IBA

IBA FCS
Faculty of Computer Science

CSE 247
Data Structures

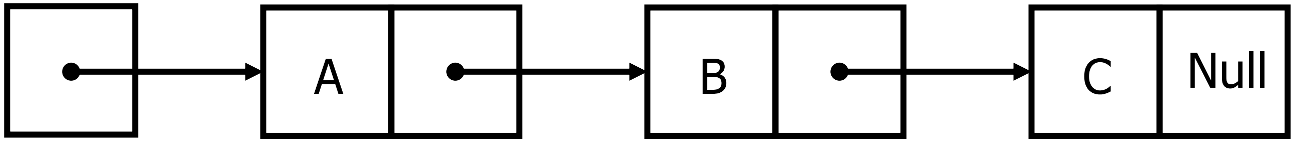
Linked list

outline

- Linked lists
  - Motivation
  - Abstract data type (ADT)
- Basic operations of linked lists
  - Insert, find, delete, traversal, etc.

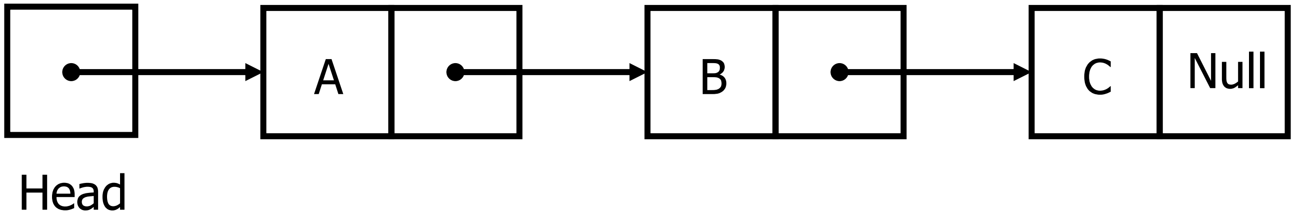
Motivation

- The organization of data is very important because the organization can effect the performance of operations.
- The linked list is a linear data structure as a collection of connected nodes. Each node consists of two parts DATA and LINK.

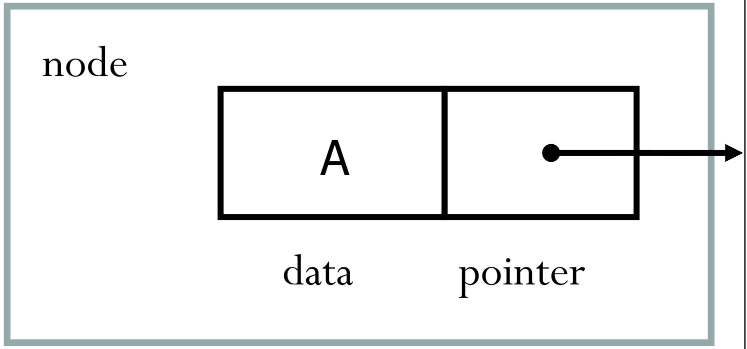


- The basic linked list operations are Insertion, Deletion, Find … so on

Linked Lists

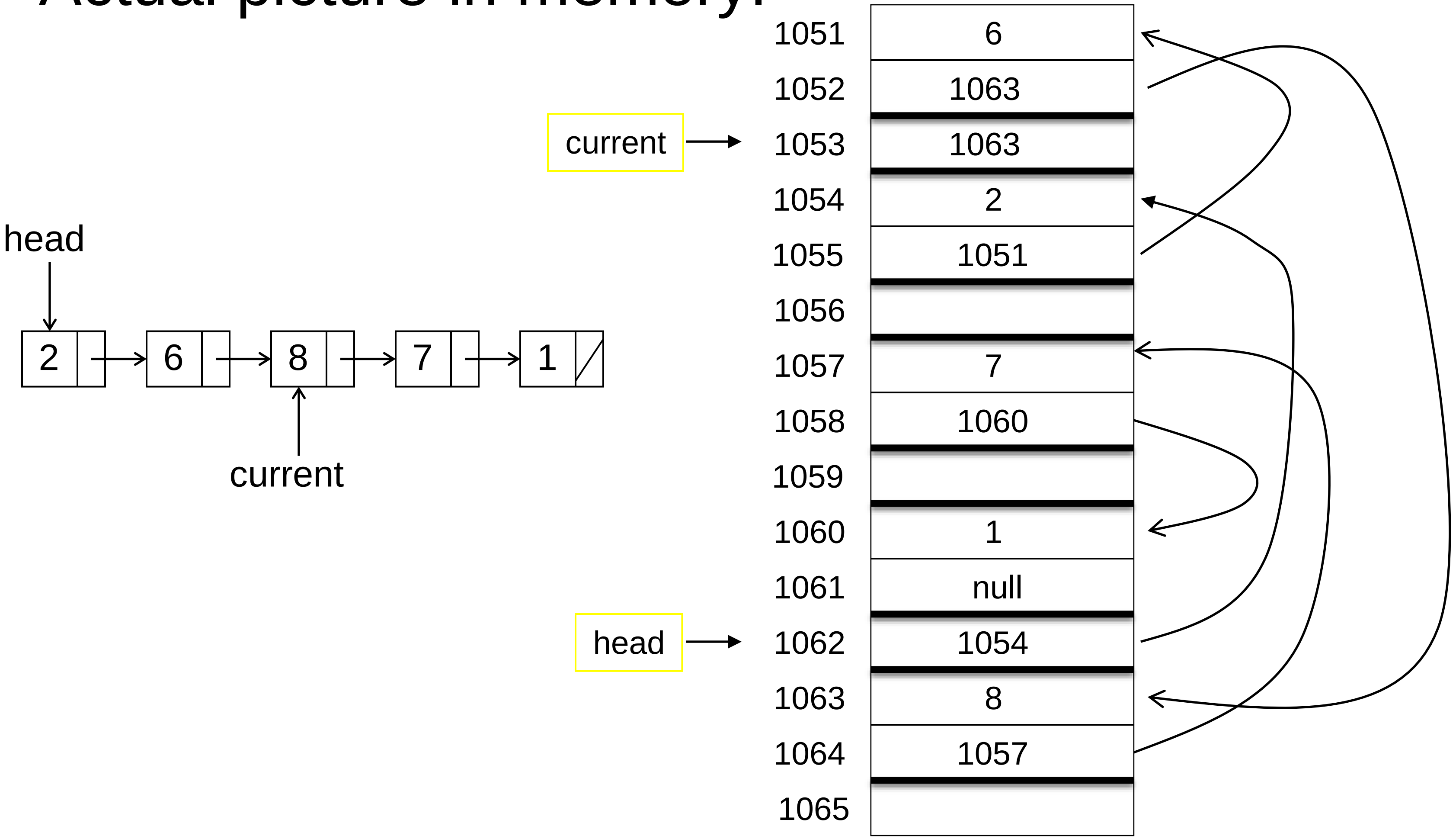


- A *linked list* is a series of connected *nodes*
- Each node contains at least
  - A piece of data (any type)
  - Pointer to the next node in the list
- *Head*: pointer to the first node
- The `Null indicates end of list`



Linked List

- Actual picture in memory:



1051	6
1052	1063
1053	1063
1054	2
1055	1051
1056	
1057	7
1058	1060
1059	
1060	1
1061	null
1062	1054
1063	8
1064	1057
1065	

Basic operations on linked list

Following are the basic operations supported by a list.

Insertion:

- Add an element at the end of the list.
- Adds an element at the beginning of the list.
- Add an element in sorted order in a list.

Deletion:

- Delete an element at the end of the list.
- Delete an element at the beginning of the list.
- Delete an element using given key.

Display: Displays the complete list.

isEmpty: determine whether or not the list is empty.

clearList: delete all node from the list and make it an empty list.

Search: Searches an element using the given key.

Implementation of Node class

```
class Node{
int data;
Node next;
 Node () {    // no argument constructor
     data=0;
     next=null;
 }
Node (int V) {      // one argument constructor
 data=V;
 next=null;
 } }
```

Implementation of List class

```
class LIST{
Node Head;
LIST (){
Head=null;
}
// list operations (insertion, deletion, …)
}
```

Insert operation in linked list

```
public void INSERT(int value){
Node N=new Node(value);
Node Temp=null;
   if(Head==null){
           Head=N;
   }else{
          Temp=Head;
           while(Temp.next!=null){


                       Temp=Temp.next;
               }
           Temp.next=N;
   }  }
```

Display list items

```
public void DisplayList(){
Node Temp=Head;

    while(Temp!=null){

    System.out.println(Temp.data);
    Temp=Temp.next;
    }

}
```

Application of Singly linked list

- implementation of stacks and queues
- Operations in databases
- In word processor, del
- Operating system cache scheduling algorithm (LRU)
- In browser BACK/FORWARD button. (List of urls)
- Performing arithmetic operation on long integers.

Analysis of Singly linked list

- Insertion and deletion at current reference is one step operation.
- No need of contiguous free memory for whole list.
- Search entire list when:
    - To find an item that is not stored in a list (or stored at the end.)
- Moving back from current position is requires to traverse a list again from beginning. This problem is resolved by doubly linked list.

Quiz

- Can binary search be performed over linked list?
- What is the big-oh of each operation (insert, delete and search) on linked list.
- Discuss pros and cons of using linked list as compare to array-list?