CS 601-01: Stacks and Queues

Ayush Sharma & Safya Osman

August 29, 2022

What is a Stack?

Stacks: Let’s Start with an Analogy

Consider a stack of plates at a salad bar:

- Can add plate to top of stack
- Can remove plate from top of stack
- But can’t access plate from middle of stack—need to remove all plates above it first

In

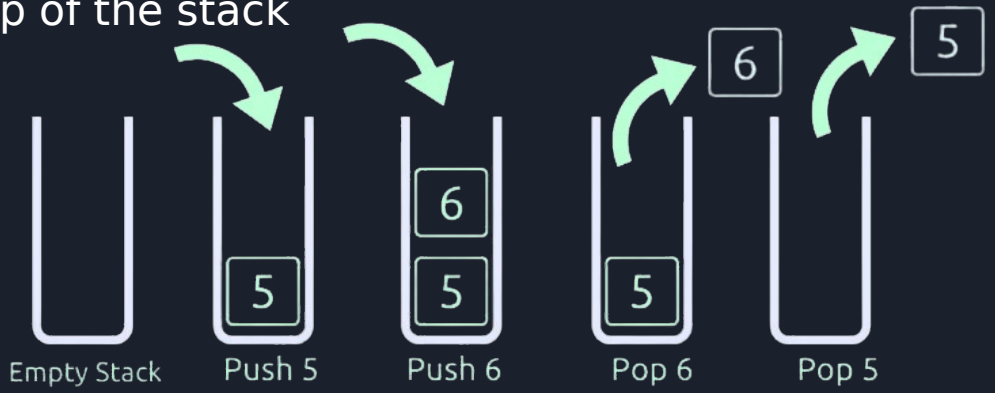
Out

Safy
a

Stacks: Definition

Definition: A stack stores a collection of elements one at a time with two main principles:

- **Push:** adds an element to the collection (to the **top** of the stack)
- **Pop:** removes and returns the most recently added element from the top of the stack



A stack is a **Last-In, First-Out (LIFO)** linear data structure

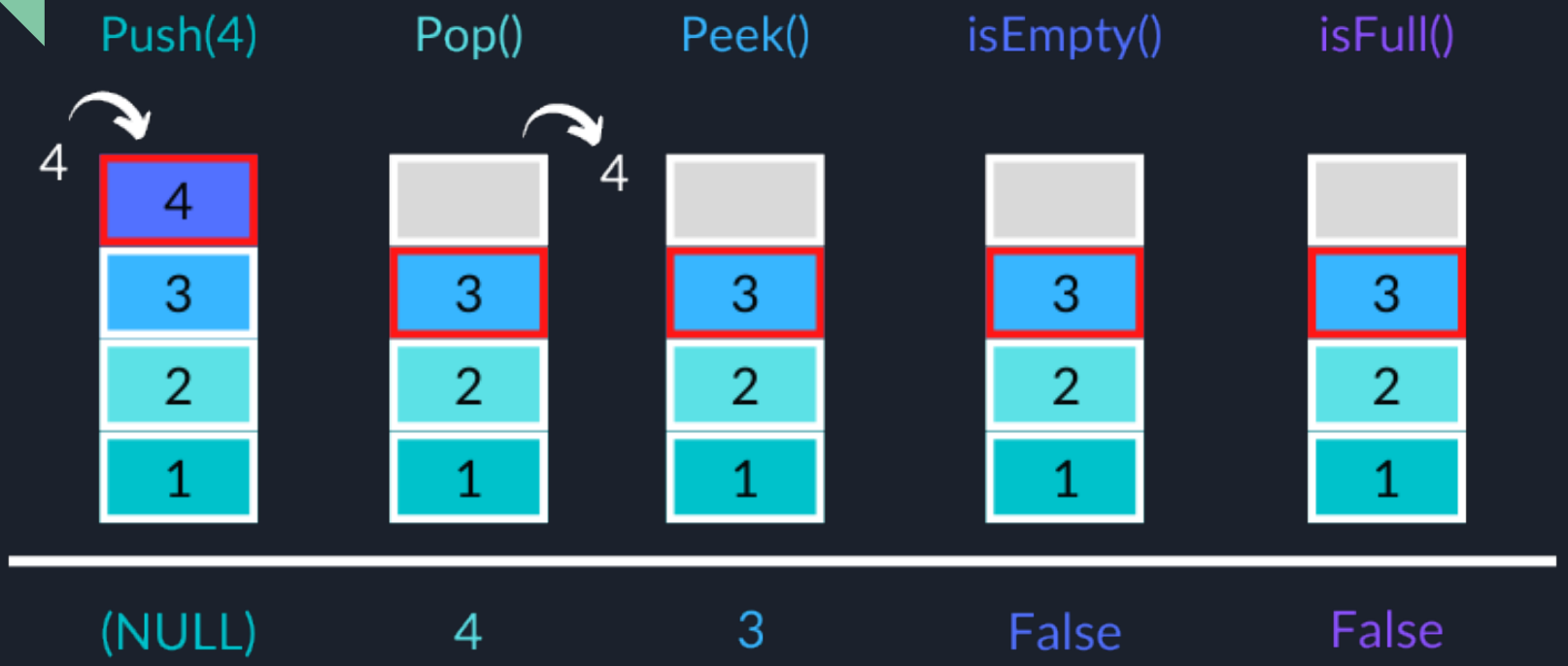
Safy a

Stacks: Basic Operations

Push	Adds element to top of stack.
Pop	Removes and returns the element at top of stack.
Peek	Returns the value of top element without removing it.
isEmpty	Checks if the stack is empty.
isFull	

Safya

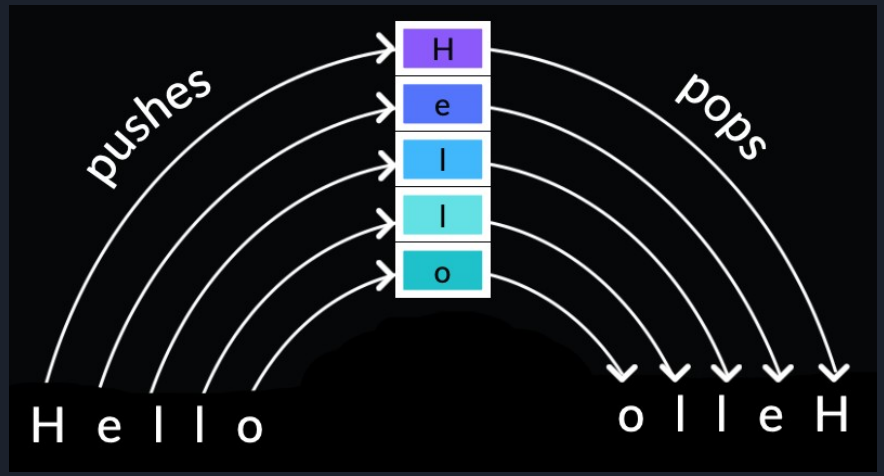
Stacks: Basic Operations

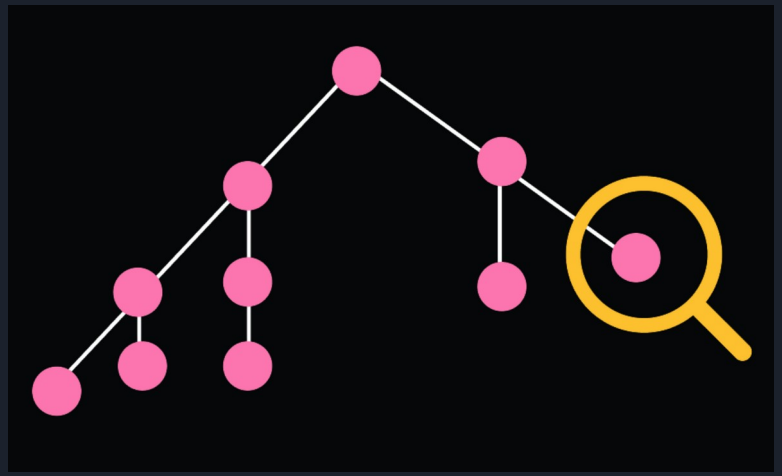


Safy
a

Stacks: Applications

- Undo & redo features for many editors
- Forward & backward features on web browsers
- Memory management in computers
- String reversal & tree traversal algorithms





Safy
a

Using Arrays to Implement Stacks

Using Arrays to Implement Stacks

- Stack data stored in array
- Index called **top** points to first unused element in array
- When element is **pushed** onto stack, it is stored at the **top** index, and **top** is then incremented
- When element is **popped**, **top** is decremented and element at index **top** is returned
- If **top == 0**, then stack is empty

Safya

Using Arrays to Implement Stacks

Top: **3**
Size = **5**

2. **Push(d)**

Top: **4**

3. **Push(e)**

Top: **5**

4. **isFull()**

Top: **5**
Return: **True**

5. **Pop()**

Top: **4**
Return: **e**

6. **Pop()**

Top: **3**
Return: **d**

7. **Pop()**

Top: **2**
Return: **c**

8. **Pop()**

Top: **1**
Return: **b**

Safya

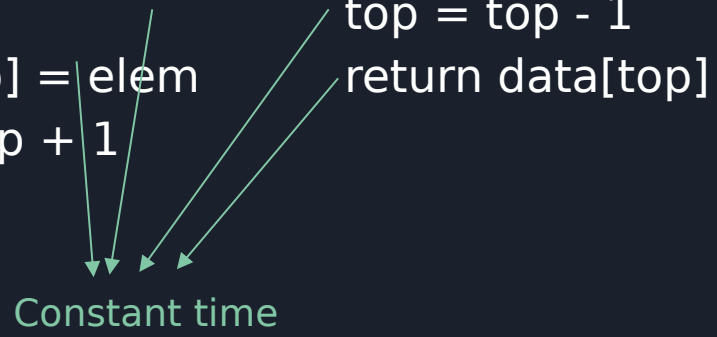
Using Arrays to Implement Stacks:
Pseudocode & Time Complexity Analysis

```
class
ArrayStack
    Object data[]
    int top
    int size

    Constructor()
```

```
void Push(Object
elem)
    data[top] = elem
    top = top + 1
```

```
Object Pop()
    top = top - 1
    return data[top]
```



Total # of Basic Operations: f(n) = C’ (some constant time)

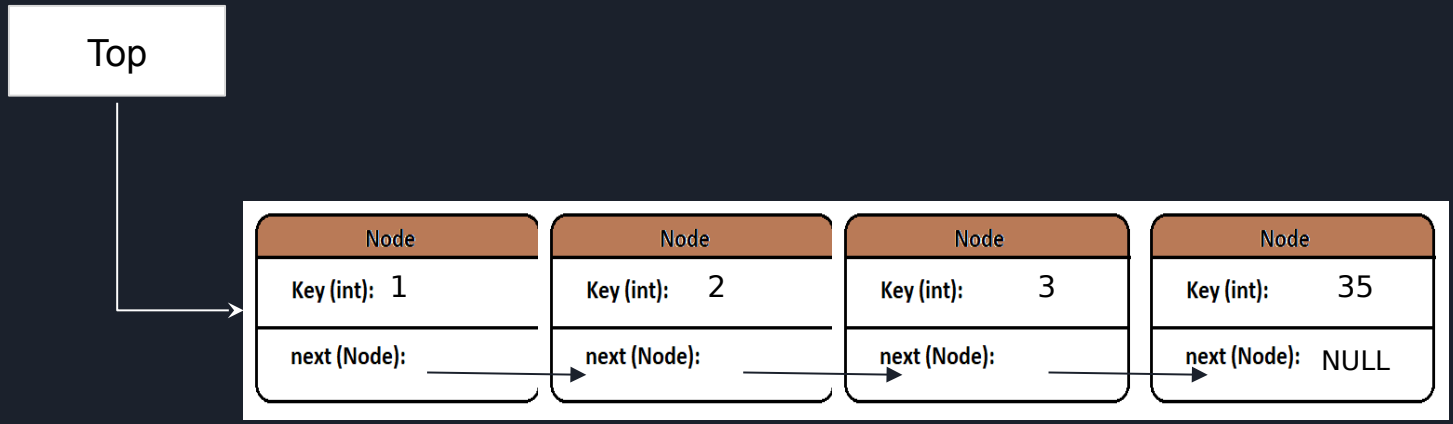
Therefore, stack insertion and deletion operations have constant time complexity in the worst case. ⇒ **O(1)**

Safya

Implementing Stack using Linked List

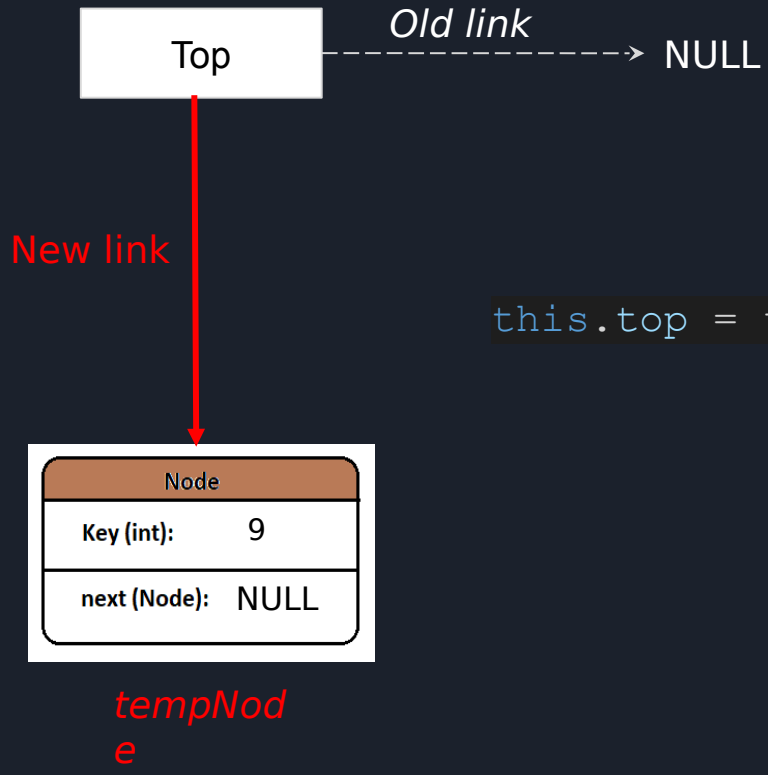
Implementing Stack using Linked List

- The main advantage of implementing stack using Linked List rather than Arrays is that, we will never encounter stack overflow situation since the nodes are created dynamically.
- In case of Arrays we have to restrict the stack to the Arrays size.



Ayush

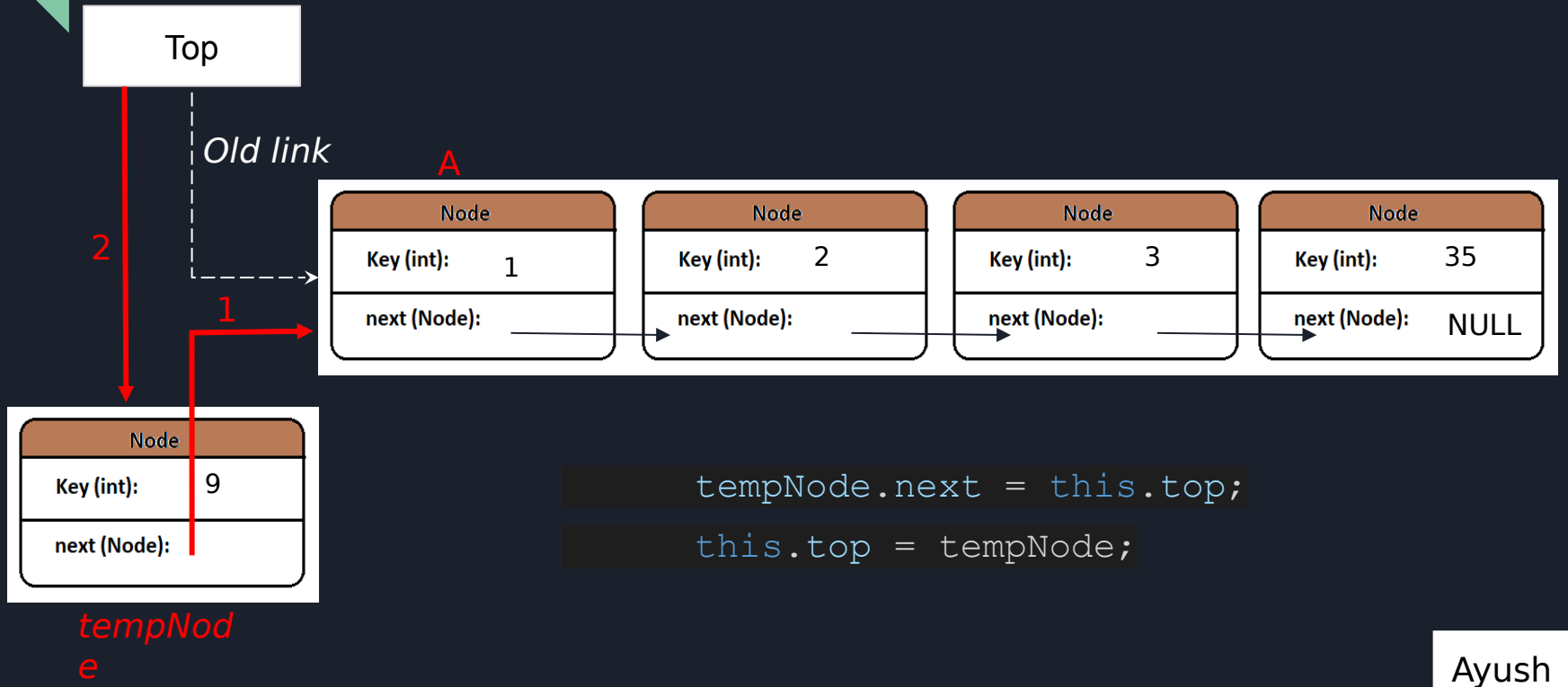
Push operation in action (*empty stack*)




```
this.top = tempNode;
```

Ayush

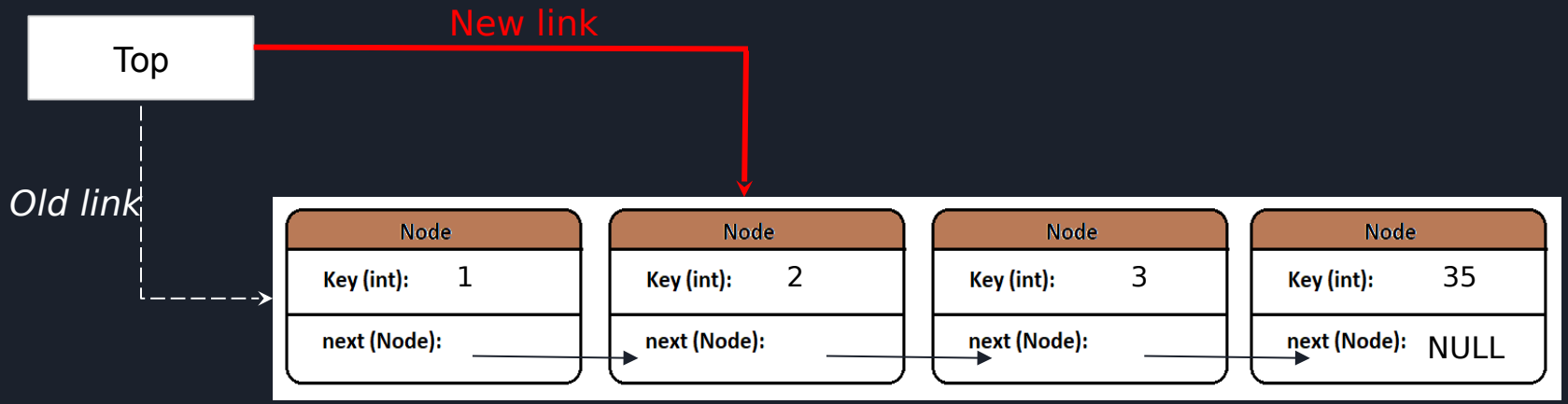
Push operation in action (*non-empty stack*)




```
tempNode.next = this.top;
this.top = tempNode;
```

Ayush

Pop operation in action




```
this.top = this.top.next;
```

Ayush

What is a Queue?

What is a Queue?

A queue is similar to the ticket queue outside a cinema hall, or at the bank where the first person entering the queue is the first person who gets the ticket or gets to the banker window.

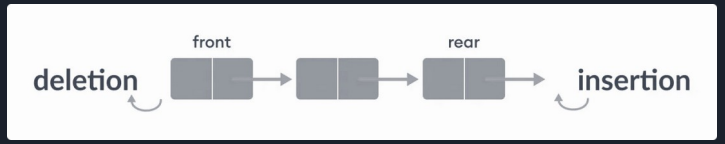
Queue follows the First In First Out (FIFO) rule.

Ayush

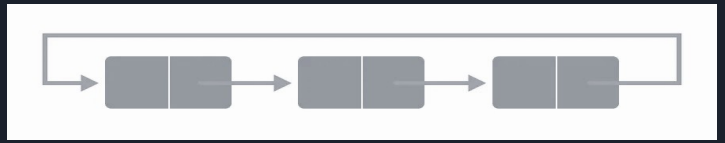
Source: https://graphicriver.net

Types of Queue

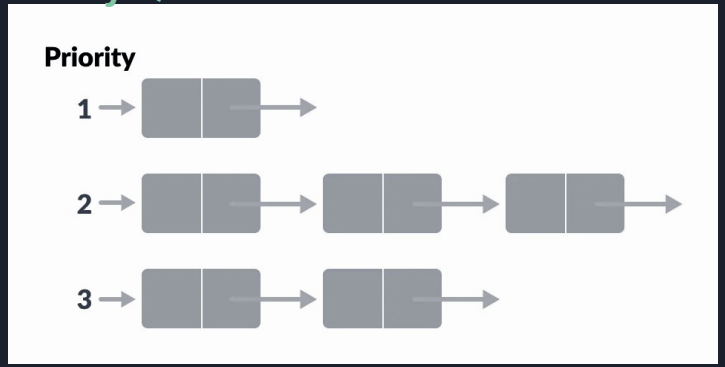
1. Linear Queue



2. Circular Queue



3. Priority Queue



4. Double Ended Queue

Ayush

Source: https://www.programiz.com

Basic operations on Queue

- **Enqueue**: Add an element to the end of the queue
- **Dequeue**: Remove an element from the front of the queue
- **IsEmpty**: Check if the queue is empty
- **IsFull**: Check if the queue is full
- **Peek**: Get the value of the front of the queue without removing it

Ayush

Source: https://www.programiz.com

Application of Queues

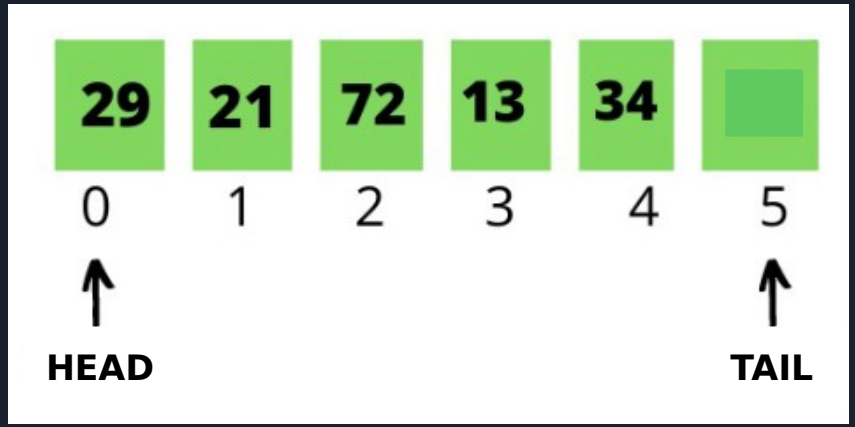
- CPU scheduling, Disk Scheduling.
- IO Buffers, pipes, etc.
- Spooling in printers
- Buffer for devices like keyboard

Ayush

Using Arrays to Implement Queues

Using Arrays to Implement Queues

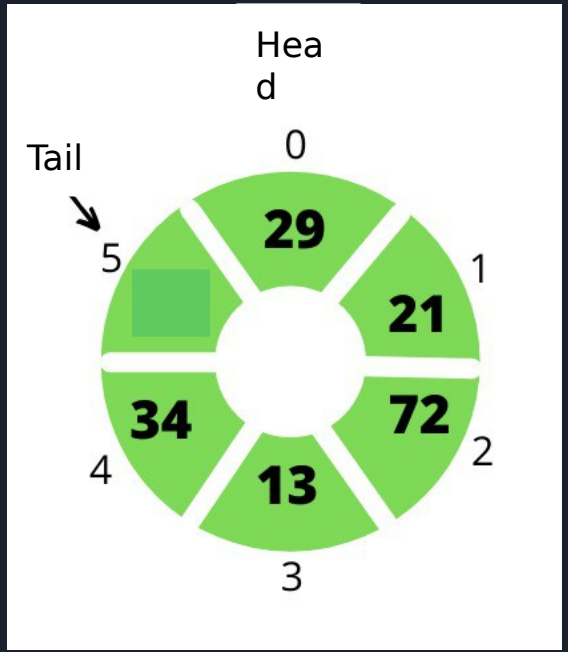
- Queue data stored in array
- Maintain **head** index at front of queue and **tail** index at end of queue
- To enqueue, add element at **tail** and increment **tail**
- To dequeue, return element at **head** and increment **head**



Safya

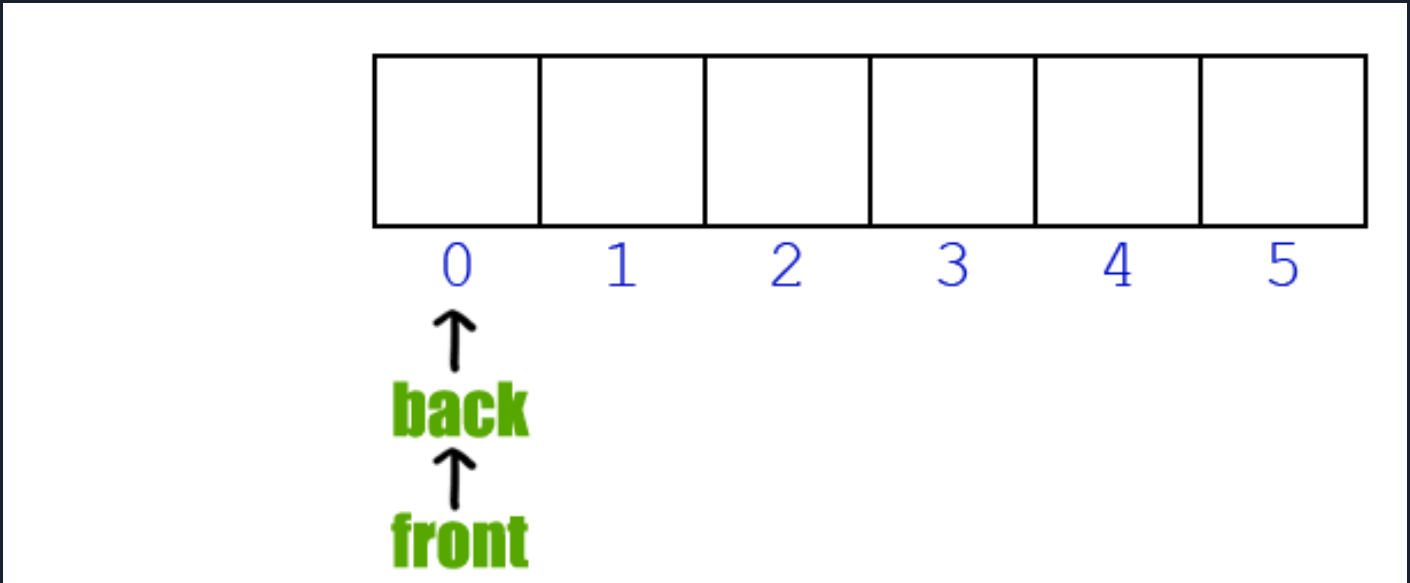
Using Arrays to Implement Queues

- Avoid falling off the end of array by making it circular → once **head/tail** reach past end of array, wrap it back around to front of array
- Queue is full when **head = (tail + 1) % size** (when **tail** is immediately behind the **head**)
- Queue is empty when **head == tail**



Safya

Using Arrays to Implement Queues



Source: http://daltonschool.github.io/CS3A/collections/

Safya

Using Arrays to Implement Queues: Example

Size = 5

Head: **0**
Tail: **2**

2. **Enqueue(c)**

Head: **0**
Tail: **3**

3. **Enqueue(d)**

Head: **0**
Tail: **4**

4. **isFull()**

Head: **0**
Tail: **4**
Return: **True**

5. **Dequeue()**

Head: **1**
Tail: **4**

6. **Dequeue()**

Head: **2**
Tail: **4**

7. **Dequeue()**

Head: **3**
Tail: **4**

8. **Dequeue()**

Head: **4**
Tail: **4**

Safya

Using Arrays to Implement Queues:
Pseudocode & Time Complexity Analysis

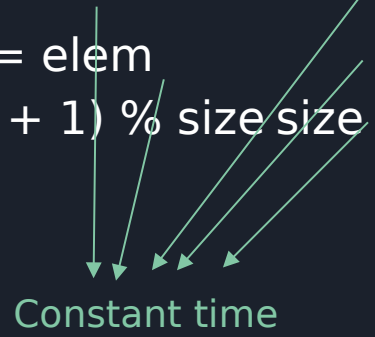
```
class ArrayQueue:
    Object data[]
    int head
    int tail
    int size

    Constructor()
```

```
void Enqueue(Object elem)
    data[tail] = elem
    tail = (tail + 1) % size size
```

```
Object Dequeue()
    item = data[head]
    head = (head + 1) %

    return item
```



Total # of Basic Operations: f(n) = C’ (some constant time)

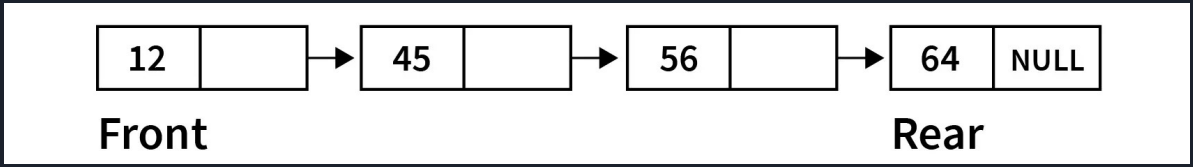
Therefore, queue insertion and deletion operations have constant time complexity in the worst case. ⇒ **O(1)**

Safya

Implementing Queue using Linked List

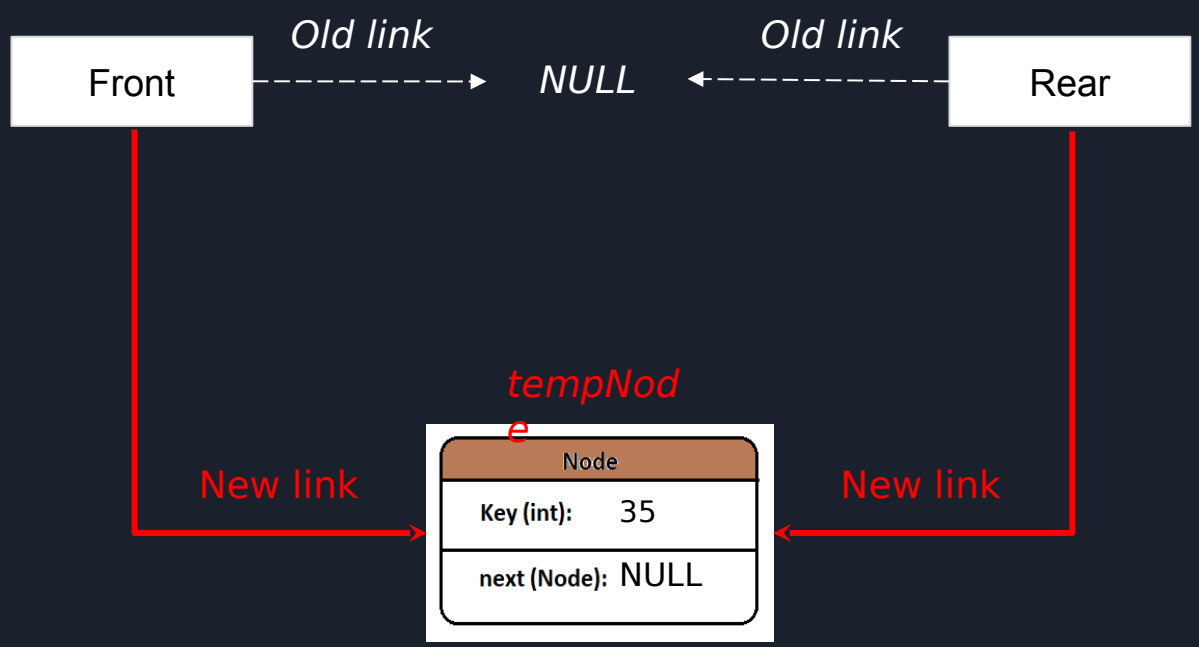
Implementing Queue using Linked List

- The benefit of implementing queue using linked list over arrays is that it allows to grow the queue as per the requirements,i.e, memory can be allocated dynamically.



Ayush

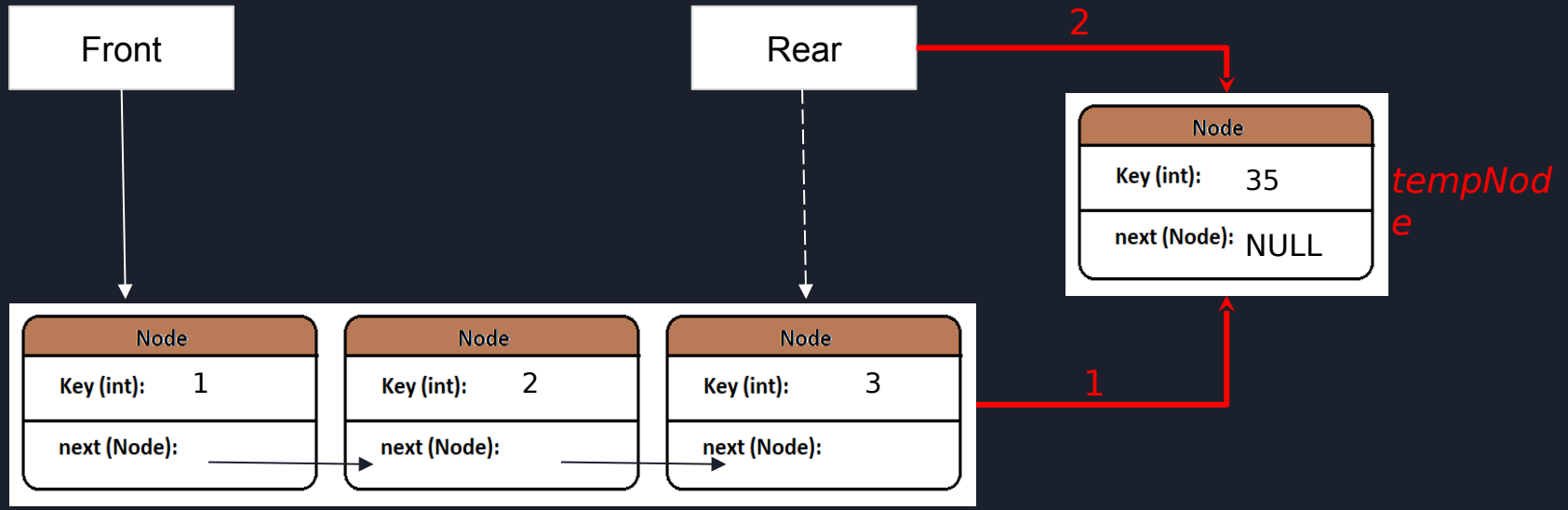
Enqueue operation in action(*empty queue*)




```
this.front = this.rear = tempNode;
```

Ayush

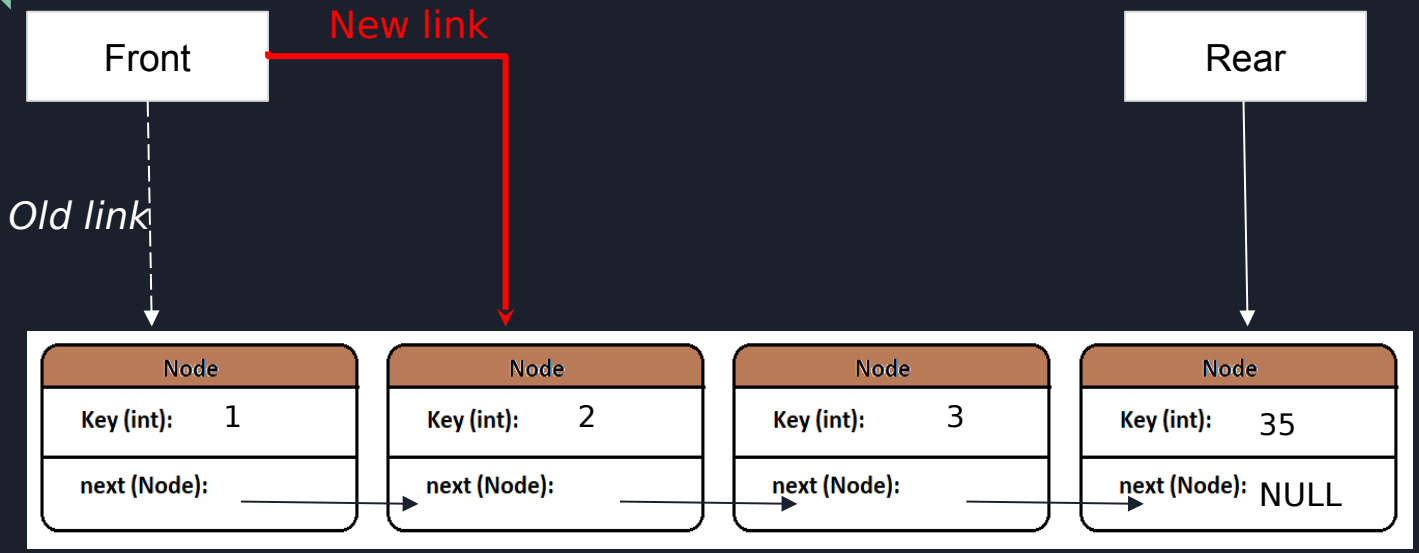
Enqueue operation in action(*non-empty queue*)




```
    this.rear.next = tempNode;
    this.rear = tempNode;
```

Ayush

Dequeue operation in action



Deleted node

```
this.front = this.front.next;
```

Ayush

Thank you!