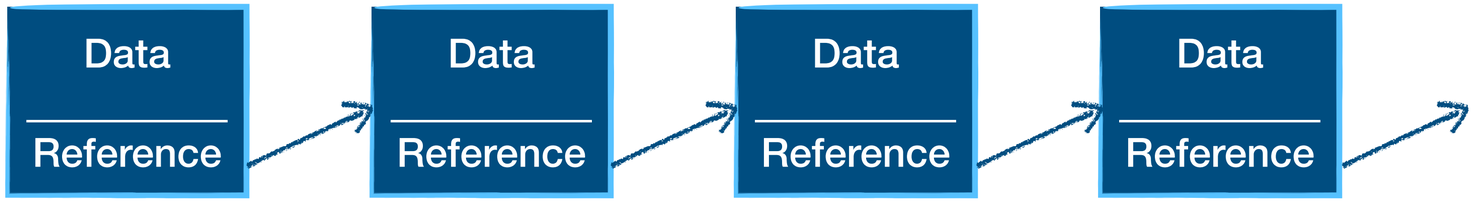
Algorithms and Data Structures

Introduction to Data Structures

Part III: Linked List

What is a Linked List?

- Python's List is NOT Linked List.
- A Linked List is a sequential list of nodes that hold data which point to other nodes also containing data.
  - Linked List is a list of Nodes;
  - Each Node has two components:
    - Data: holds value.
    - Reference: holds the pointer to other Node(s).

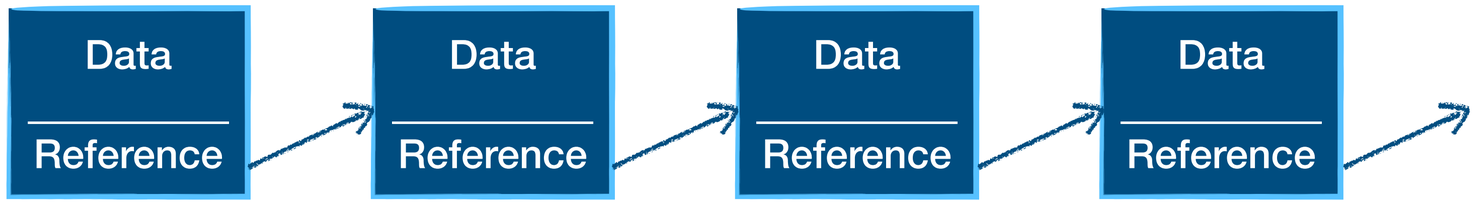


Where a Linked List is used?

- Linked Lists are widely used in many Lists, Stack, and Queue implementations (will talk more later on)
- Great for creating circular lists
- Naturally for lots of real world objects such as trains
- Underneath structure for Block Chain and BitCoin.
- Often used in the implementation of the adjacency lists for graphs (will discuss in detail when we learn graphs)

Terminology About Nodes

- Nodes
  - Data / Value
  - Reference / Pointer
- Special Nodes:
  - Head
  - Tail



Terminology About Linked Lists

- Depends on the number of references and the node the references point to, we have:
  - Only one reference to next node: Singly Linked Lists
  - One reference points to next, the other one points to the previous: Doubly Linked Lists
  - Tail node's reference points to the head node: Circularly Linked Lists

Comparison

- Depends on the number of references and the node the references point to, we have:
  - Only one reference to next node: Singly Linked Lists
  - One reference points to next, the other one points to the previous: Doubly Linked Lists
  - Tail node's reference points to the head node: Circularly Linked Lists

Ops\List Type	Singly	Doubly	Circularly
Access	O(n)	O(n)	O(n)
Search	O(n)	O(n)	O(n)
Append	O(1)	O(1)	O(1)
Prepend	O(1)	O(1)	O(1)
InsertAt	O(n)	O(n)	O(n)
Remove Head	O(1)	O(1)	O(1)
Remove Tail	O(n)	O(1)	O(n)
Remove At	O(n)	O(n)	O(n)

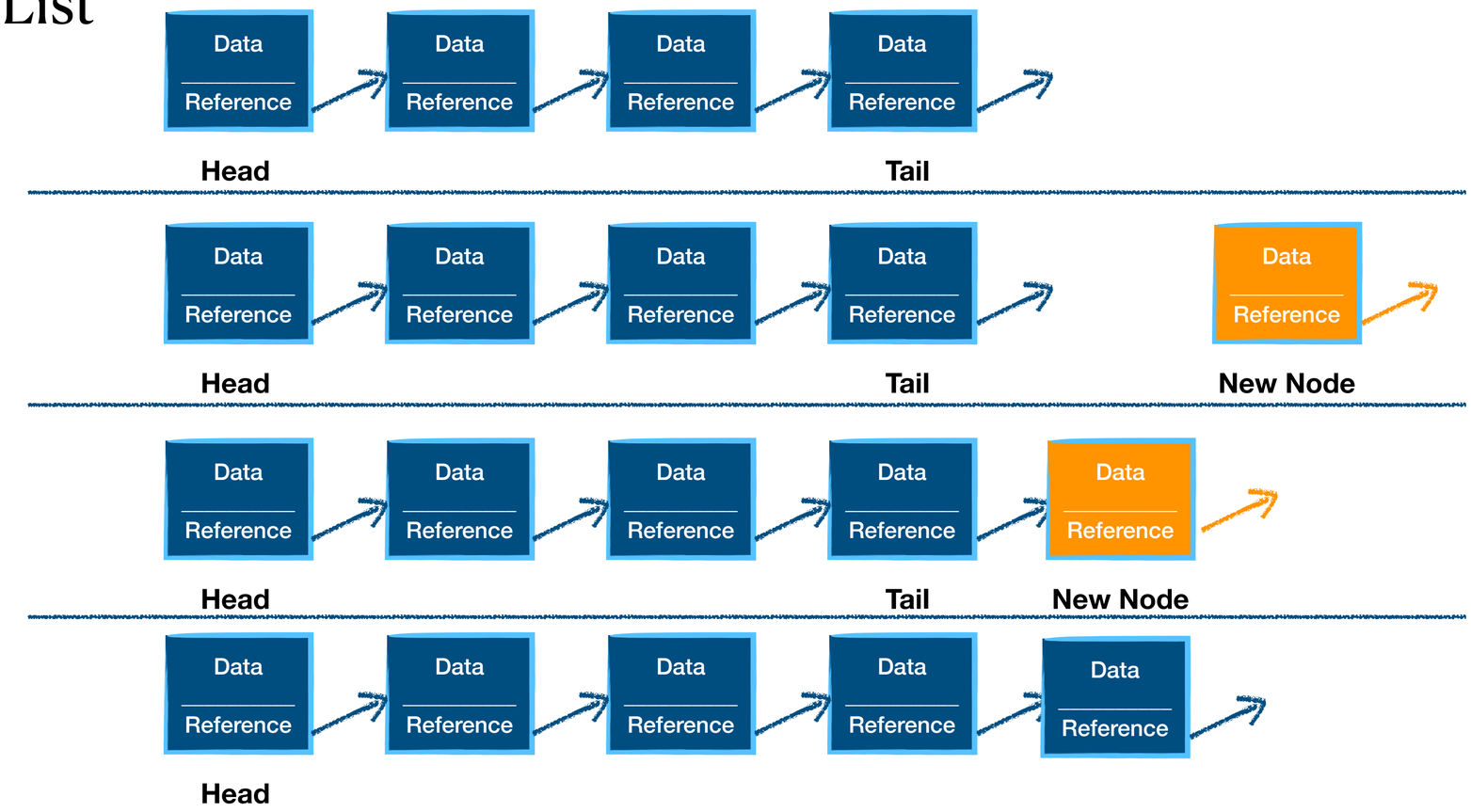
Some Other Pros and Cons

- Depends on the number of references and the node the references point to, we have:
  - Only one reference to next node: Singly Linked Lists
  - One reference points to next, the other one points to the previous: Doubly Linked Lists
  - Tail node's reference points to the head node: Circularly Linked Lists

Crite\List Type	Singly	Doubly	Circularly
Space	Pro	Con	Pro
Search	Con	Pro	Con
Implementation	Pro	Con	Pro

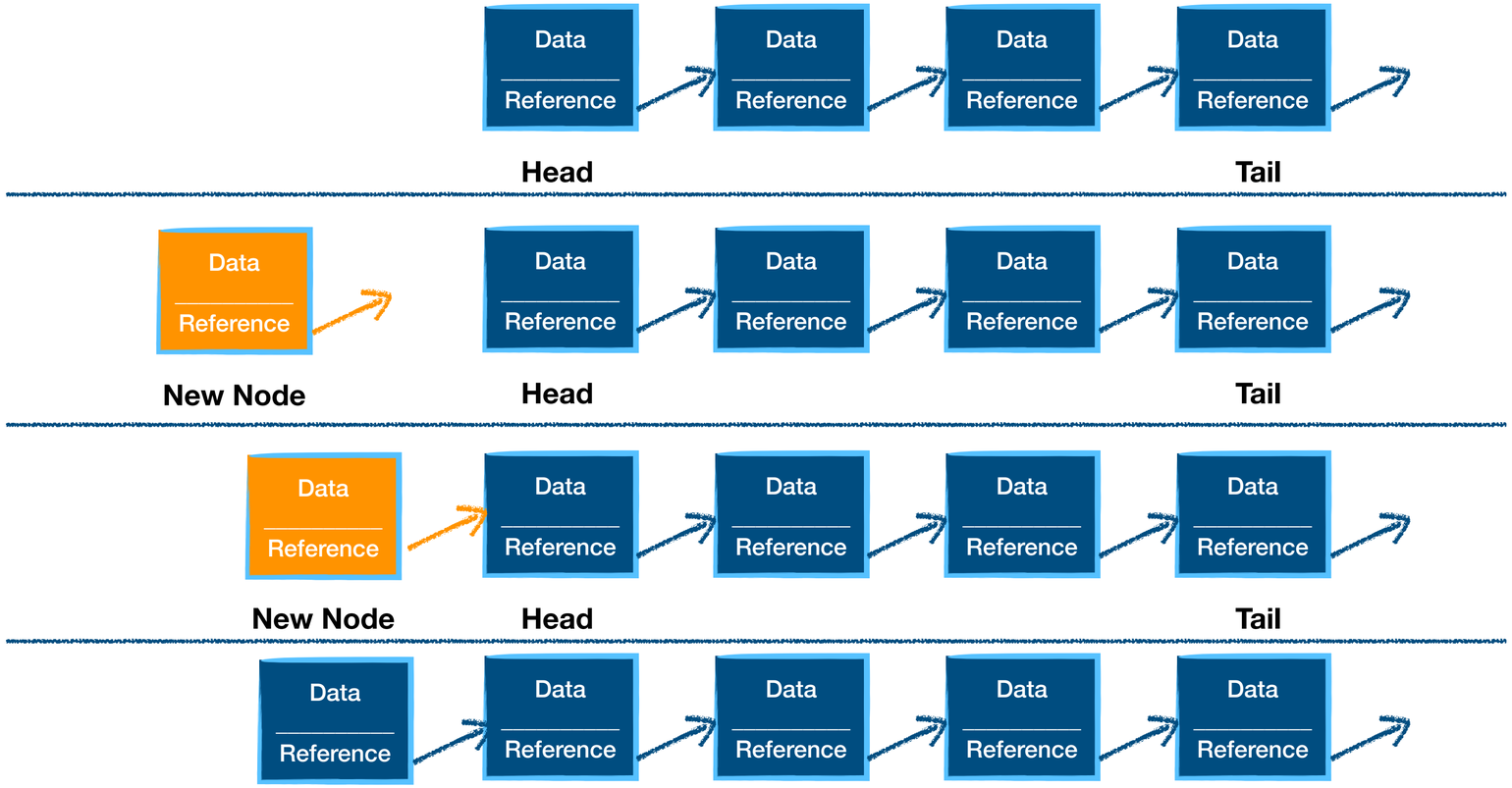
Insertion

- Append To a Singly Linked List



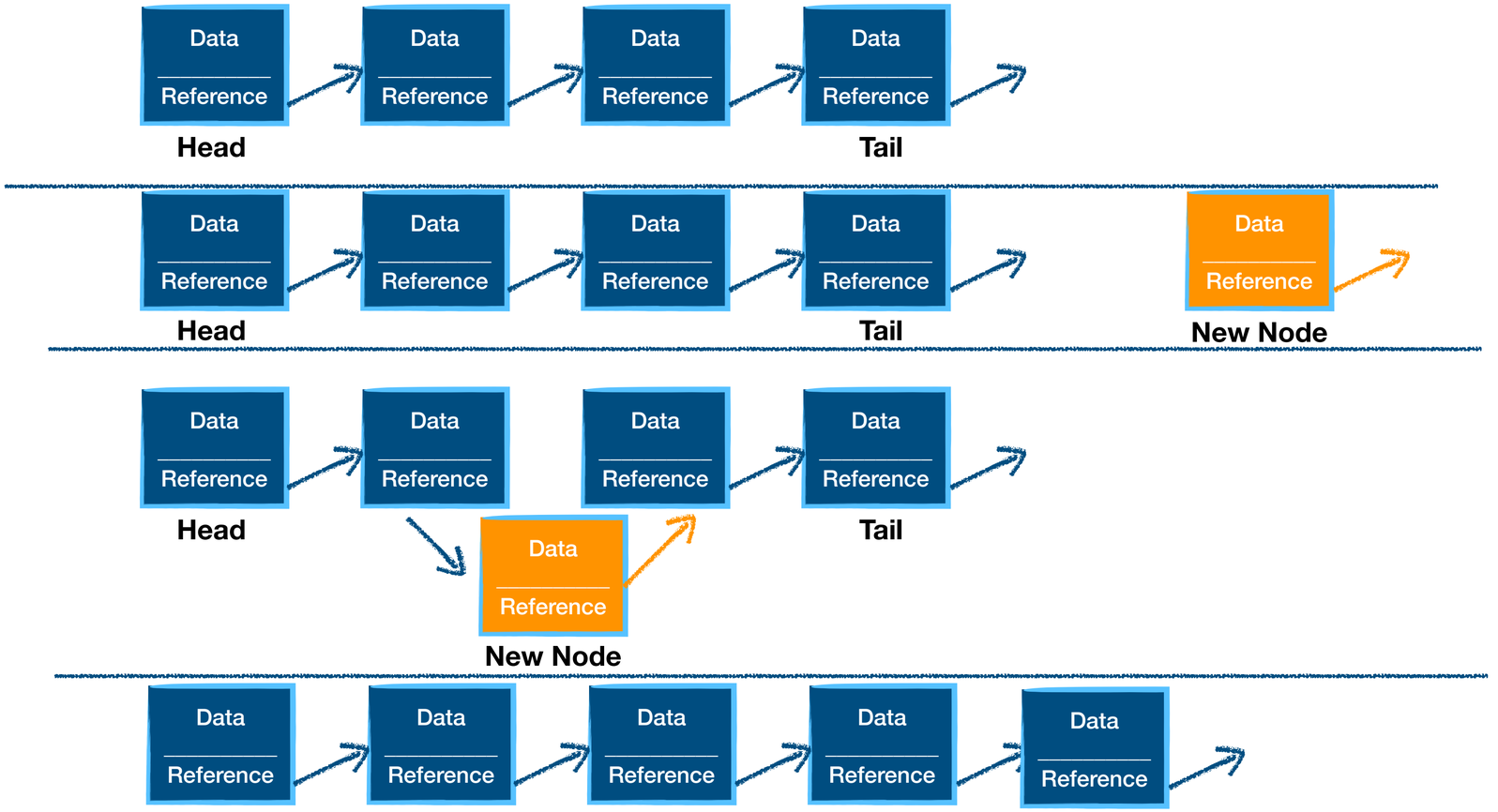
Insertion

- Prepend To a Singly Linked List



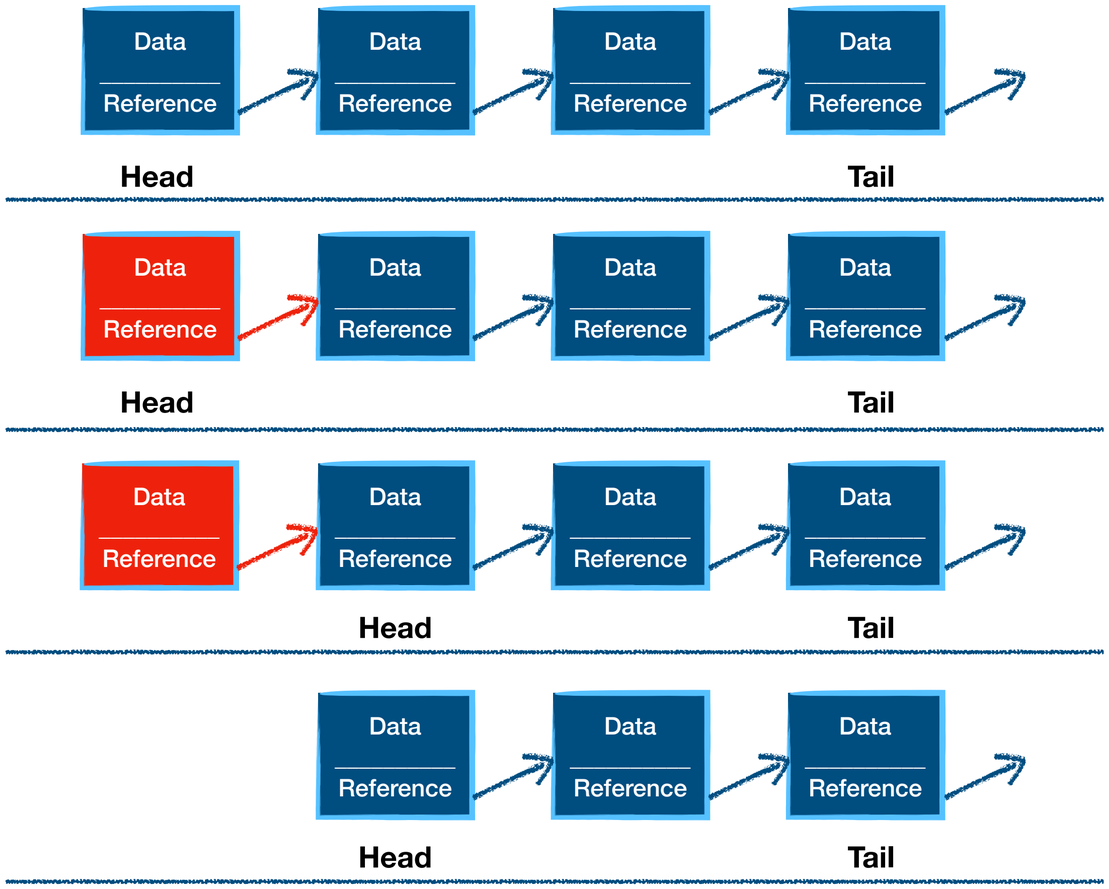
Insertion

- InsertAt(2, Data) to a Singly Linked List



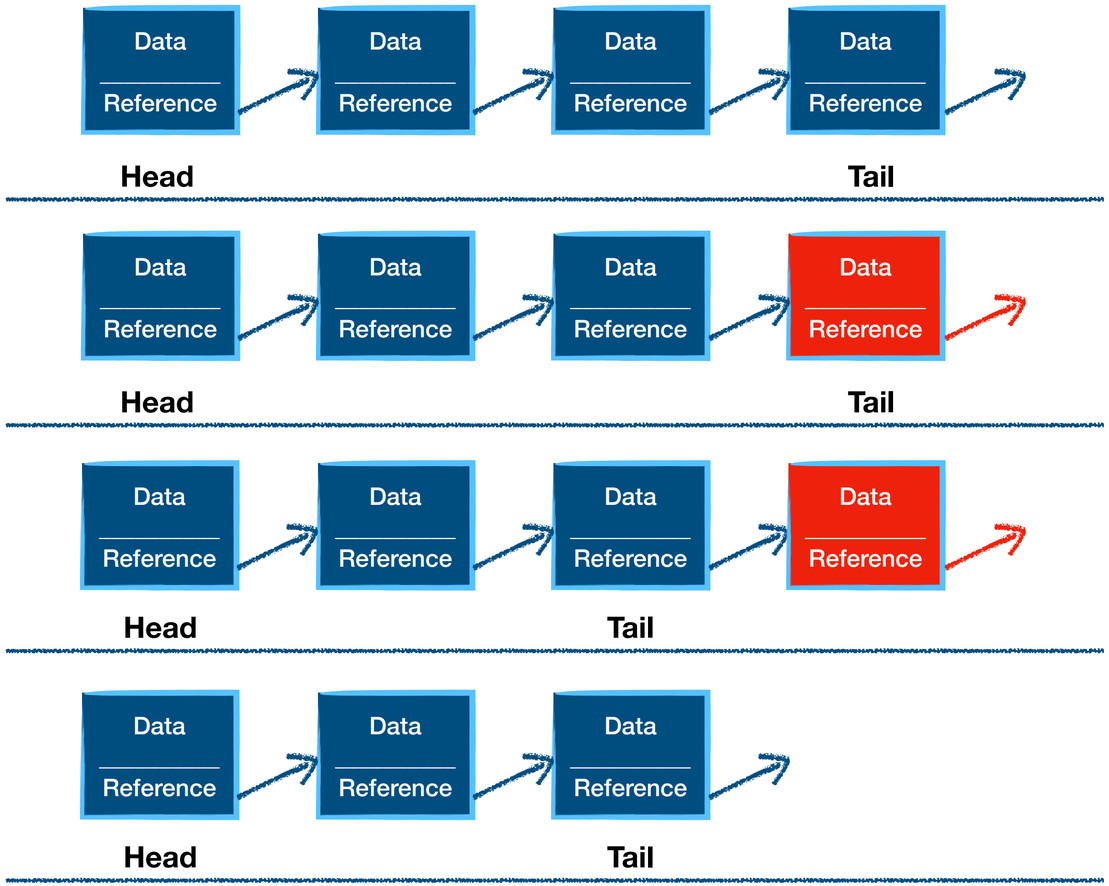
Deletion

- Delete Head of a Singly Linked List



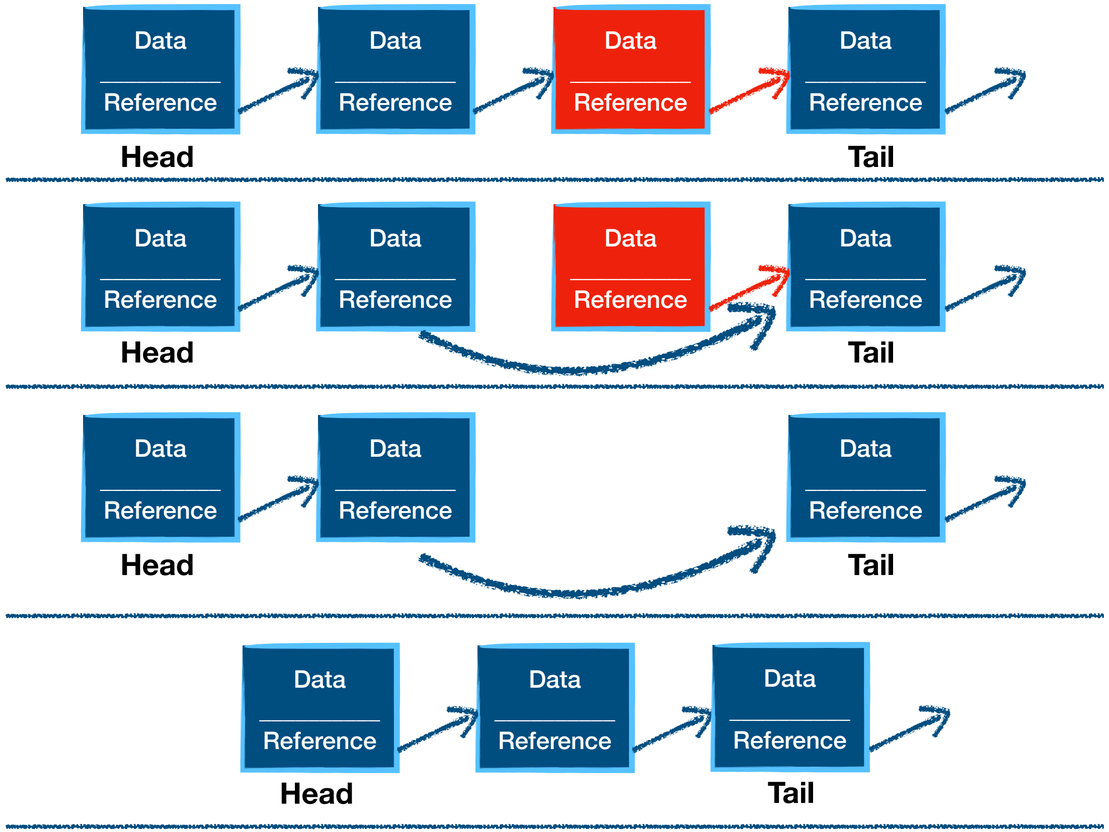
Deletion

- Delete Tail of a Singly Linked List



Insertion

- RemoveAt(2) to a Singly Linked List



Exercise

- Given an empty Integer **Singly** Linked List, draw / describe following operations:
  - append(5);
  - append(1);
  - prepend(0);
  - prepend(2);
  - addAt(1, 2);
  - removeAt(1);
  - remove(5);
  - clear();

Exercise

- Given an empty Integer **Doubly** Linked List, draw / describe following operations:
  - append(5);
  - append(1);
  - prepend(0);
  - prepend(2);
  - addAt(1, 2);
  - removeAt(1);
  - remove(5);
  - clear();

Arrays VS LinkedLists

- Arrays have quick access to any elements (because most computers use caches that favor reading from sequential addresses) while LinkedLists have linear time to access elements (and caches take longer time to read from scattered addresses).
- However, LinkedLists still have quick access to the head / tail (top / bottom) elements (soon we will learn two data structures only plays with top/bottom elements).
- Arrays need a continuous memory space (sometimes might not be possible) while LinkedLists need the total memory space (they can be scattered).
- LinkedLists are more efficient in terms of space.

Challenges

- Write a program to print the middle element of a list
  - Given a singly linked list, find middle of the linked list. For example, if given linked list is 1->2->3->4->5 then output should be 3.
  - If there are even nodes, then there would be two middle nodes, we need to print second middle element. For example, if given linked list is 1->2->3->4->5->6 then output should be 4.
  - While two-traverse solution is easy, can you do it with one-traverse?

Challenges

- Write a program to reverse a linked list
  - Given pointer to the head node of a linked list, the task is to reverse the linked list. We need to reverse the list by changing links between nodes.
  - **Input**: Head of following linked list:
    1->2->3->4->NULL
  - **Output**: Linked list should be changed to:
    4->3->2->1->NULL

Thank you!