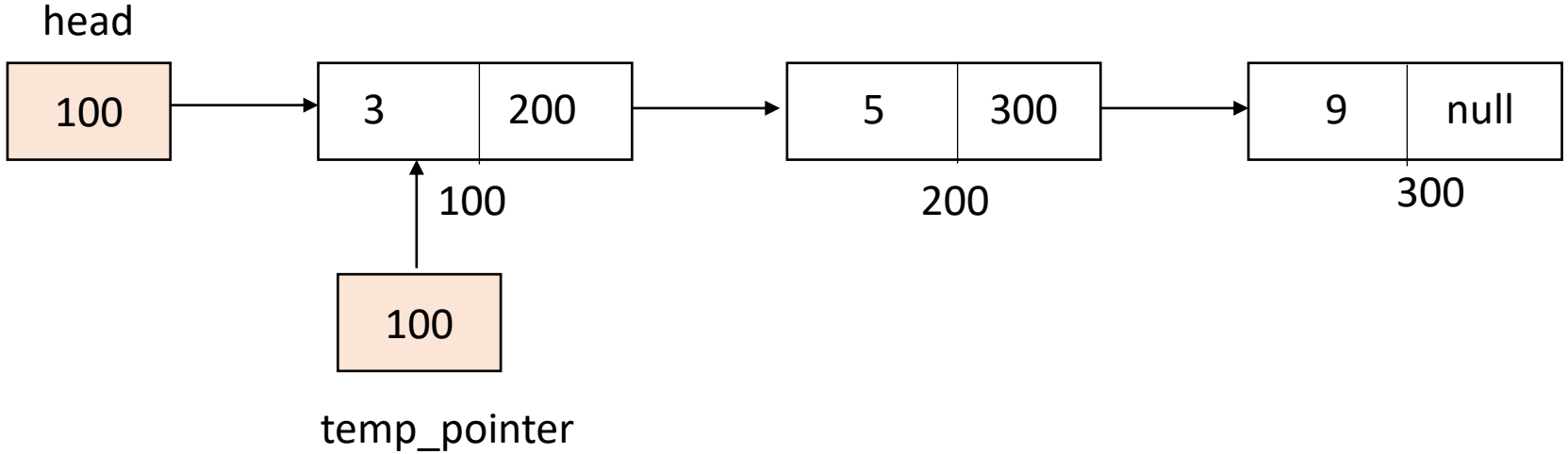
Insertion (after a given location)

- Point previous node's next value to new node's address



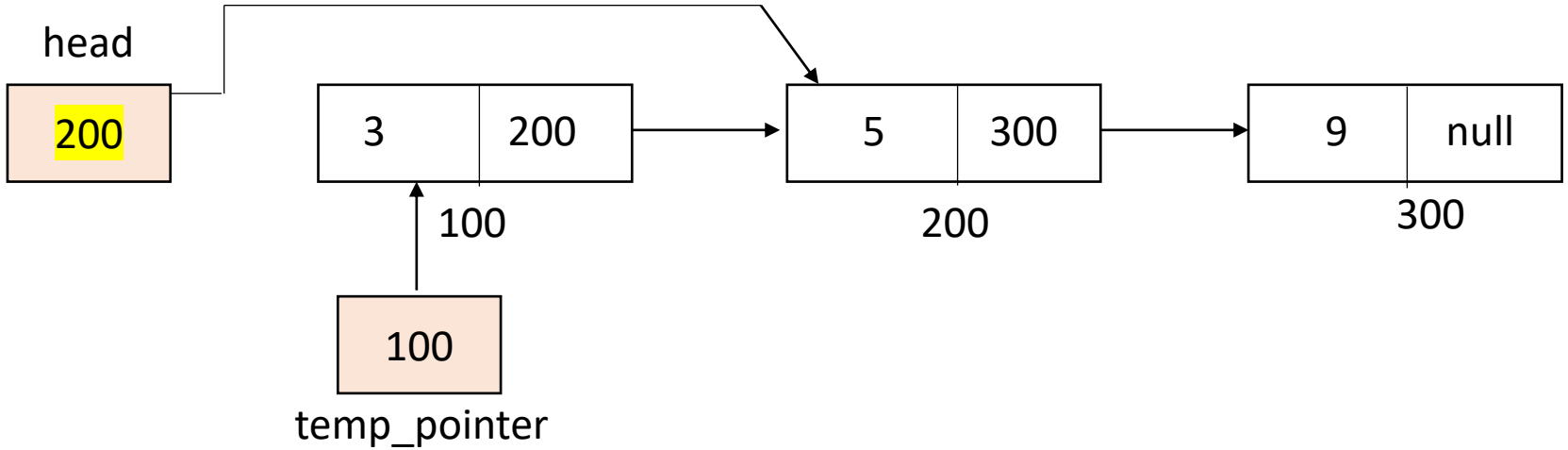
Deletion (from the beginning)

- Delete the first element
- Step 1 – Create a pointer and point first node to it



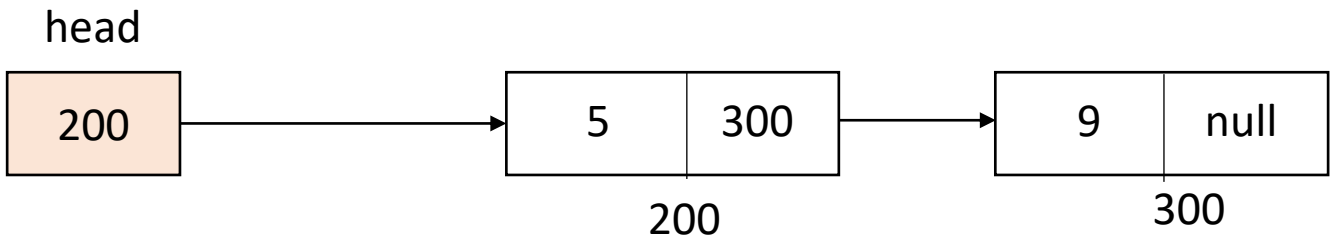
Deletion (from the beginning)

- Step 2 – Point head to the second node and delete first node.



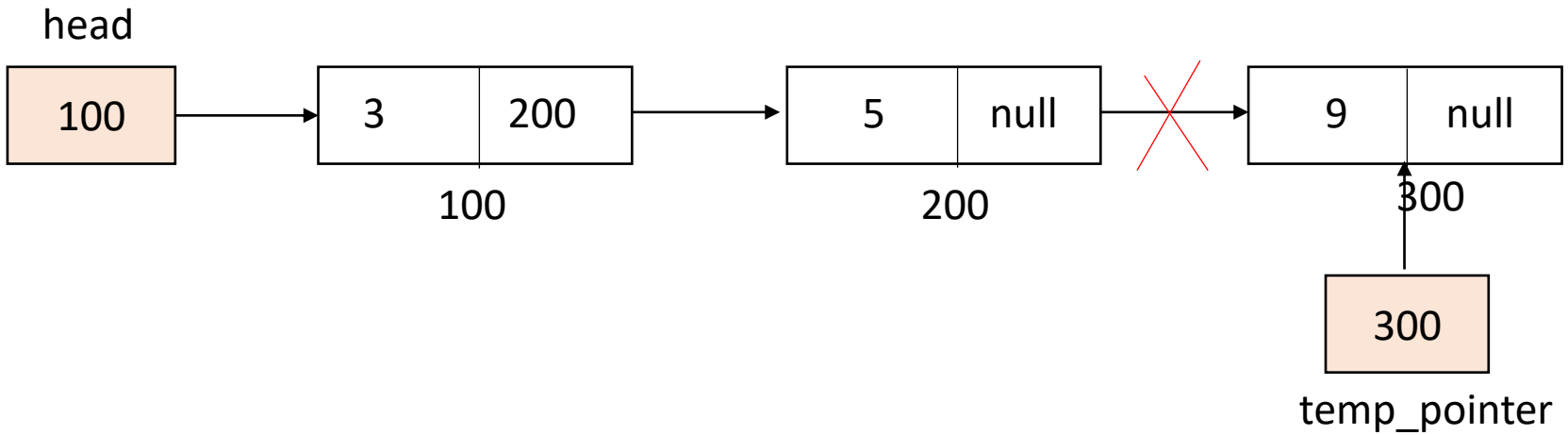
Deletion (from the beginning)

- Step 3 – Delete the first node. (Free allocated space)



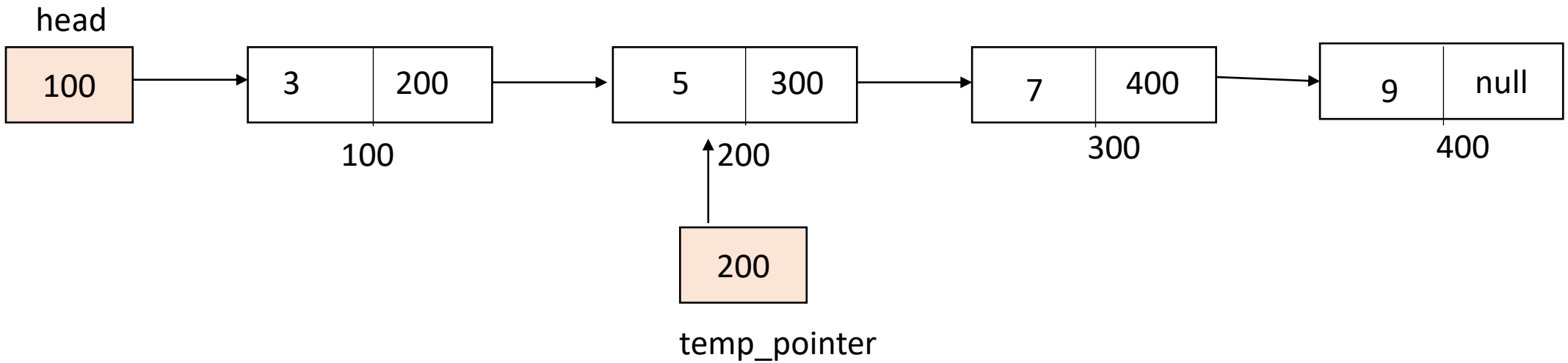
Deletion (from the end)

- Change the previous node's link value to null and delete last node.



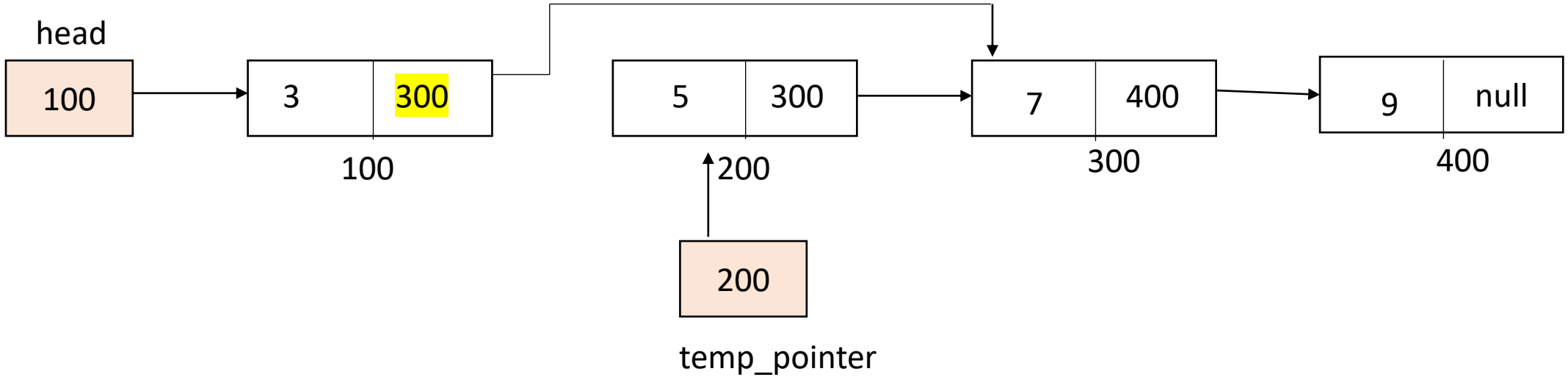
Deletion (from the middle)

Step 1 – Create a pointer and point target node to it.

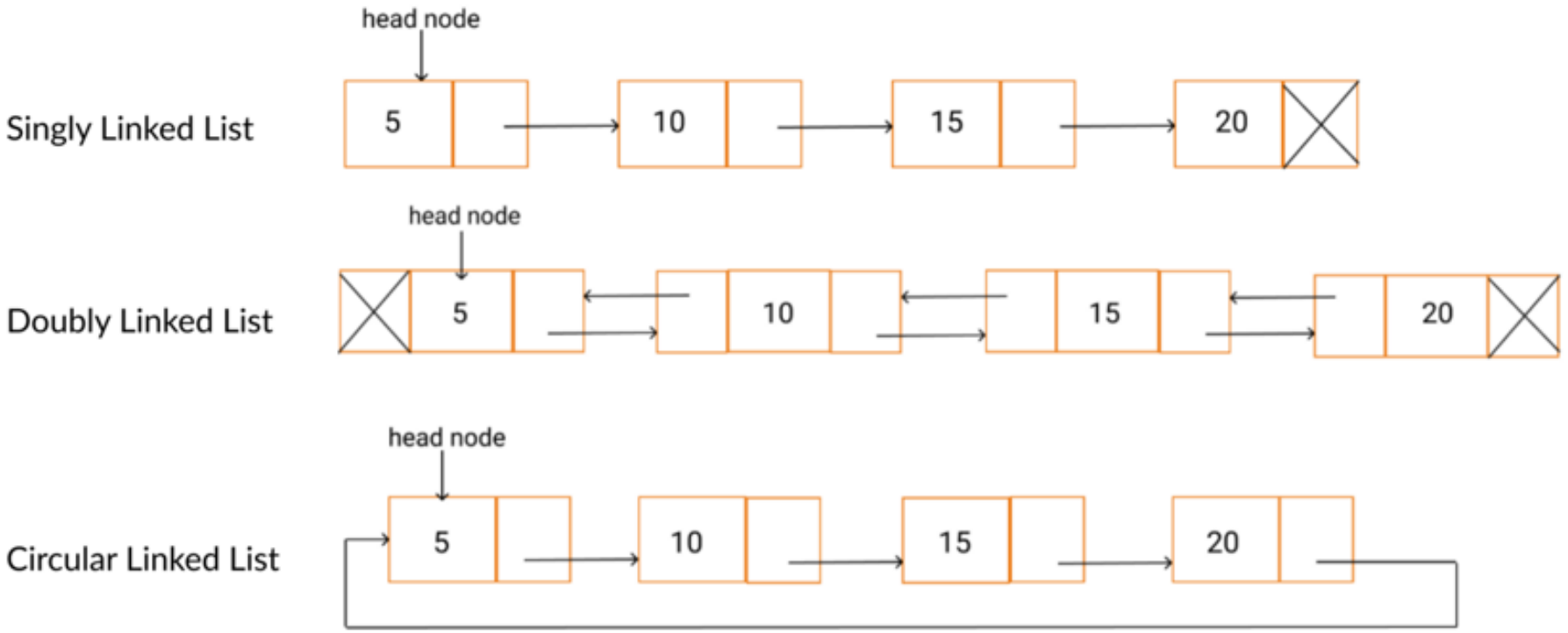


Deletion (from the middle)

- Delete the target element.



Types of Linked List



Arrays vs Linked List

- Cost of accessing an element
- Memory utilization
- Memory requirement
- Cost of insertion
- Cost of searching

Arrays vs Linked List

- Arrays are suitable for:
  - Inserting/deleting an element at the end.
  - Randomly accessing any element
  - Searching the list for a particular value
- Linked lists are suitable for:
  - Inserting an element
  - Deleting an element.
  - Applications where sequential access is required.
  - In situations where the number of elements cannot be predicted before.

Linked List vs Array

ARRAY	LINKED LIST
Size of an array is fixed	Size of a list is not fixed
Memory is allocated from stack	Memory is allocated from heap
It is necessary to specify the number of elements during declaration (i.e., during compile time).	It is not necessary to specify the number of elements during declaration (i.e., memory is allocated during run time).
It occupies less memory than a linked list for the same number of elements.	It occupies more memory.
Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room.	Inserting a new element at any position can be carried out easily.
Deleting an element from an array is not possible.	Deleting an element is possible.

Thank You

In linked list each node contains a minimum of two fields. One field is data field to store the data second field is?

a) Pointer to character
b) Pointer to integer
c) Pointer to node
d) Node

Which of the following is not a disadvantage to the usage of array?

a) Fixed size
b) There are chances of wastage of memory space if elements inserted in an array are lesser than the allocated size
c) Insertion based on position
d) Accessing elements at specified positions

Consider an implementation of unsorted singly linked list. Suppose it has its representation with a head pointer only. Given the representation, which of the following operation can be implemented in O(1) time?

i) Insertion at the front of the linked list

ii) Insertion at the end of the linked list

iii) Deletion of the front node of the linked list

iv) Deletion of the last node of the linked list

a) I and II

b) I and III

c) I, II and III

d) I, II and IV