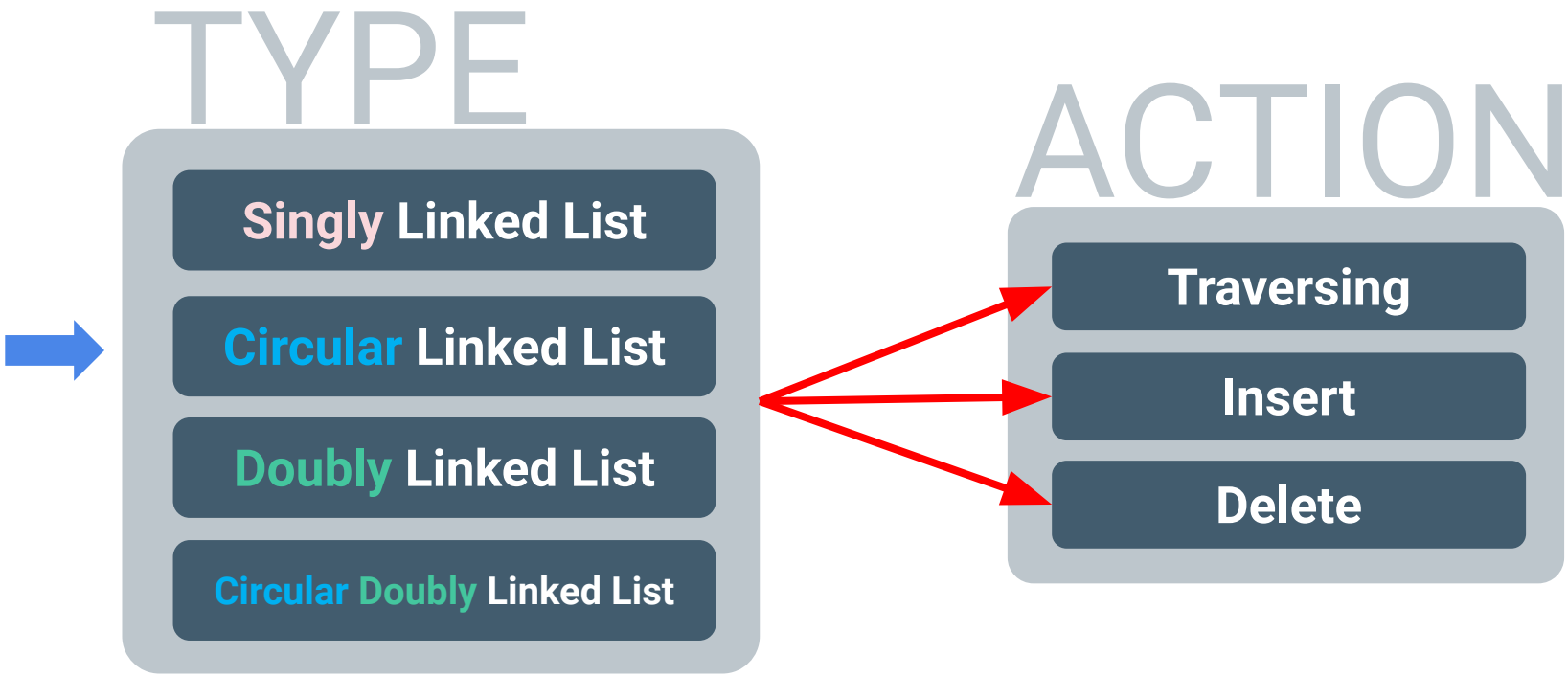
Chapter 5: Linked Lists

Dr. Sirasit Lochanachit

Today's Outline

1. Circular Linked Lists
    - Queue Implementation
2. Doubly Linked Lists
    - Traversing
    - Insert a node
    - Delete a node
3. Circular Doubly Linked Lists

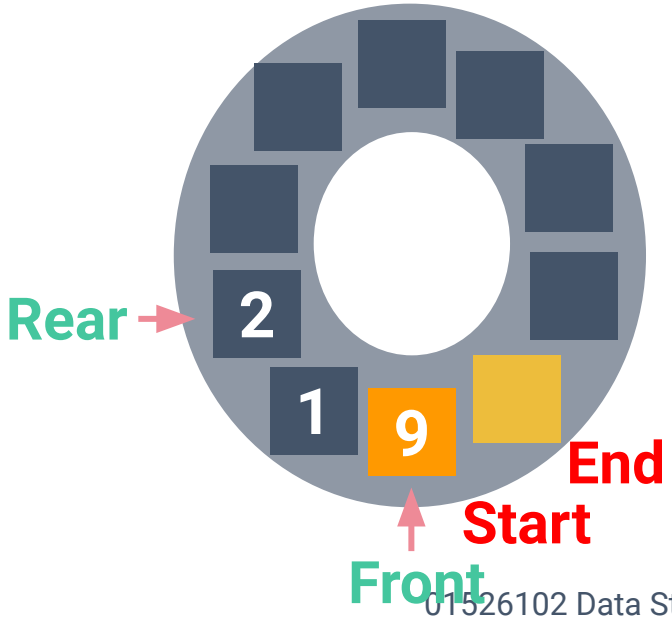
Linked Lists



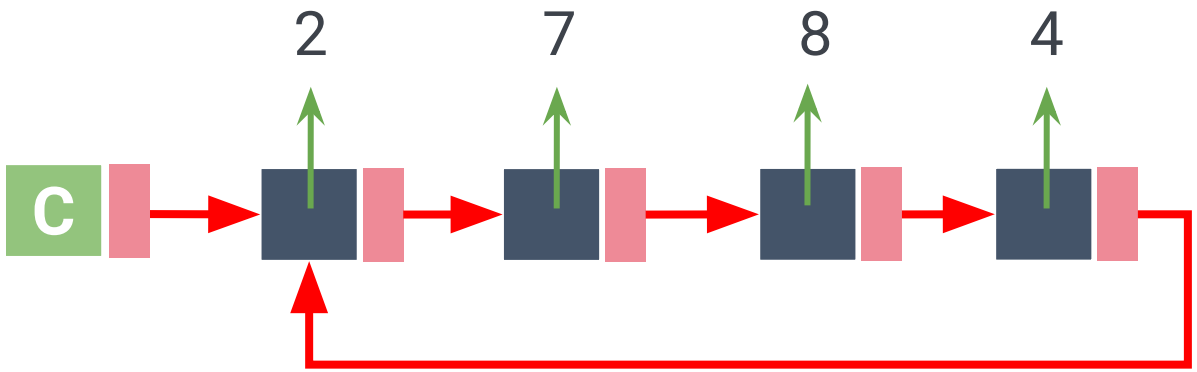
Circular Queue

Circular is artificial for array

- Modular arithmetic (%)



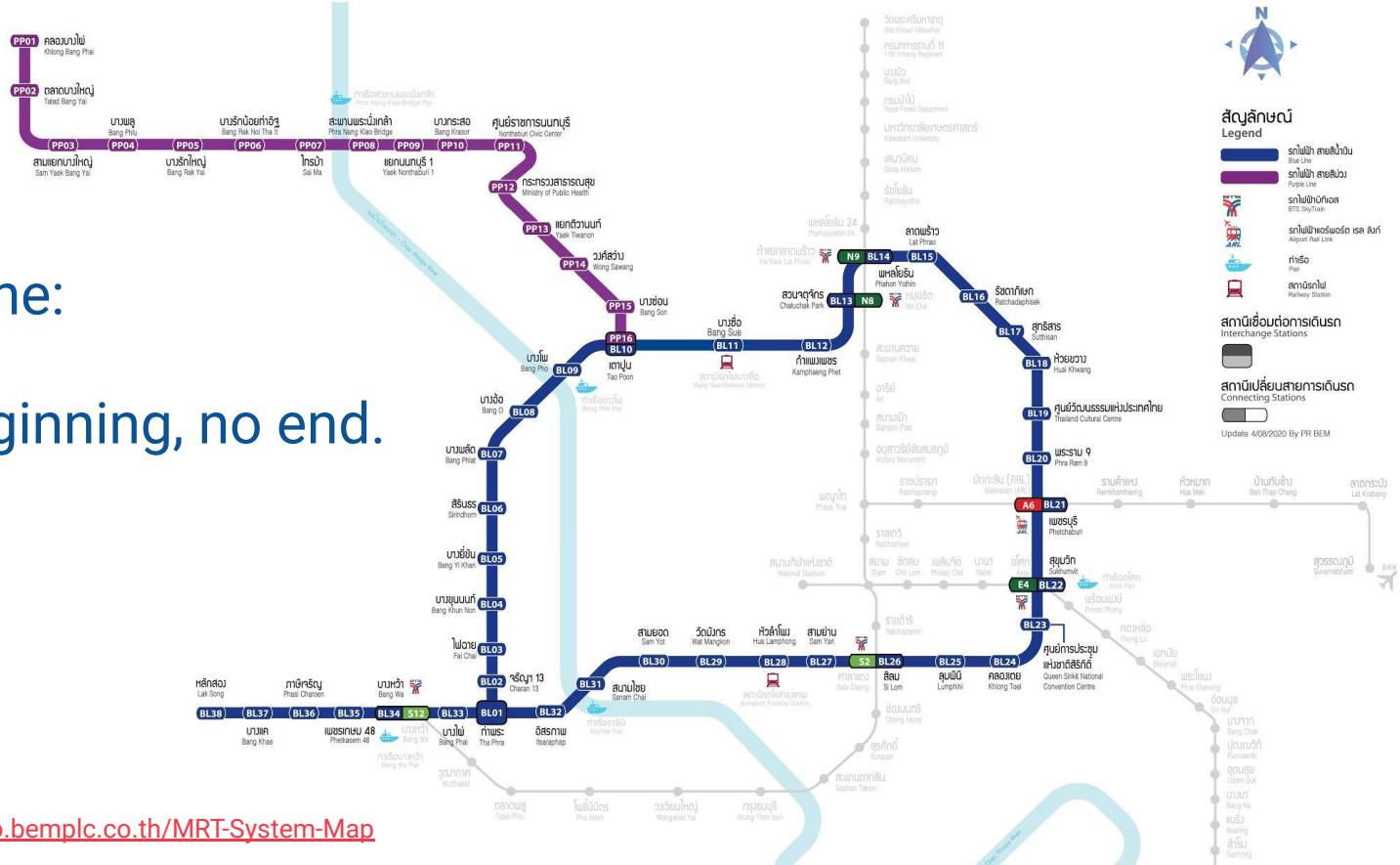
Circular Linked Lists



Circular Linked Lists

Blue line:

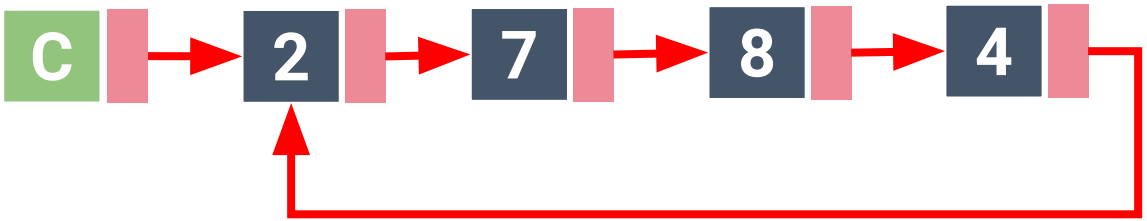
No beginning, no end.



Ref - https://metro.bemplc.co.th/MRT-System-Map

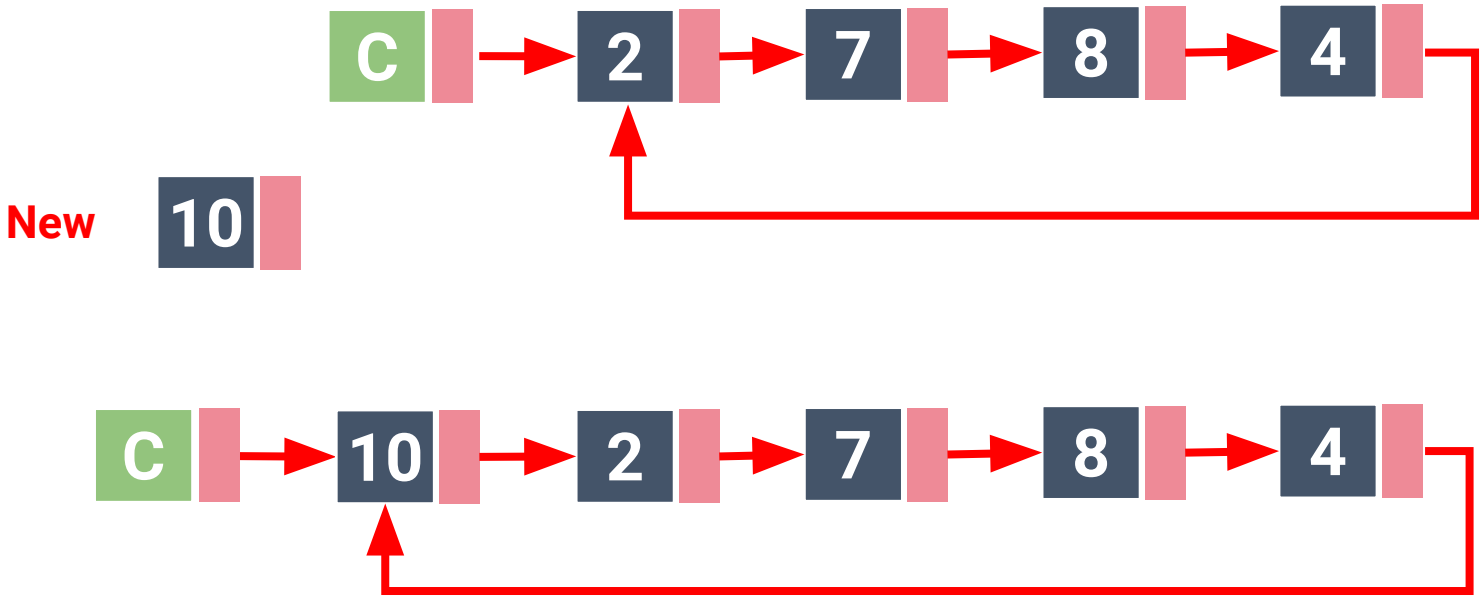
Circular Linked Lists

Address/ Byte#	Value
6000	4
6001	6002
6002	2
6003	6008
6004	8
6005	6012
6006	
6007	
6008	7
6009	6004
6010	
6011	
6012	4
6013	6002



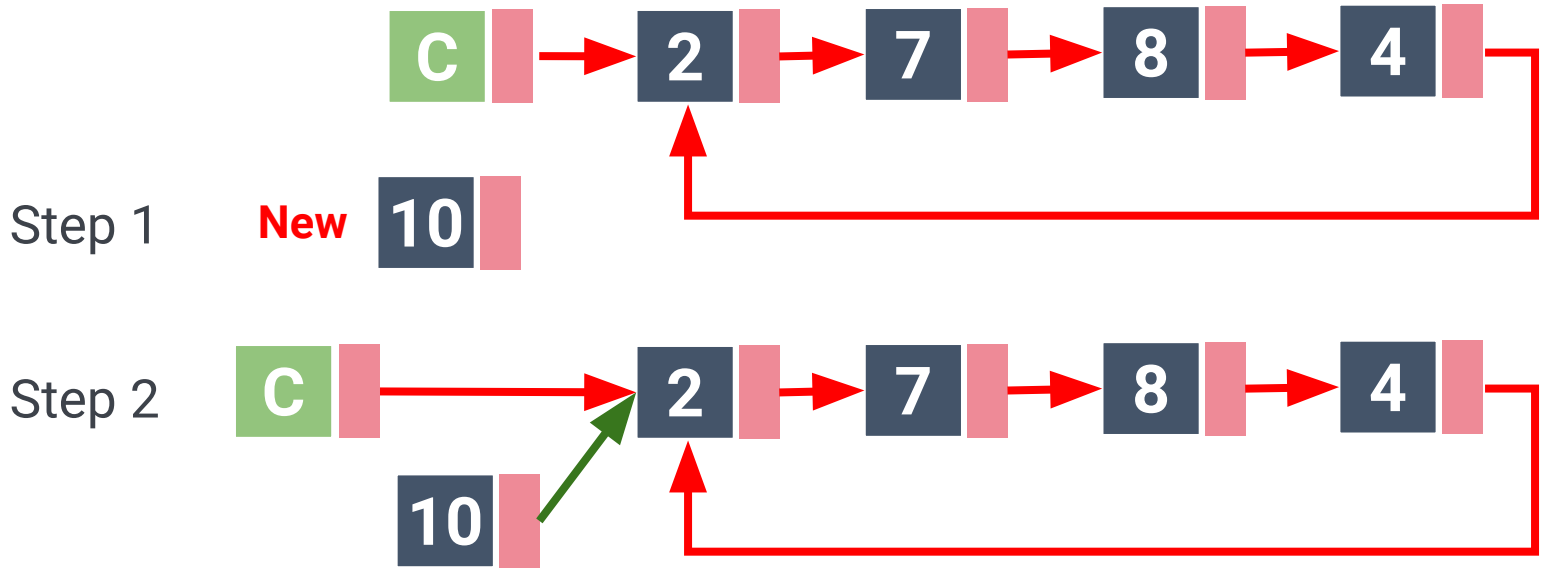
Circular Linked Lists

Insert at the head of the list



Circular Linked Lists

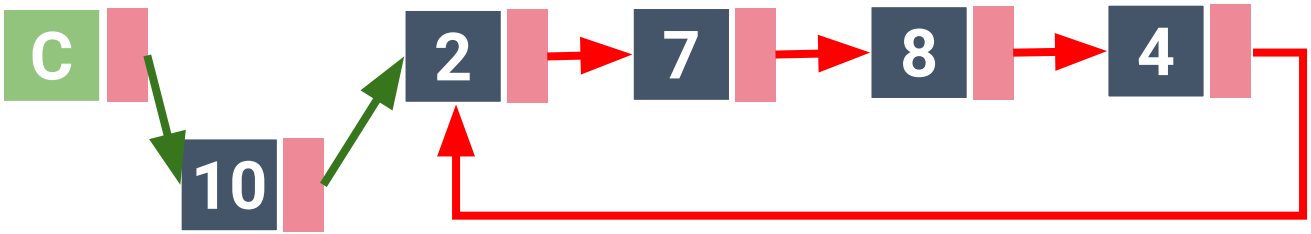
Insert at the head of the list



Circular Linked Lists

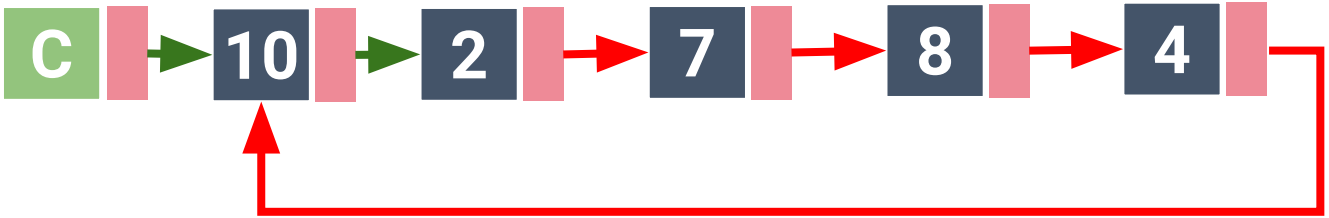
Insert at the head of the list

Step 3



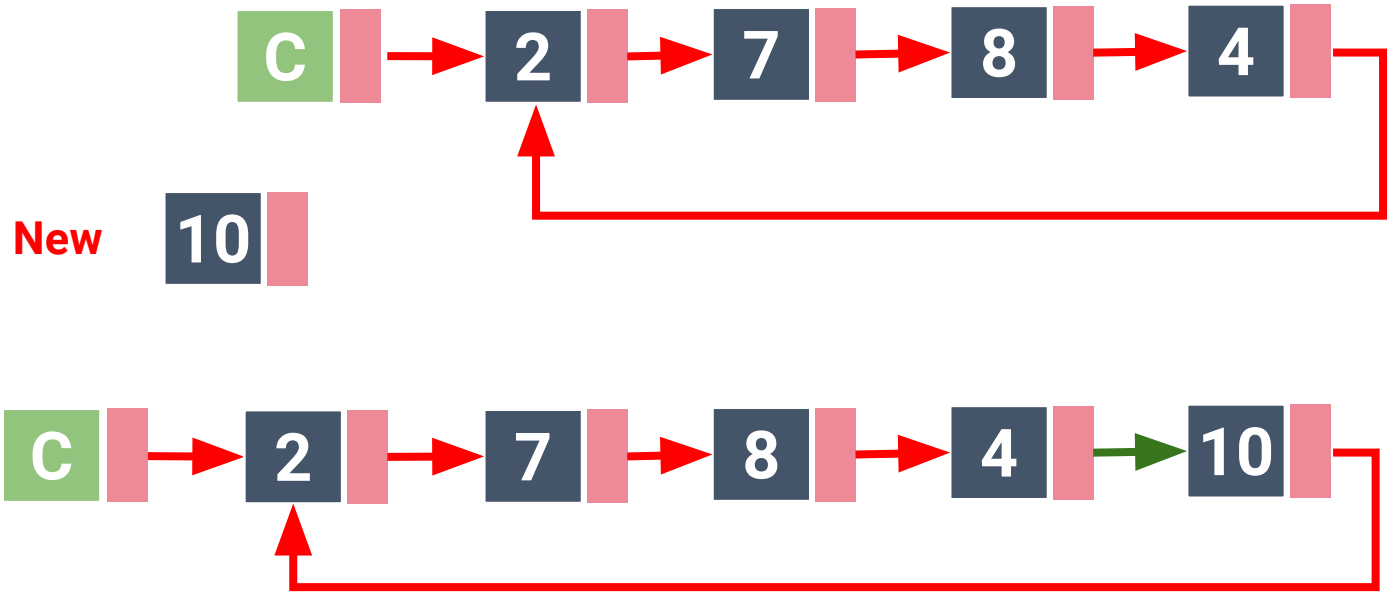
count += 1

Step 4&5



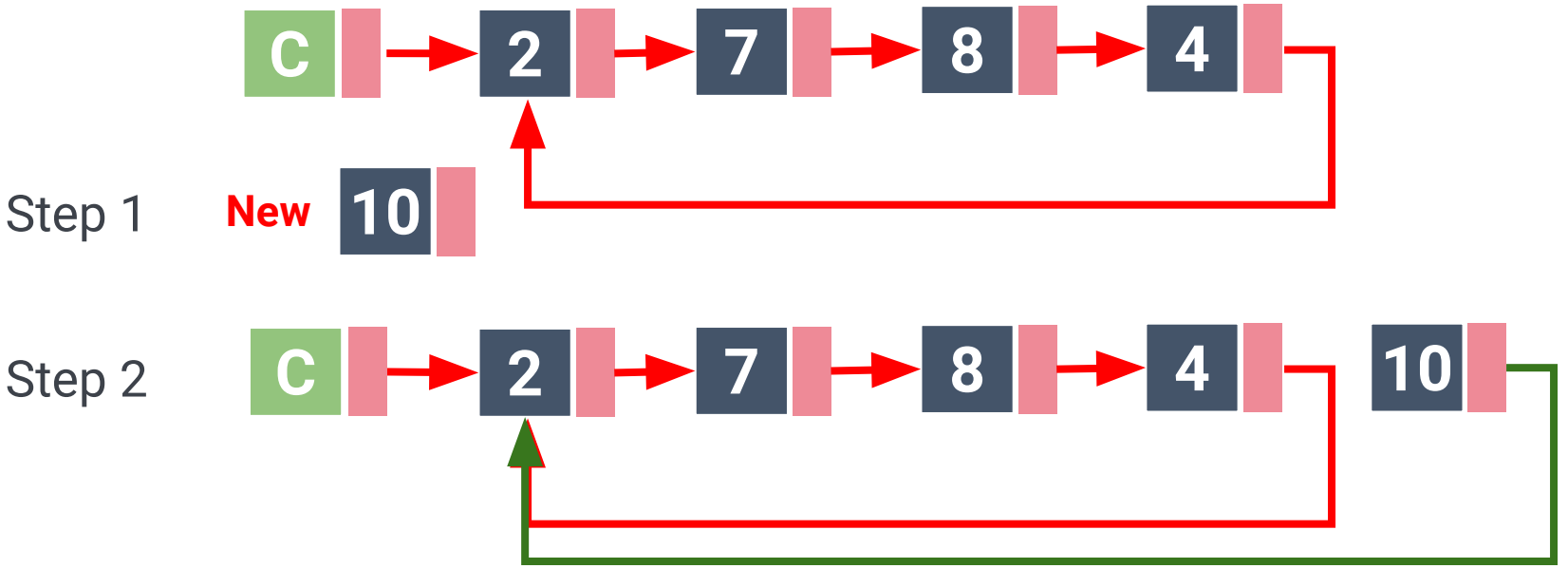
Circular Linked Lists

Insert at the tail of the list



Circular Linked Lists

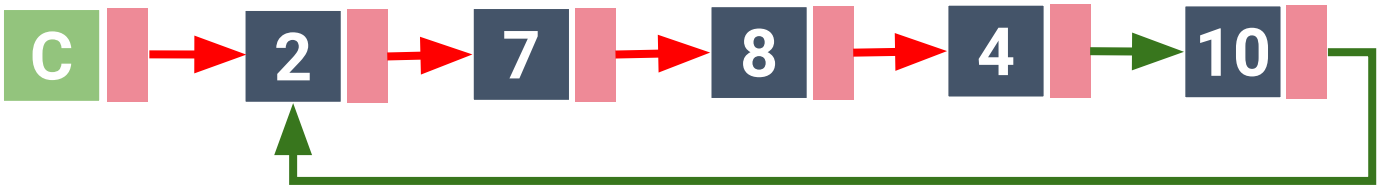
Insert at the tail of the list



Circular Linked Lists

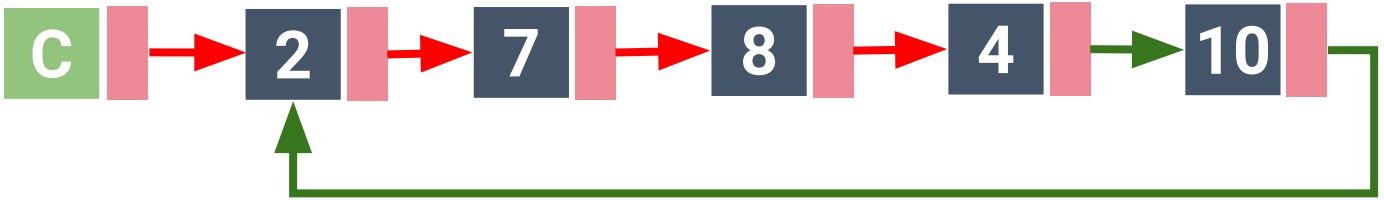
Insert at the tail of the list

Step 3

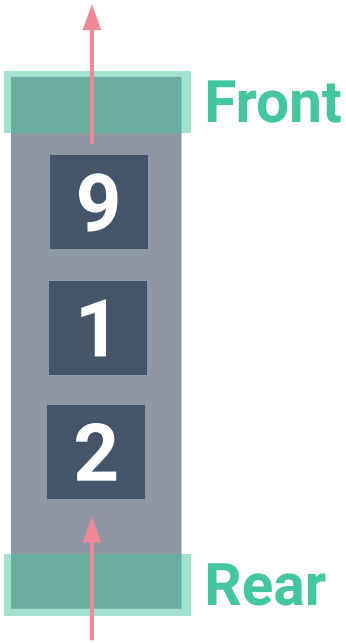


Step 4

count += 1



Circular Linked Lists: Queues



How to Implement a Queue?

Array!!

and

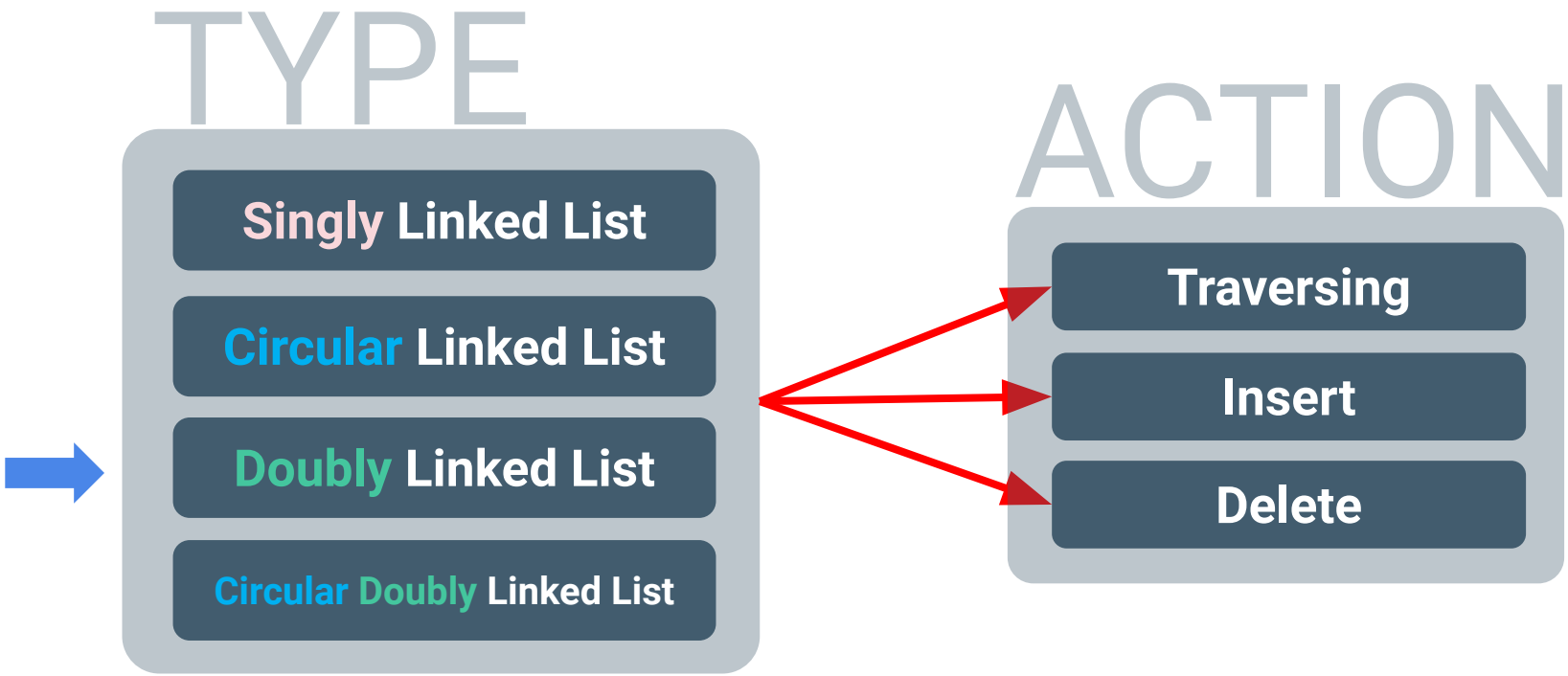
Linked Lists!!

- Singly Linked Lists
- **Circular Linked Lists**

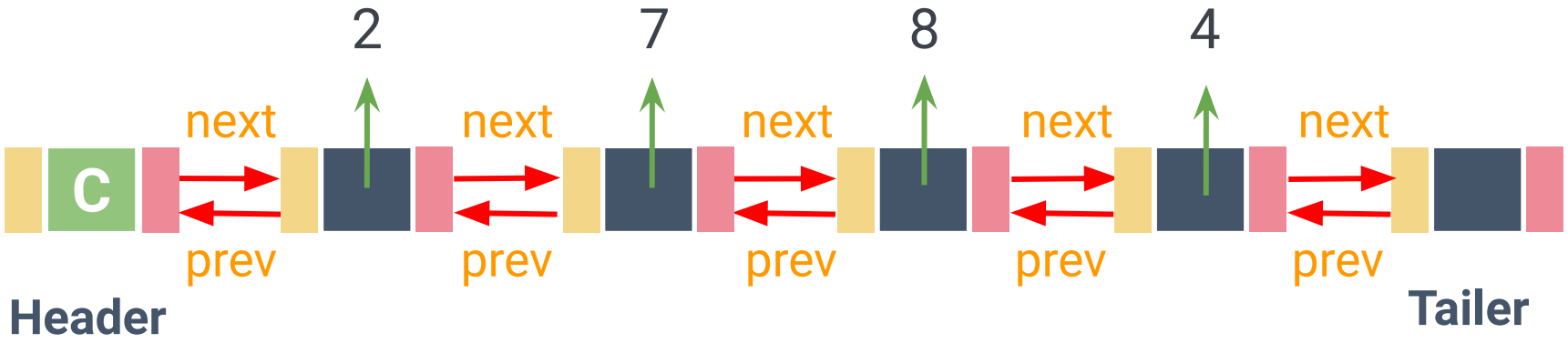
Asymptotic Performance

	Queue		Circular Queue	
Operation	**Running Time - Array**	**Running Time - Singly Linked List**	**Running Time - Array**	**Running Time - Circular Linked List**
Q.enqueue(e)	O(n)	$O(1)$	$O(1)$	
Q.dequeue()	O(n)	$O(1)$	$O(1)$	
Q.first()	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Q.is_empty()	$O(1)$	$O(1)$	$O(1)$	$O(1)$
len(Q)	$O(1)$	$O(1)$	$O(1)$	$O(1)$

Linked Lists



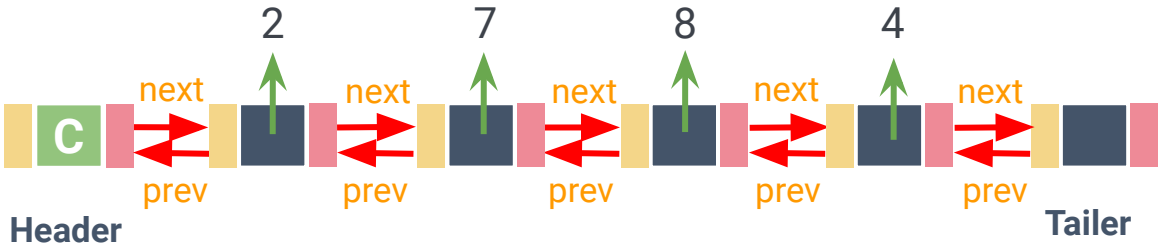
Doubly Linked Lists



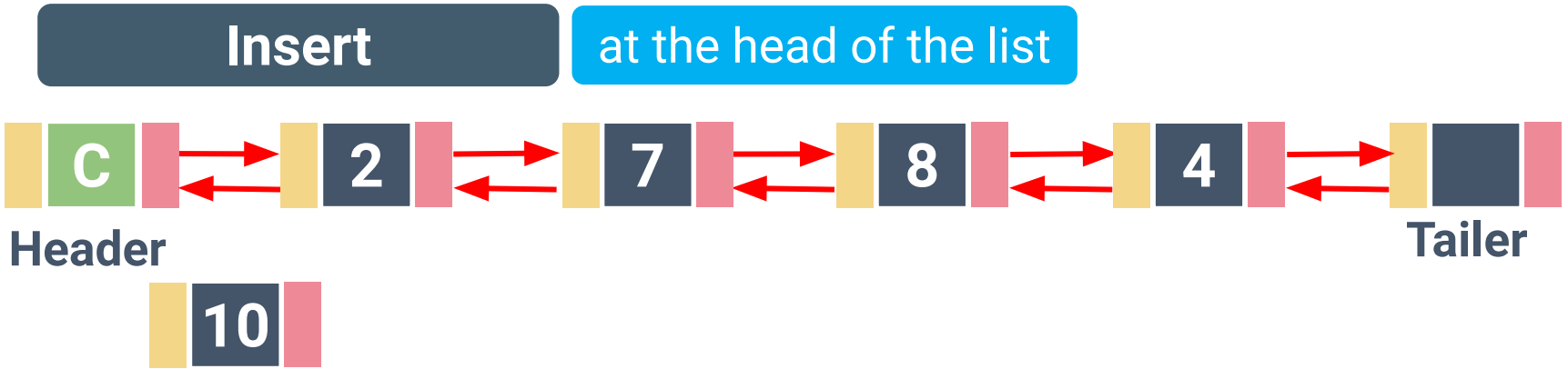
Doubly Linked Lists

Address/Byte#	Value	
6000	4	
6001	6003	next **Header**
6002		
6003	2	
6004	6009	next
6005	6000	prev
6006	8	
6007	6013	next
6008	6009	prev
6009	7	
6010	6006	next
6011	6003	prev
6012		
6013	4	
6014	6016	next
6015	6006	prev
6016	T	**Tailer**
6017	6013	prev

Suppose that it takes 1 byte to store an integer.



Doubly Linked Lists



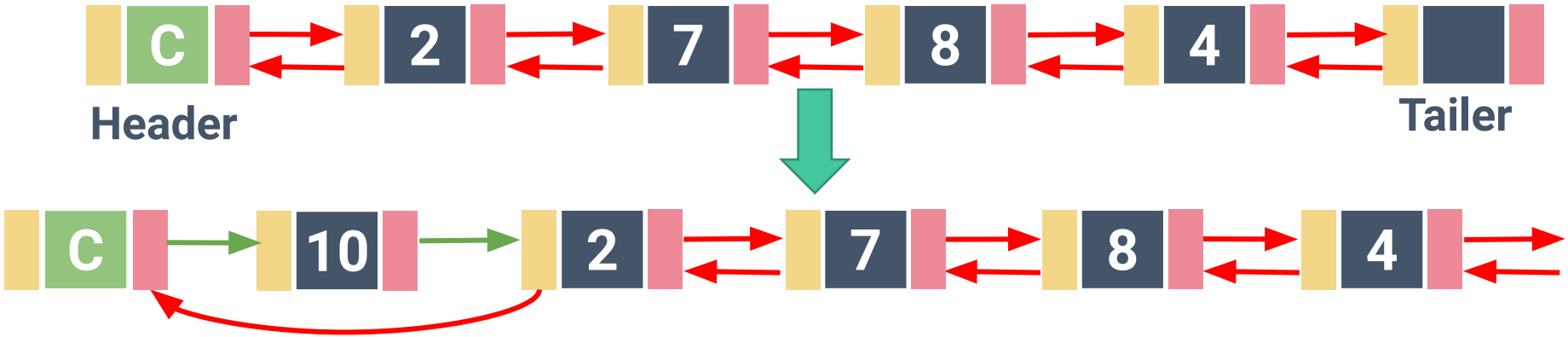
Step 1: Create a new node storing reference to an element

Step 2: Set a new node's next pointer to the current head node

Step 3: Set the list's head to refer to the new node

Doubly Linked Lists

Insert at the head of the list

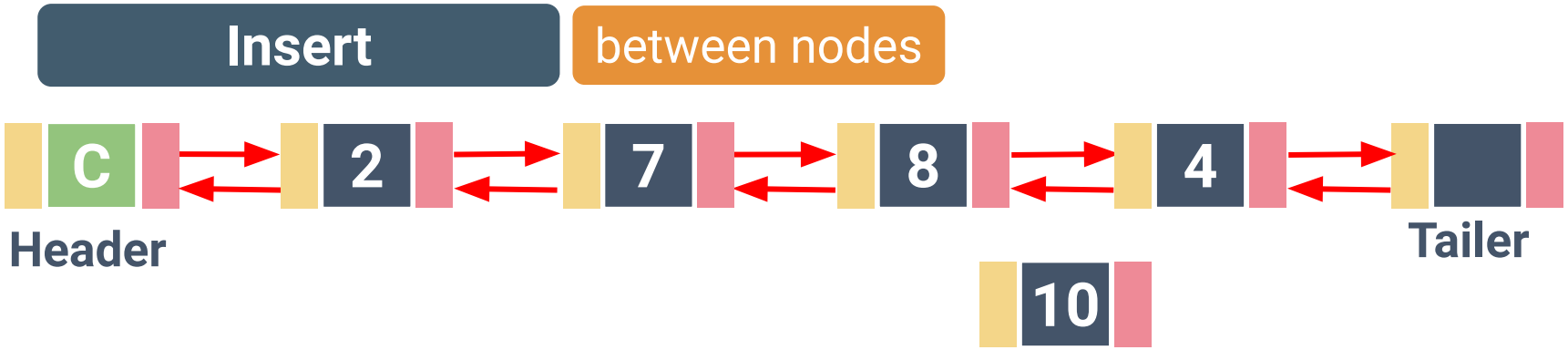


Step 4:

Step 5:

Step 6: Increment the node count

Doubly Linked Lists



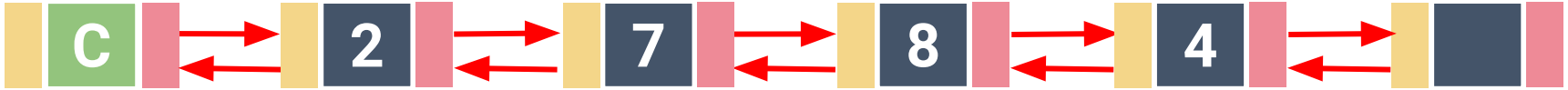
Step 1: Create a new node storing reference to an element

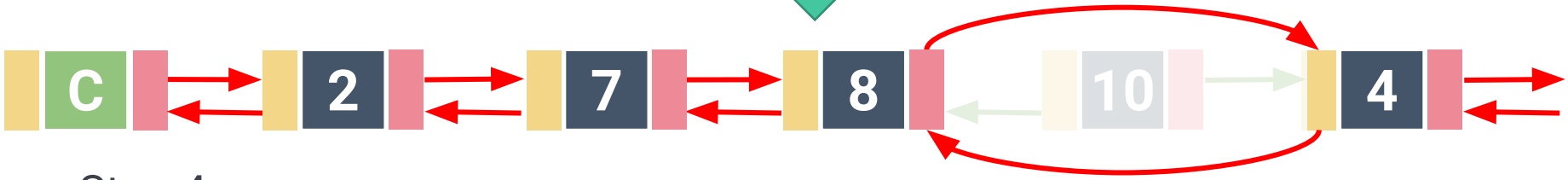
Step 2: Set a new node's next pointer to the next node

Step 3:

Doubly Linked Lists

Insert between nodes





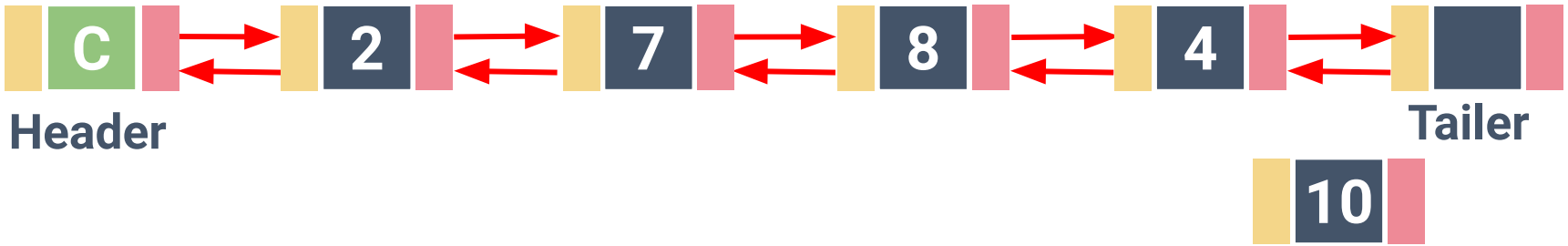
Step 4:

Step 5:

Step 6: Increment the node count

Doubly Linked Lists

Insert at the tail of the list



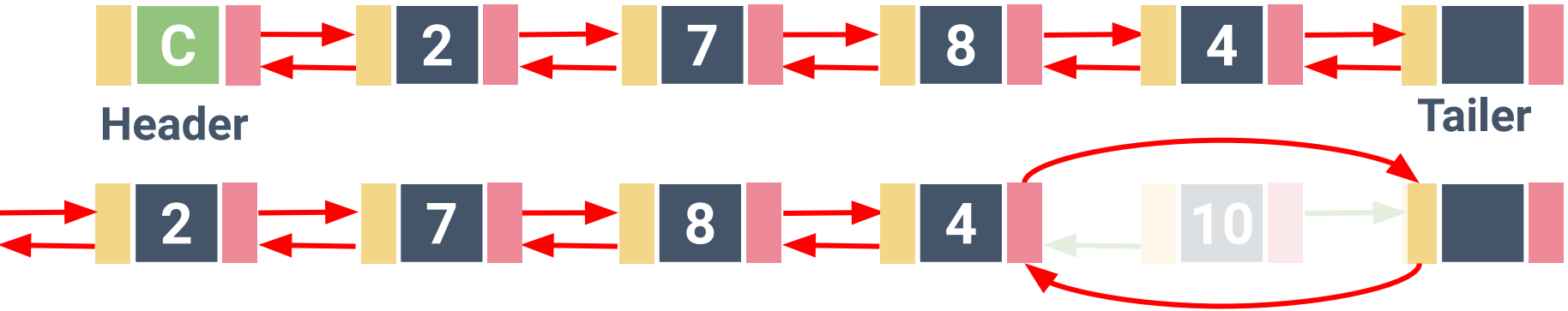
Step 1: Create a new node storing reference to an element

Step 2: Set a new node's next pointer to the tailer node

Step 3: Set the new node's previous pointer to the previous node.

Doubly Linked Lists

Insert at the tail of the list



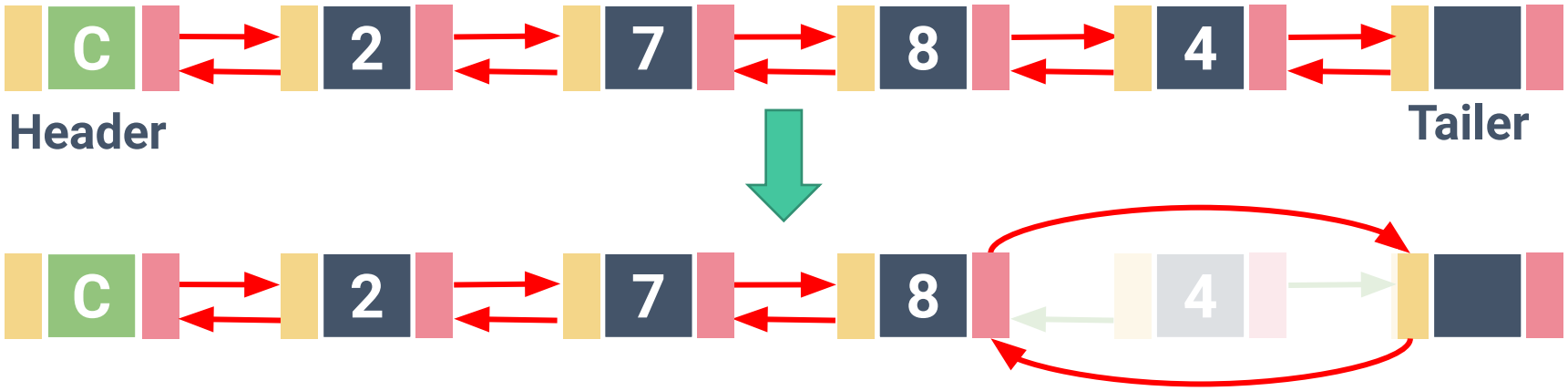
Step 4:

Step 5: Set the previous node's next pointer to the new node.

Step 6: Increment the node count

Doubly Linked Lists

Delete at the tail of the list



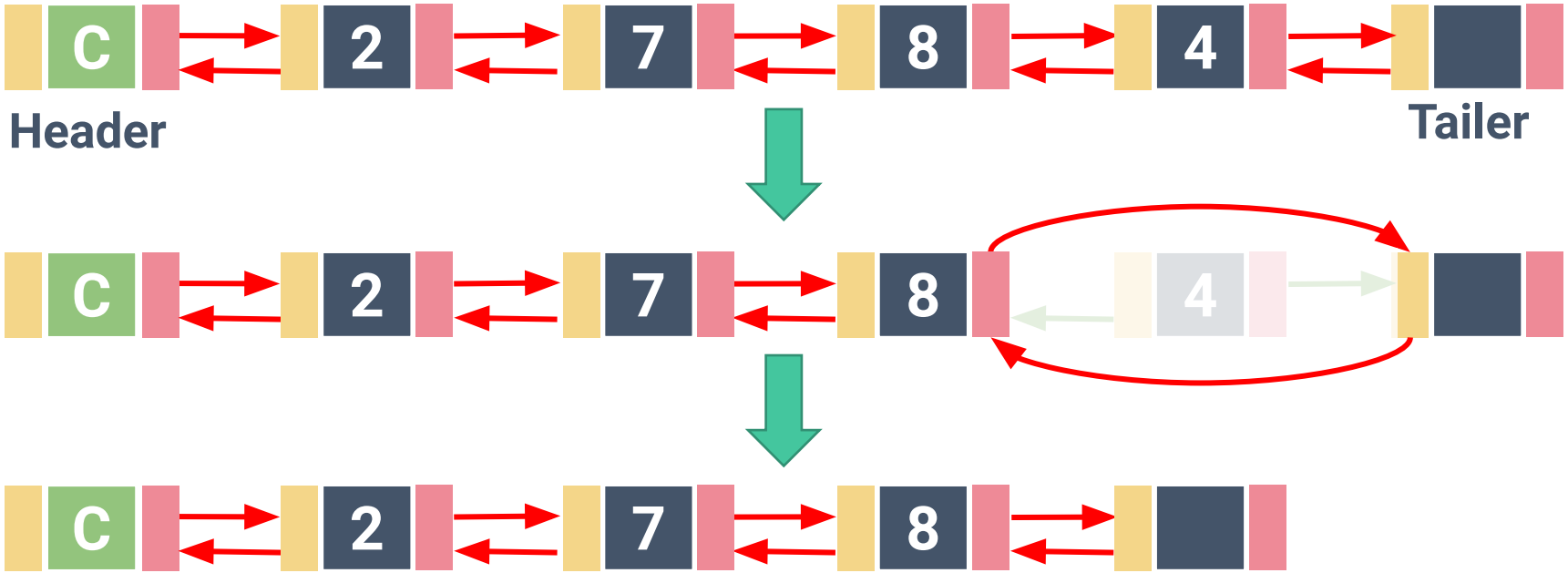
Step 1:

Step 2:

Step 3:

Doubly Linked Lists

Delete at the tail of the list



Asymptotic Performance

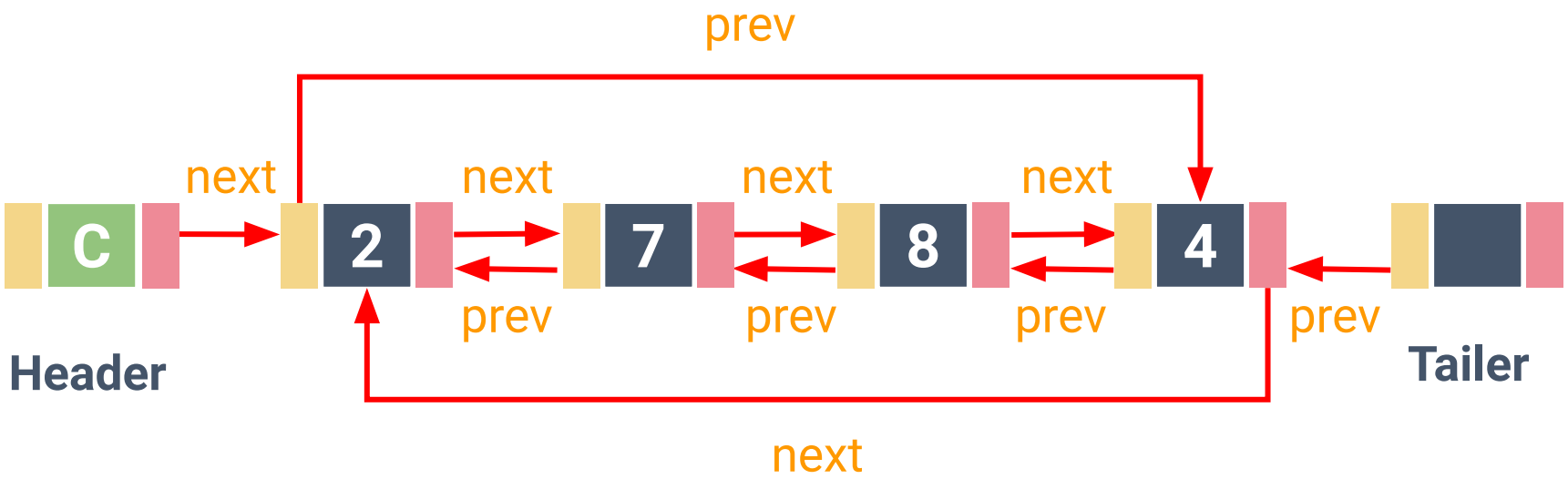
Operation		**Running Time - Singly Linked List**	**Running Time - Doubly Linked List**
Insert	At the head	O(1)	
	Between nodes	O(1)	
	At the tail	O(1)	
Delete	At the head	O(1)	
	Between nodes	O(n)	
	At the tail	O(n)	

Linked Lists

TYPE

- Singly Linked List
- Circular Linked List
- Doubly Linked List
- Circular Doubly Linked List

Circular Doubly Linked Lists



Summary

Linked list properties:

- Each node contains an element and a pointer(s) to the next node (and previous node).
- **Sequential access** only: nodes are read from the beginning.
  - Not convenient to have an index, unlike array-based sequences.
- No pre-allocated fixed size of memory, resizeable.
- Insertion and deletion operations are more efficient compared to array.
  - Take $O(1)$ - constant time to add and remove elements at any part in linked lists.

Summary

Linked list's limitations:

- Accessing the data/node in lists takes linear time - *O*(n)
  - To find the item or node at certain location, linked list has to start from the first node and traversing until the target is found.
  - Unable to perform binary search.
- Use extra storage than the array to keep pointers/references.
  - Impractical for storing small data such as characters.