Linked Lists

Saikrishna Arcot
M. Hudachek-Buswell

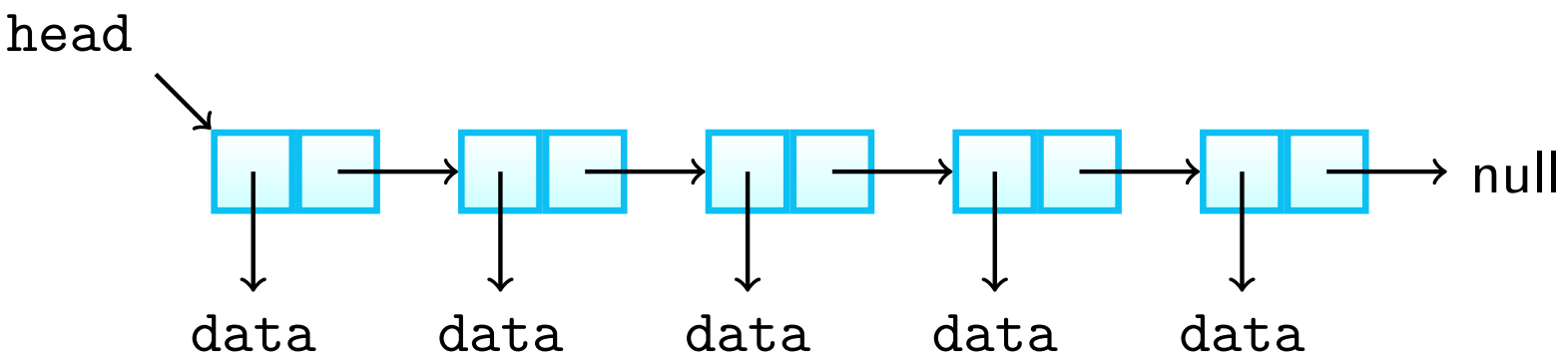
July 19, 2020

Recall Linked Lists

- A linked list is a data structure consisting of a sequence of nodes.
- Each node in the linked list stores the data and a pointer to the next/previous node(s).
- Nodes are created/destroyed as necessary. When adding a new element, a new node is created, and linked into the linked list. When removing an element, the node containing the element is destroyed.
- Linked list have a head pointer, and may have a tail pointer.

Recall Singly Linked Lists

We refrain from creating "dummy" nodes that lead to null references at the end of a singly linked list. We just have the next reference of the last node point to null.



Doubly Linked List

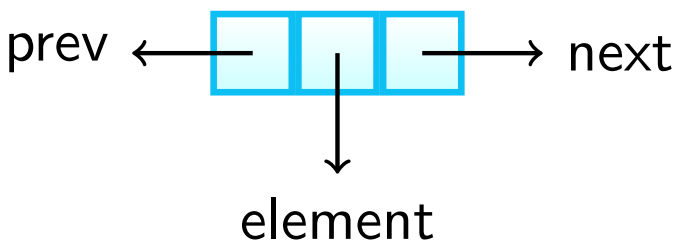
- A doubly linked list is a version of a linked list where each node has a pointer to the next node *and* a pointer to the previous node.
- Doubly linked lists can be traversed forward and backward.

Doubly Linked List

- A doubly linked list is a version of a linked list where each node has a pointer to the next node *and* a pointer to the previous node.
- Doubly linked lists can be traversed forward and backward.
- Nodes store:
  - element
  - link to the previous node
  - link to the next node

Doubly Linked List

- A doubly linked list is a version of a linked list where each node has a pointer to the next node *and* a pointer to the previous node.
- Doubly linked lists can be traversed forward and backward.
- Nodes store:
  - element
  - link to the previous node
  - link to the next node



Generic Code For Doubly Linked List Node

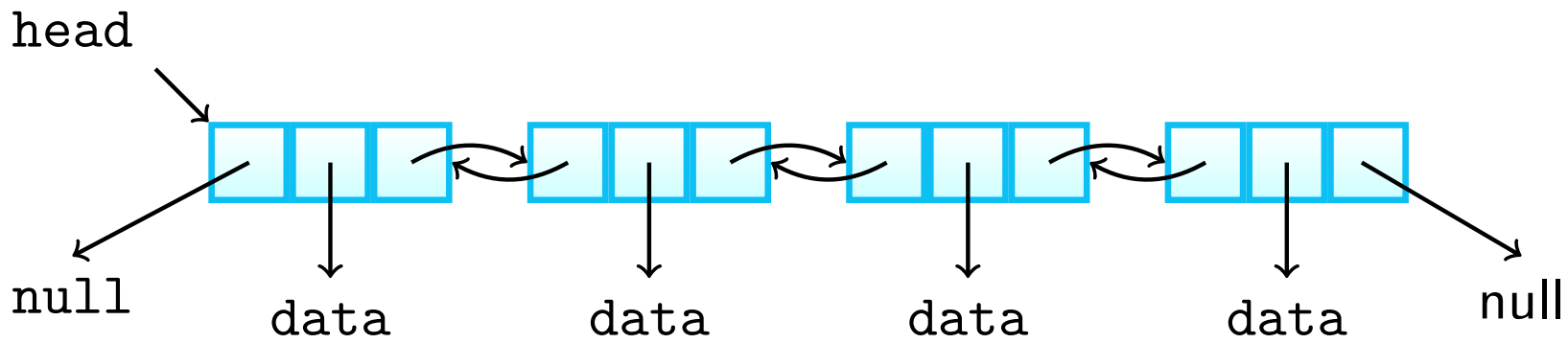
```
public class DoublyLinkedList<Type> {
    private class Node<Type> {
        private Type data;
        private Node<Type> next;
        private Node<Type> prev;

        private Node(Type data, Node<Type> next,
          Node<Type> prev) {
            this.data = data;
            this.next = next;
            this.prev = prev;
        }
    }
}
```

Generic Code For Doubly Linked List Node

```
// Node constructor chaining
private Node(Type data) {
    this(data, null, null)
// this.data = data
// this.next = null
// this.prev = null
}
```

Doubly Linked List

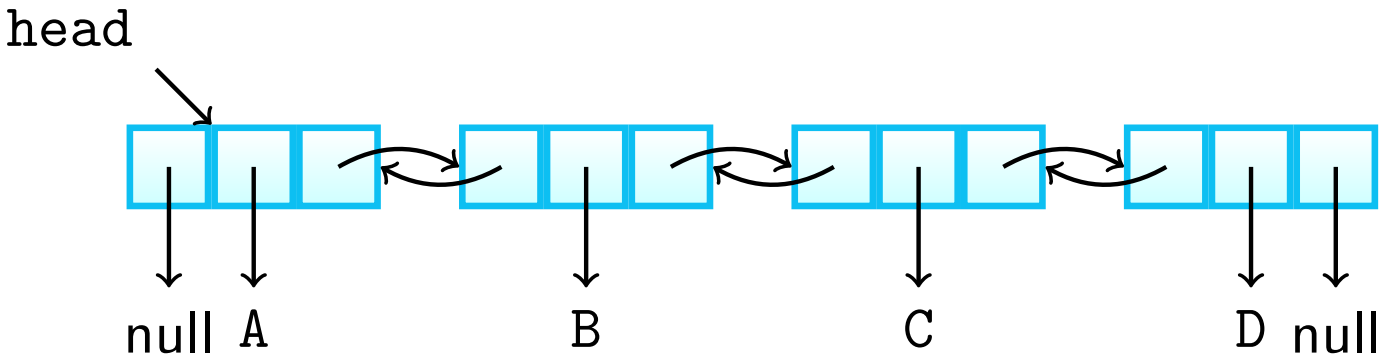
Example of a doubly linked list:



Notice that `prev` reference of the first node points to `null`, as does the `next` reference of the last node

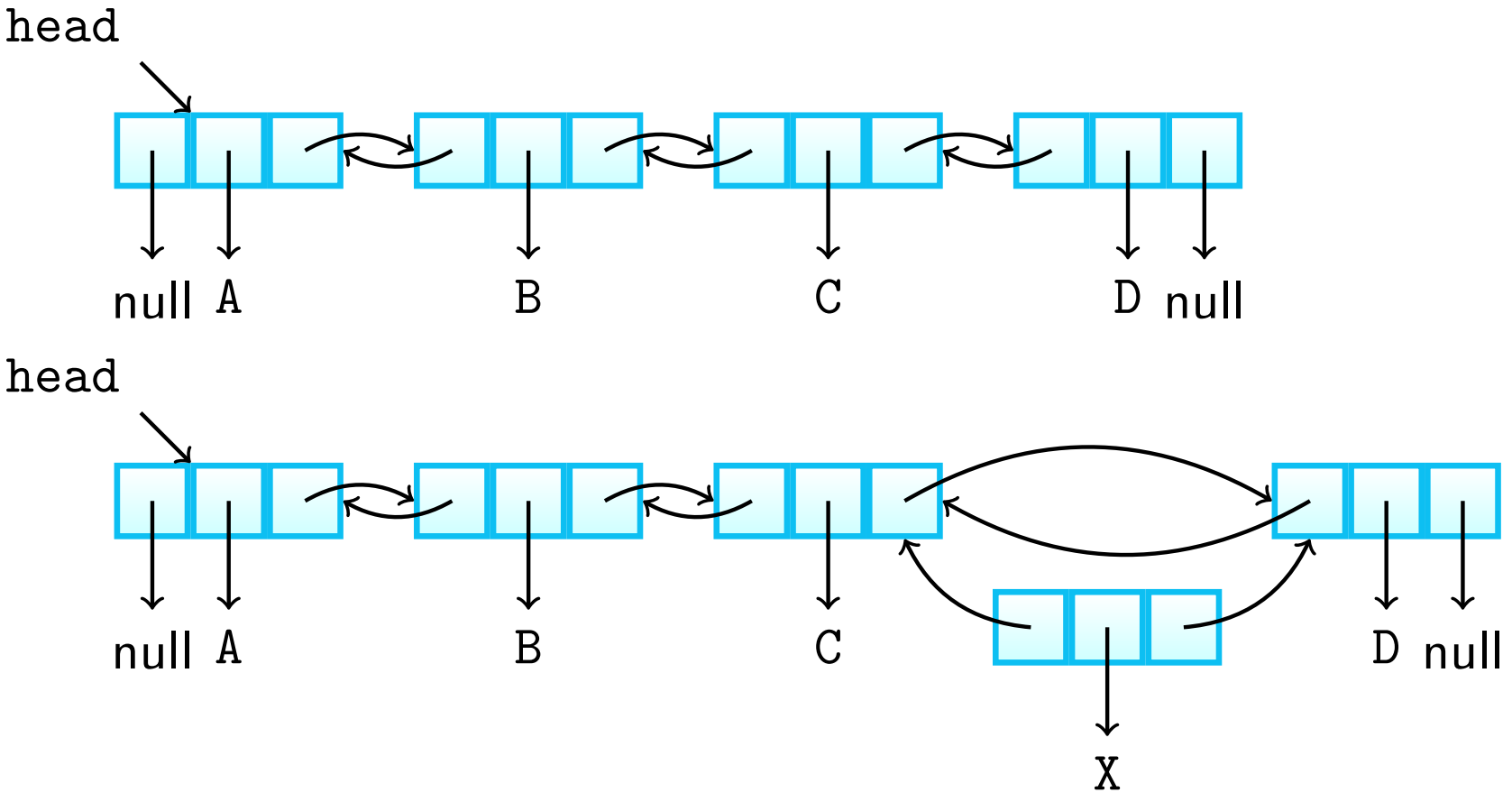
Insertion

Create new node `X`. Insert a new node, `X`, between node `C` and node `D`. Set new node's `prev` reference to `C` and `next` reference to `D`.

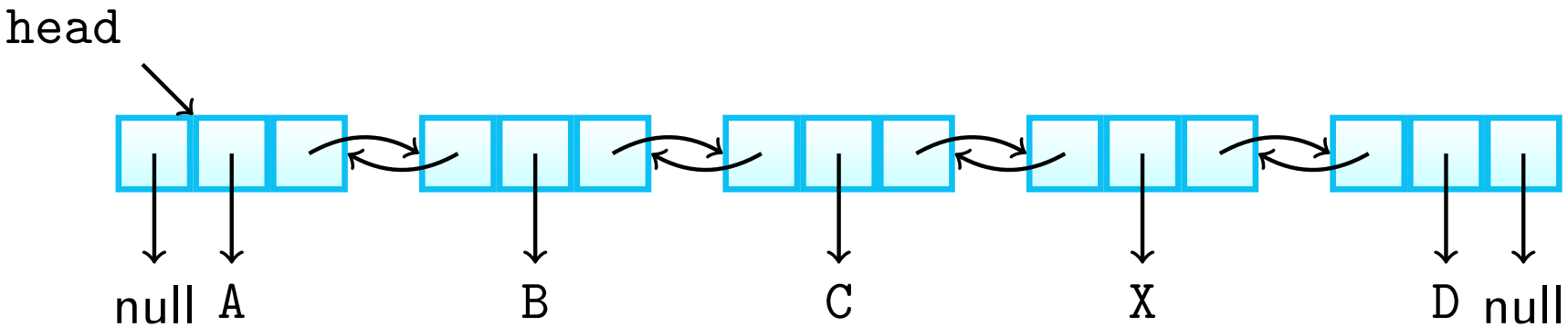


Insertion

Create new node `X`. Insert a new node, `X`, between node `C` and node `D`. Set new node's `prev` reference to `C` and `next` reference to `D`.



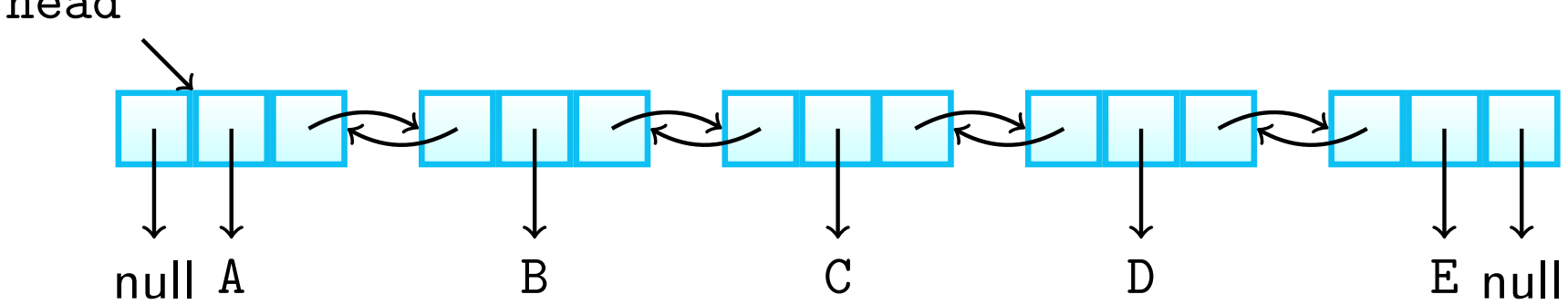
Insertion



Once node `X` is connected, set Node `C`'s `next` reference to node `X`, and Node `D`'s `prev` reference to node `X`.

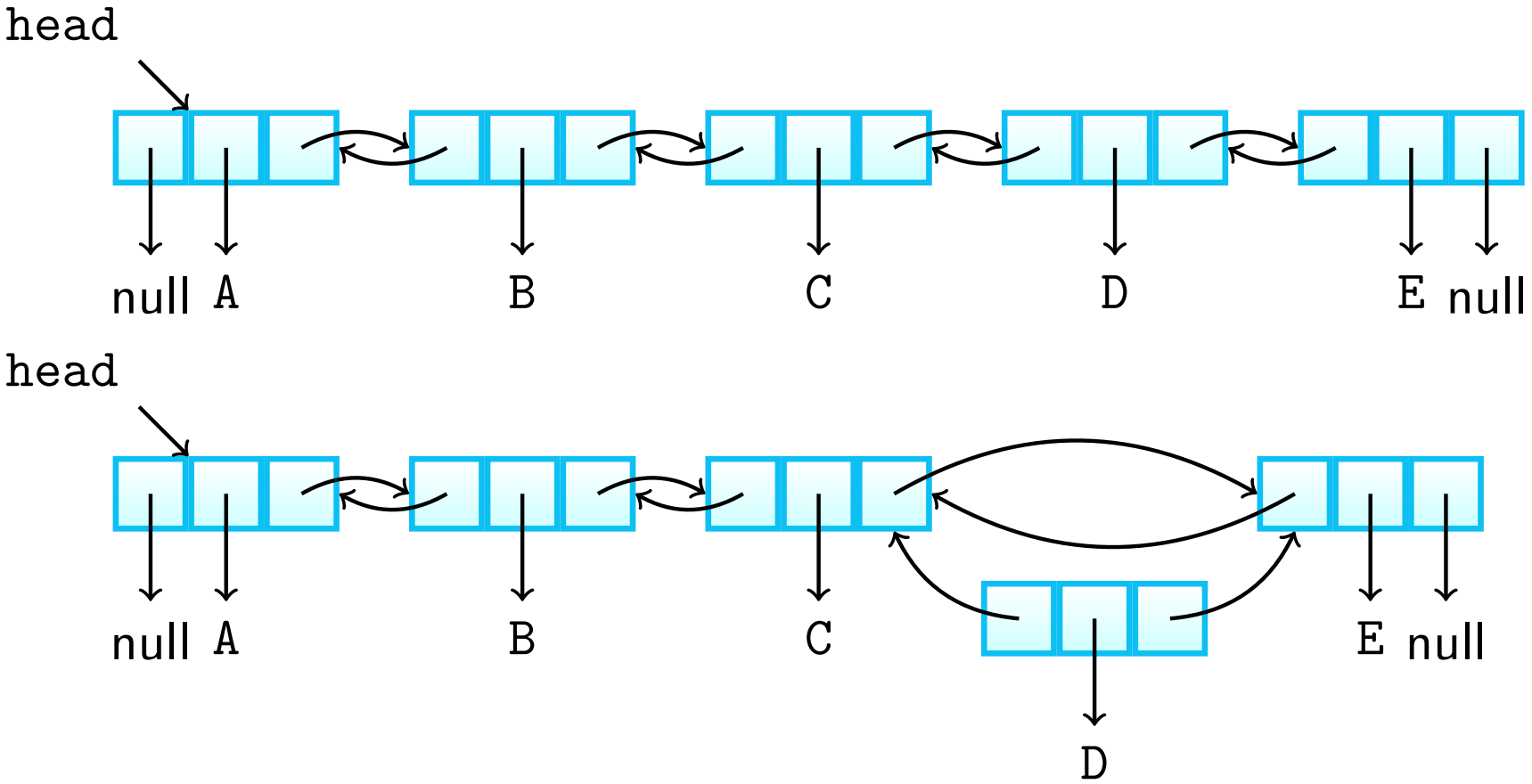
Deletion

Remove node D, between nodes C and E. The order in which you redirect references is important so you do not lose your linked list.



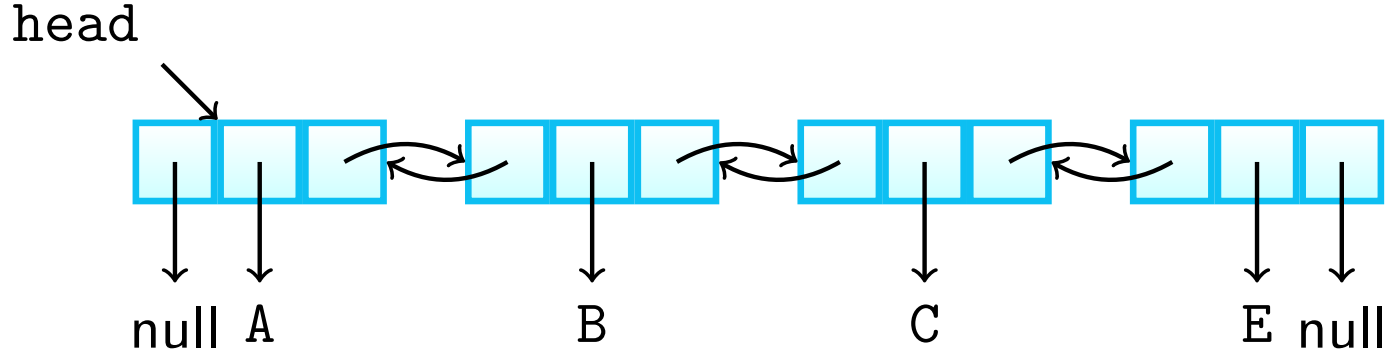
Deletion

Remove node D, between nodes C and E. The order in which you redirect references is important so you do not lose your linked list.



Deletion

Set node `C`'s `next` reference to node `E`, and node `E`'s `prev` reference to `C`. Node `D` will be garbage collected.



Linked List Summary and Complexity

- Linked Lists are most definitely dynamic, and can store primitives or objects.

Linked List Summary and Complexity

- Linked Lists are most definitely dynamic, and can store primitives or objects.
- Accessing or searching any linked list is a cost of O(n), linear time

Linked List Summary and Complexity

- Linked Lists are most definitely dynamic, and can store primitives or objects.
- Accessing or searching any linked list is a cost of O(n), linear time
- Adding or removing *from the front* of any linked list with a *head pointer* is a cost of O(1), constant time

Linked List Summary and Complexity

- Linked Lists are most definitely dynamic, and can store primitives or objects.
- Accessing or searching any linked list is a cost of O(n), linear time
- Adding or removing *from the front* of any linked list with a *head pointer* is a cost of O(1), constant time
- Linked lists are useful when you have to manipulate data

Linked List Summary and Complexity

- Linked Lists are most definitely dynamic, and can store primitives or objects.
- Accessing or searching any linked list is a cost of O(n), linear time
- Adding or removing *from the front* of any linked list with a *head pointer* is a cost of O(1), constant time
- Linked lists are useful when you have to manipulate data
- Singly linked lists are used in RFID scanning, app wizards or levels in a video game

Linked List Summary and Complexity

- Linked Lists are most definitely dynamic, and can store primitives or objects.
- Accessing or searching any linked list is a cost of O(n), linear time
- Adding or removing *from the front* of any linked list with a *head pointer* is a cost of O(1), constant time
- Linked lists are useful when you have to manipulate data
- Singly linked lists are used in RFID scanning, app wizards or levels in a video game
- Doubly linked lists are used in browser history, scroll bars or forward/back buttons

Circular Linked Lists

- Circular Linked List is a more complicated Linked list where the last node points to the first node, (and the first node could point to the last node).

Circular Linked Lists

- Circular Linked List is a more complicated Linked list where the last node points to the first node, (and the first node could point to the last node).
- Circular linked lists can be either singly or doubly linked.

Circular Linked Lists

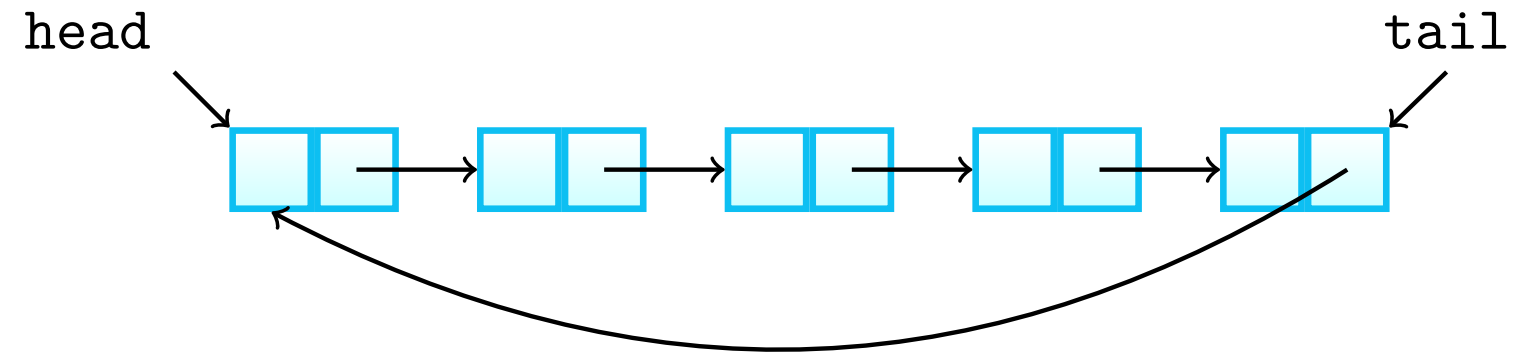
- Circular Linked List is a more complicated Linked list where the last node points to the first node, (and the first node could point to the last node).
- Circular linked lists can be either singly or doubly linked.
- Depending on single or double links, adding or removing is the same as for the type of link

Circular Linked Lists

- Circular Linked List is a more complicated Linked list where the last node points to the first node, (and the first node could point to the last node).
- Circular linked lists can be either singly or doubly linked.
- Depending on single or double links, adding or removing is the same as for the type of link
- Applications that use circular linked lists are gifs, music playlists; round-robin scheduling algorithms for operating systems

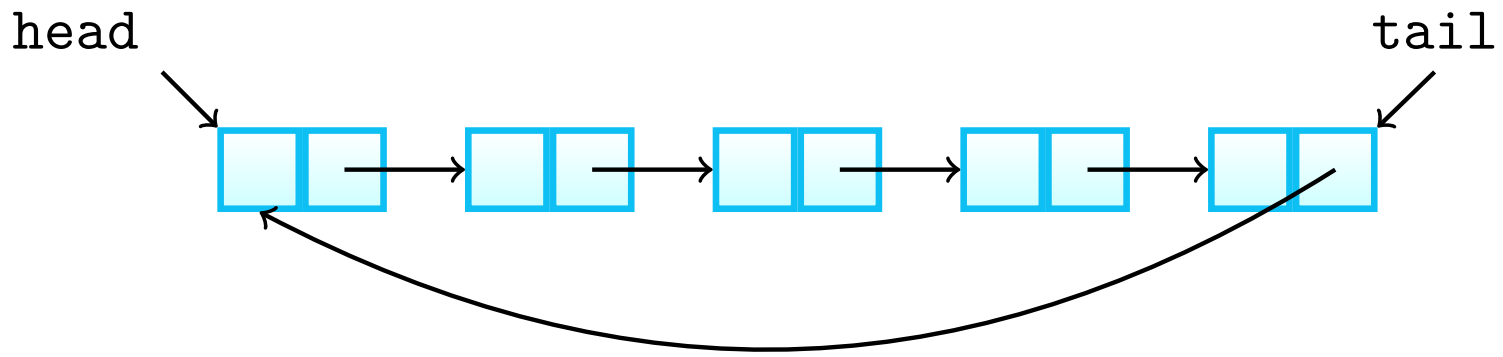
Circular Linked Lists

Example of Singly Circular Linked List



Circular Linked Lists

Example of Singly Circular Linked List



Example of Doubly Circular Linked List