DATA STRUCTURE AND ALGORITHMS

LECTURE 3

Abstract Data Type and List ADT

DATA STRUCTURE AND ALGORITHMS

LECTURE 3b

List ADT

Reference links:

https://www.comp.nus.edu.sg/~stevenha/cs2040.html

By Dr. Steven Halim - NUS

Book [M.Goodrich, chapter 7]

Lecture outline

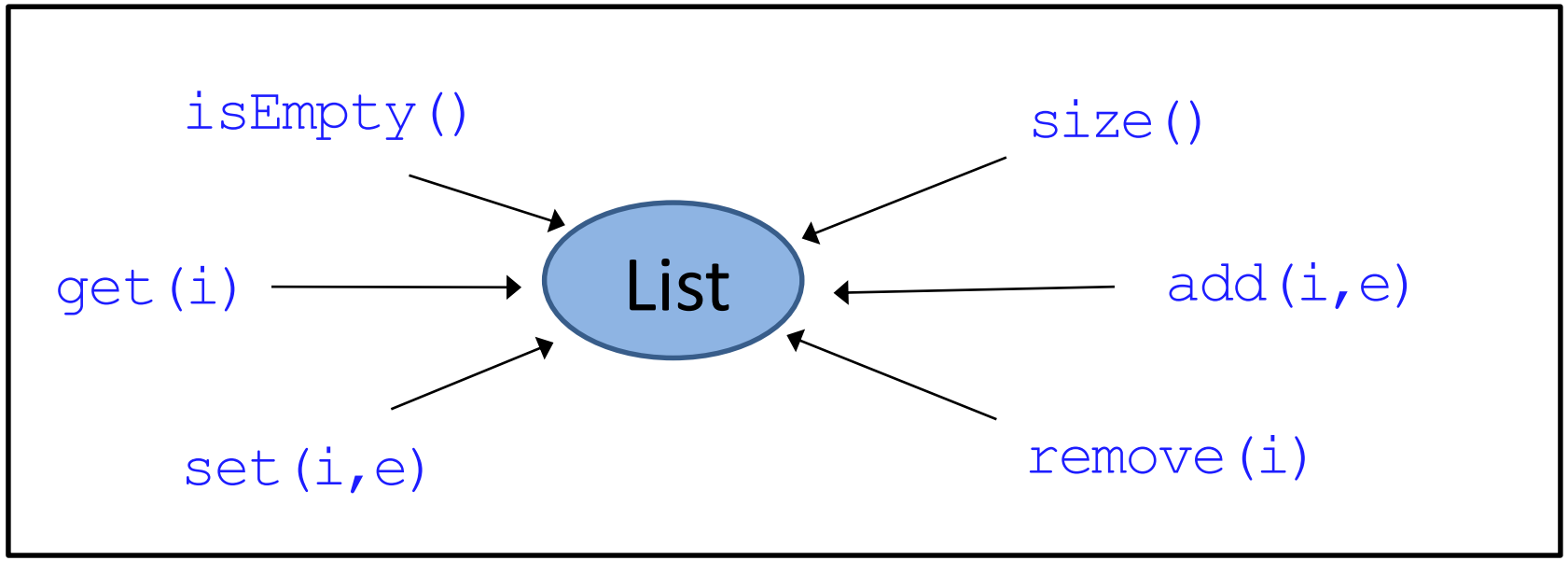
- List ADT
  - Specification
- Implementation for List ADT
  - Array Based
  - Linked List Based
    - Variation of Linked Lists

List specification

- List: A sequence of items where positional order matter <a1, a2,…, an-1, an>
- Lists are very pervasive in computing
  - e.g. student list, list of events, list of appointments etc

List specification

The **`list`** ADT

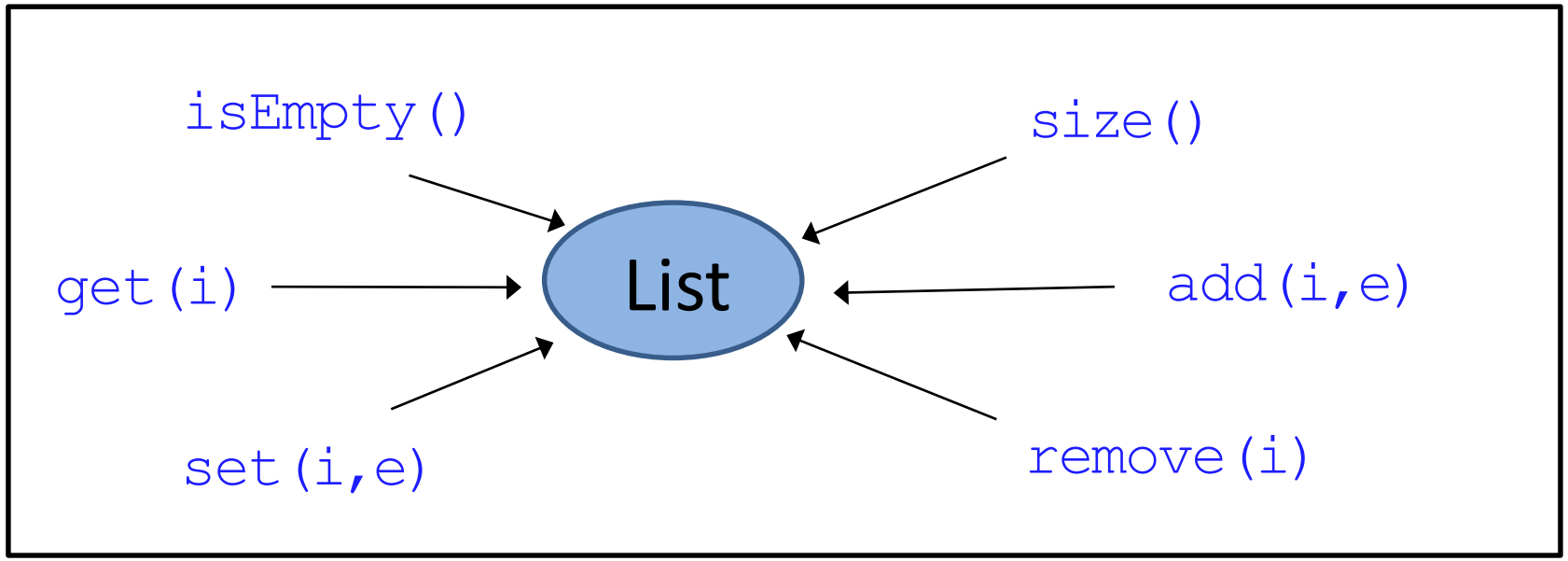


`i`: Position, integer

`e`: Data stored in list, can be any data type

List specification

The **`list`** ADT



`i`: Position, integer

`e`: Data stored in list, can be any data type

List specification: illustration

Method	Return Value	List Contents
add(0, A)	–	(A)
add(0, B)	–	(B, A)
get(1)	A	(B, A)
set(2, C)	"error"	(B, A)
add(2, C)	–	(B, A, C)
add(4, D)	"error"	(B, A, C)
remove(1)	A	(B, C)
add(1, D)	–	(B, D, C)
add(1, E)	–	(B, E, D, C)
get(4)	"error"	(B, E, D, C)
add(4, F)	–	(B, E, D, C, F)
set(2, G)	D	(B, E, G, C, F)
get(2)	G	(B, E, G, C, F)

Example: Some operations on a list of characters

List specification: in Java

```
1  /** A simplified version of the java.util.List interface. */
2  public interface List<E> {
3    /** Returns the number of elements in this list. */
4    int size( );
5
6    /** Returns whether the list is empty. */
7    boolean isEmpty( );
8
9    /** Returns (but does not remove) the element at index i. */
10   E get(int i) throws IndexOutOfBoundsException;
11
12   /** Replaces the element at index i with e, and returns the replaced element. */
13   E set(int i, E e) throws IndexOutOfBoundsException;
14
15   /** Inserts element e to be at index i, shifting all subsequent elements later. */
16   void add(int i, E e) throws IndexOutOfBoundsException;
17
18   /** Removes/returns the element at index i, shifting subsequent elements earlier. */
19   E remove(int i) throws IndexOutOfBoundsException;
20 }
```

A simple version of the list interface [M.Goodrich,259]

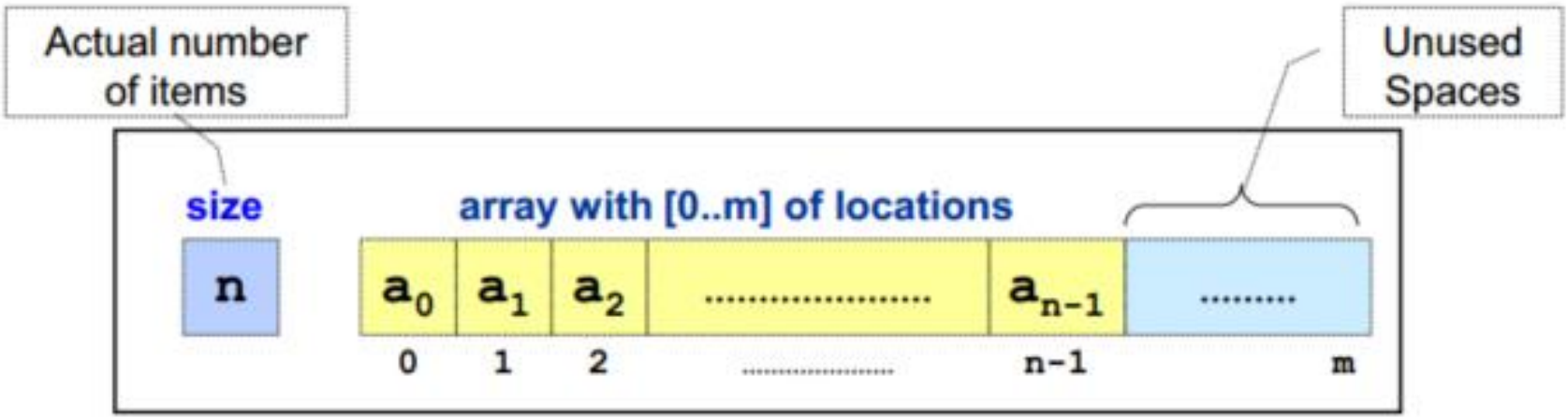
List implementation

Using array

[M. Goodrich, section 7.2]

Array lists

- Array is a prime candidate for implementing the list
  - Simple construct to handle a collection of items
- Advantage:
  - Very fast retrieval

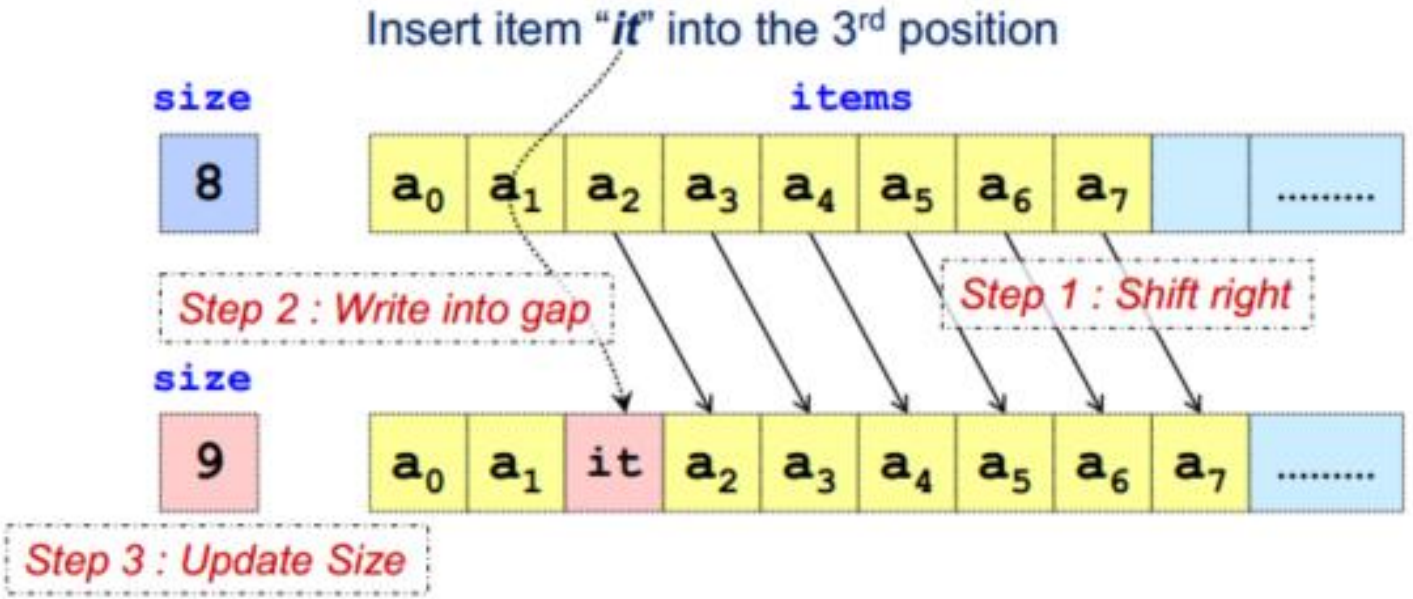


Internal of the list ADT, Array Version

Array lists: Insertion (chèn)

- Simplest Case: Insert to the end of array
- Other Insertions:
  - Some items in the list needs to be shifted
  - Worst case: Inserting at the head of array

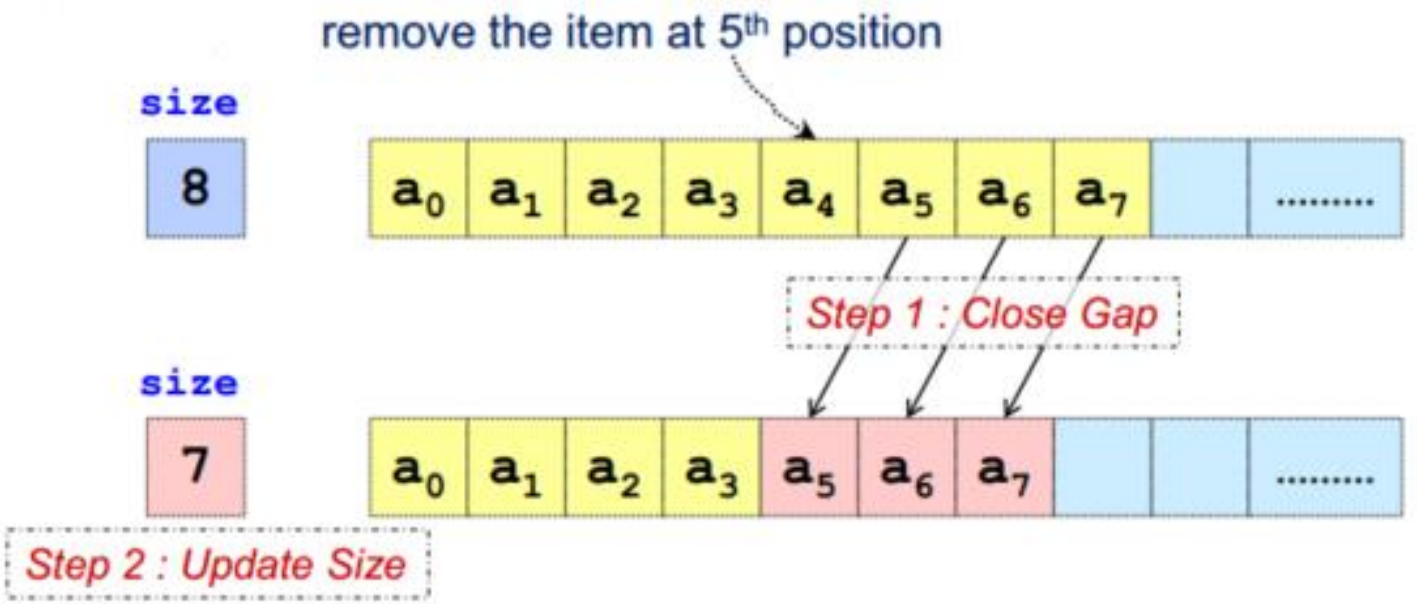
Example



Array lists: Deletion (chèn)

- Simplest Case: Delete item from the end of array
- Other deletions:
  - Item needs to be shifted
  - Worst case: Deleting at the head of array

Example



Array lists: Efficiency (time)

- Retrieval – lấy ra một phần tử
  - Fast: one access O(1)
- Insertion – chèn thêm vào 1 phần tử
  - Best case: No shifting of elements (thêm vào cuối) O(1)
  - Worst case: Shifting of all N elements (thêm vào đầu) O(*n*)
- Deletion – xóa đi 1 phần tử
  - Best case: No shifting of elements (xóa phần tử cuối) O(1)
  - Worst case: Shifting of all N elements (xóa phần tử đầu) O(*n*)

Array lists: Implementation in Java

- Class ArrayList in java.util
  - https://docs.oracle.com/javase/9/docs/api/java/util/ArrayList.html
- How is your implementation? What for?
  - To understand (để hiểu)
  - To customize (để tùy chỉnh cho những ứng dụng riêng)

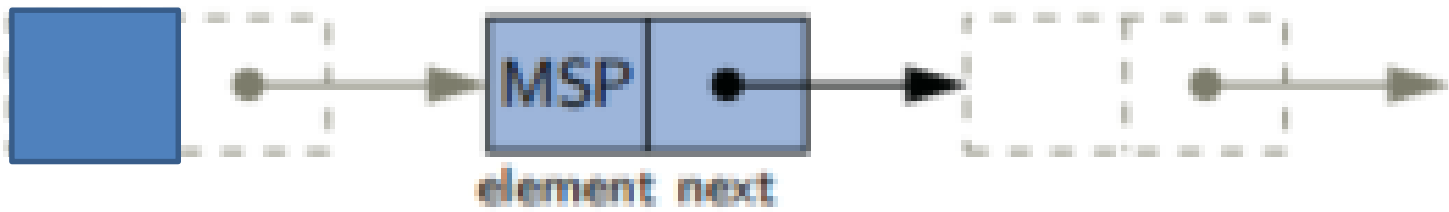
List ADT implementation

Using Linked List

[M. Goodrich, section 7.3]

Linked lists

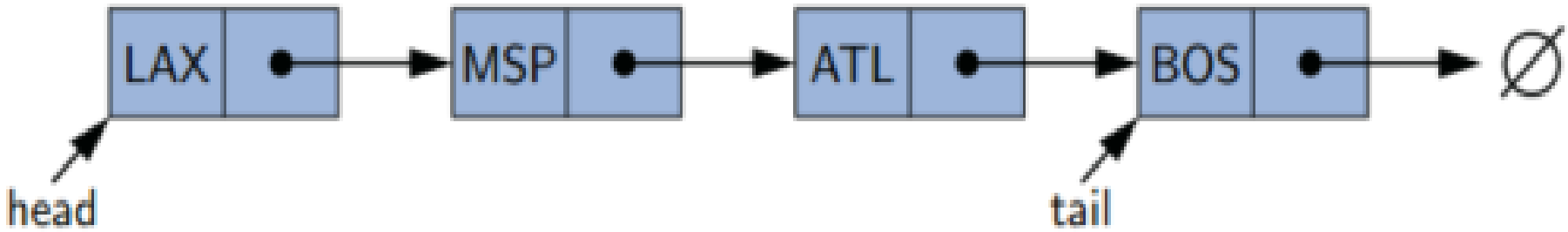
- Linked List is a collection of **nodes** that collectively form a linear sequence
  - Allow elements to be non-contiguous in memory
  - Order the elements by associating each with its neighbour(s) through pointers




```
class Node {
                T element; //data của node
                Node next; //địa chỉ phần tử lân cận
            }
```

Linked lists: iilustration

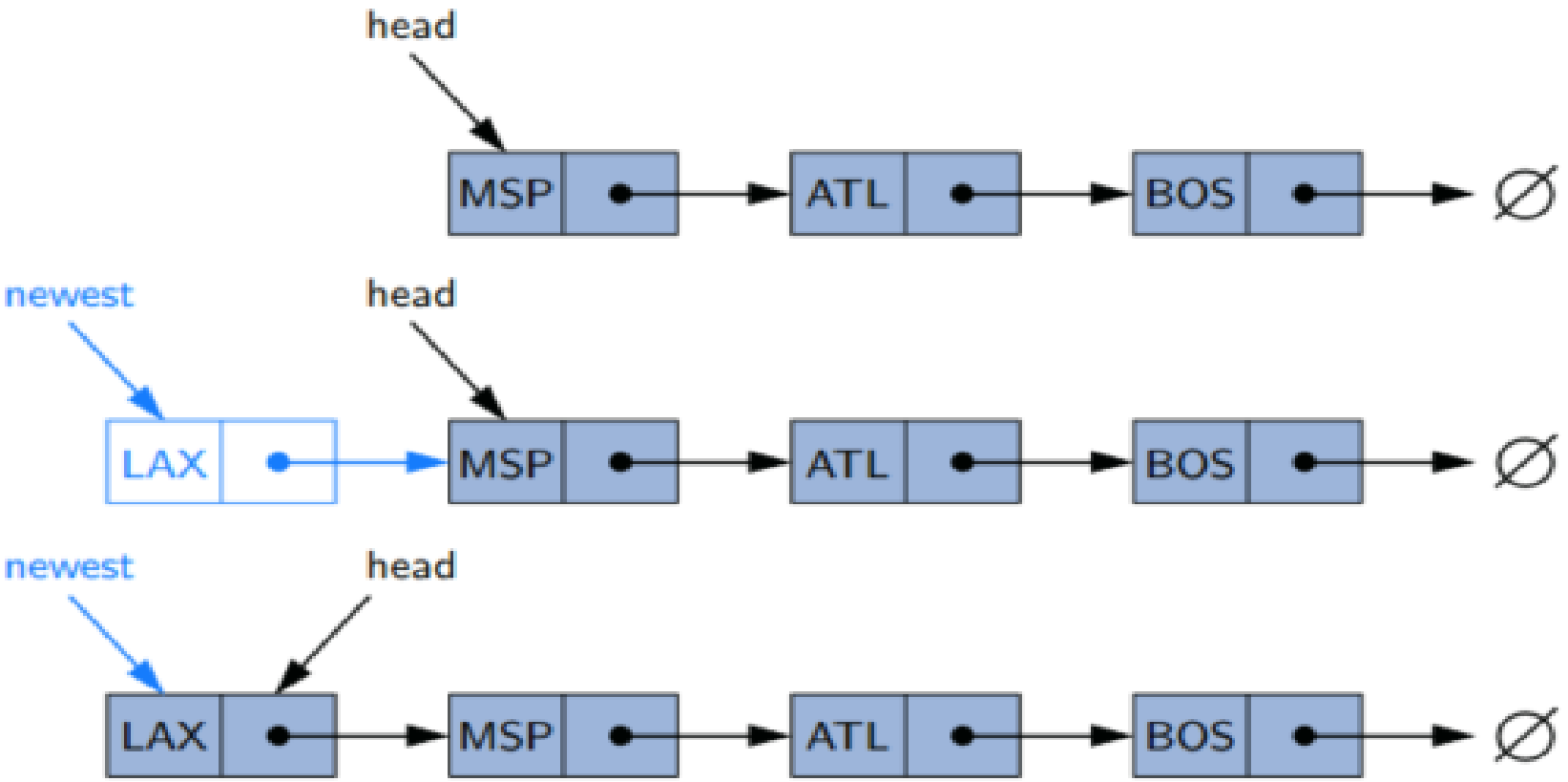
Linked list of four items < LAX, MSP, ATL, BOS >



- We need
  - head is reference to indicate the first node
  - tail is reference to indicate the last node, which has null as its next reference

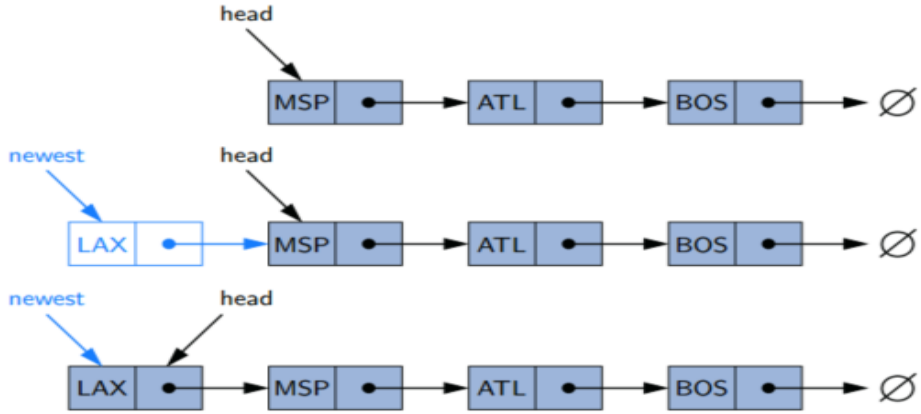
Link lists: Insertion (chèn)

- Insertion of an element **at the head** of a linked list
  - 3 steps (định vị list; thêm node mới; định vị lại list)



Link lists: Insertion (chèn)

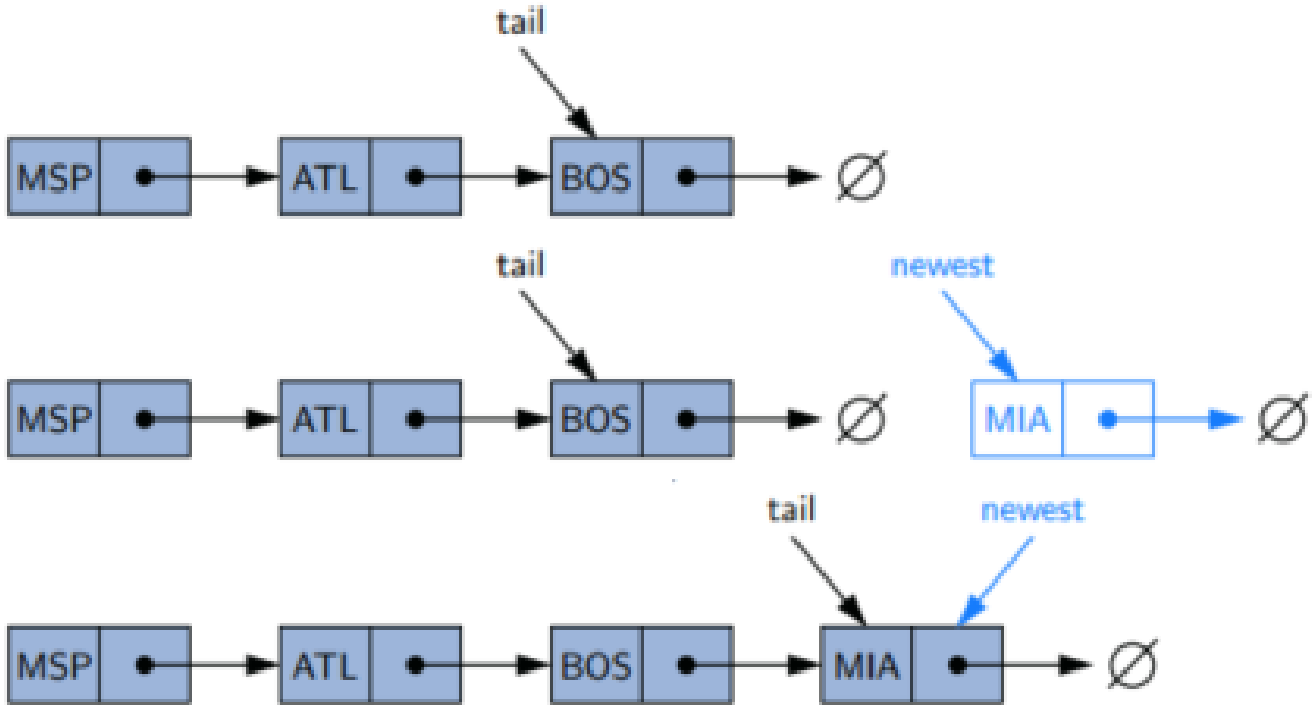
- Insertion of an element **at the head** of a linked list
  - 3 steps (định vị list; thêm node mới; định vị lại list)



  - **Algorithm** addFirst(e):
    newest = Node(e)
    newest.next = head
    head = newest
    size = size + 1

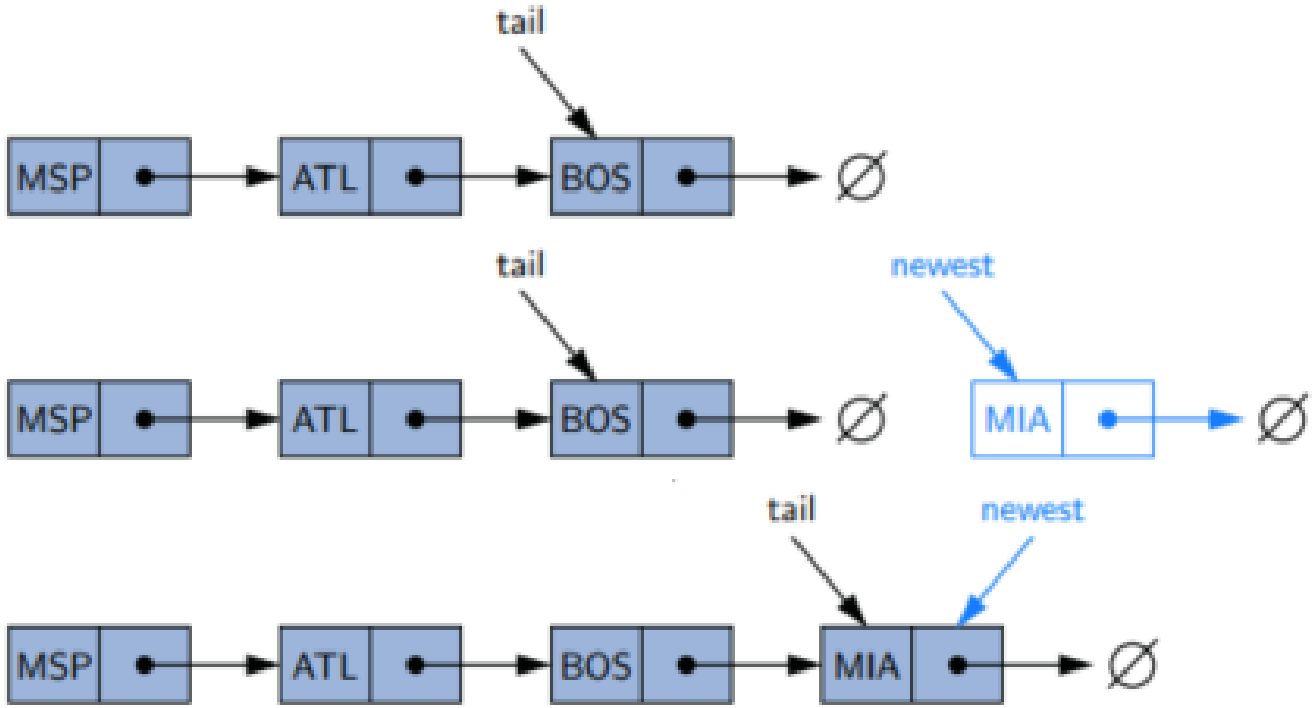
Link lists: Insertion (chèn)

- Insertion of an element **at the end** of a linked list
  - 3 steps (định vị node cuối; thêm node mới; gắn vào cuối)



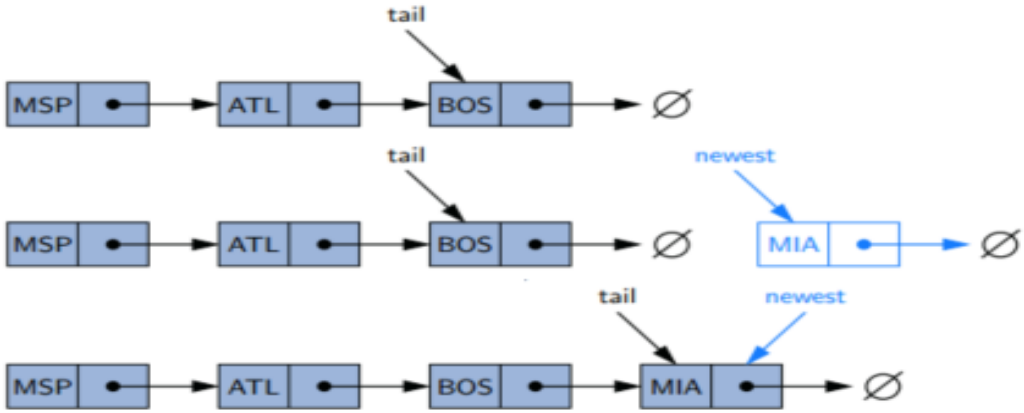
Link lists: Insertion (chèn)

- Insertion of an element **at the end** of a linked list
  - 3 steps (định vị node cuối; thêm node mới; gắn vào cuối)



Link lists: Insertion (chèn)

- Insertion of an element **at the end** of a linked list
  - 3 steps (định vị node cuối; thêm node mới; gắn vào cuối)

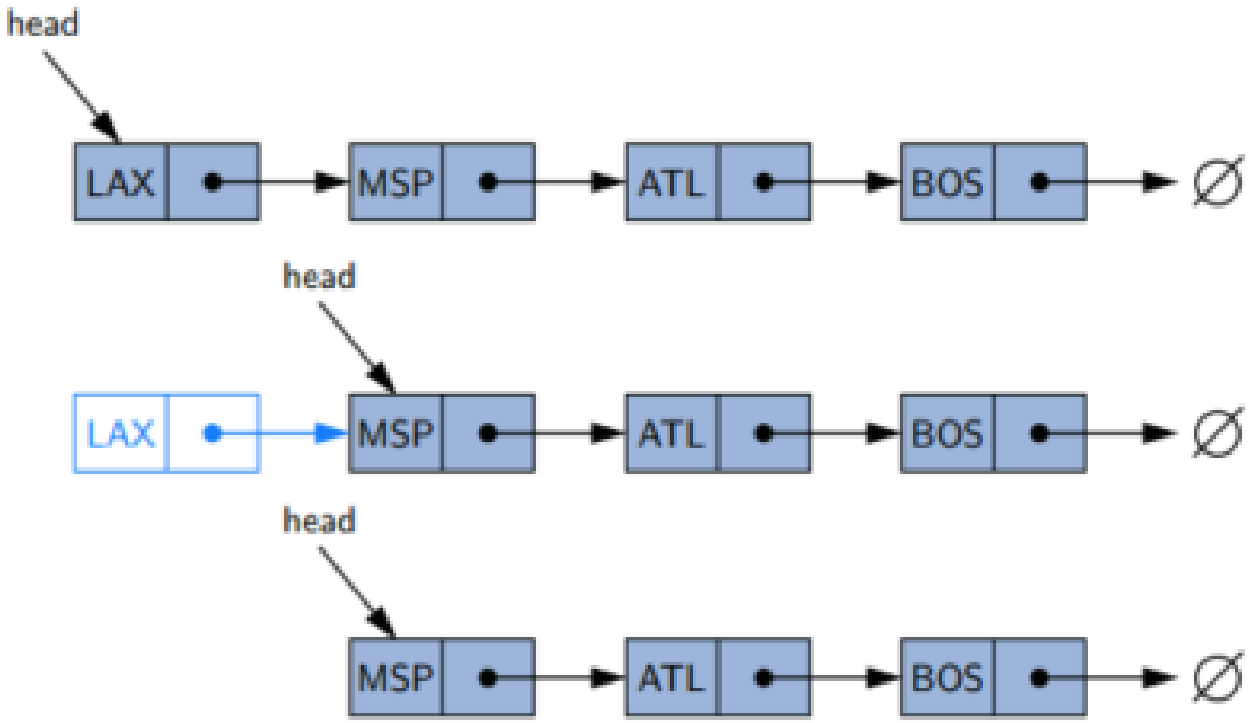


  - **Algorithm** addLast(e):

```
newest = Node(e)
newest.next = null
tail.next = newest
tail = newest
size = size + 1
```

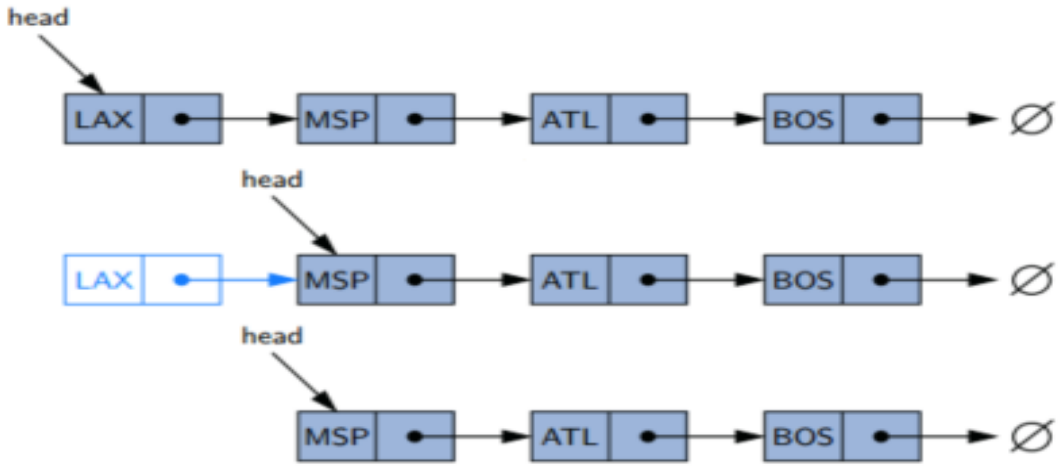
Link lists: Deletion (xóa)

- Removing an element **from the head** of a linked list
  - 2 steps (định vị list; cho head tham chiếu đến phần tử kề)



Link lists: Deletion (xóa)

- Removing an element **from the head** of a linked list
  - 2 steps (định vị list; cho head tham chiếu đến phần tử kề)

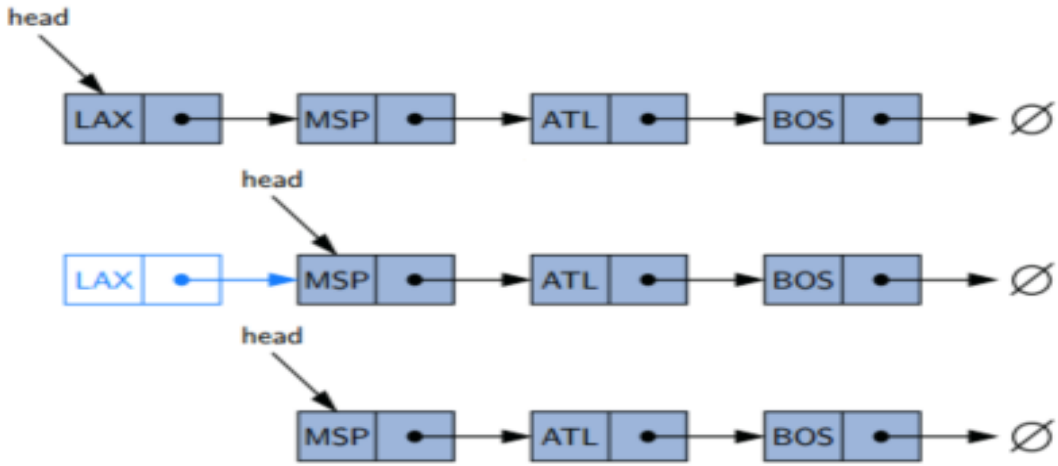


  - **Algorithm** removeFirst( ):

```
if head == null then
    the list is empty.
head = head.next
size = size − 1
```

Link lists: More

- Removing an element **from the head** of a linked list
  - 2 steps (định vị list; cho head tham chiếu đến phần tử kề)



  - **Algorithm** removeFirst( ):

```
if head == null then
    the list is empty.
head = head.next
size = size − 1
```

Linked lists: Efficiency (time)

- Traversion
  - Index: stop at index node
  - Value: stop at node with a particular value
  - Time: max O(*n*) - *n* = size of linked list
- Retrieval:
  - One access - O(1)
- Insertion:
  - One access (three basic steps) - O(1)
- Deletion:
  - One access (one/two basic step(s)) - O(1)

Linked lists: implementation in Java

- Class LinkedList in java.util

  https://docs.oracle.com/javase/9/docs/api/java/util/LinkedList.html

- Visualize linked list

  https://visualgo.net/en/list

- How is your implementation? What for?
  - To understand
  - To customize

List implementation

Other variations

Linked lists: Variations

- The linked list implementation shown is known as singly linked list: Each node has one pointer.
  (Danh sách liên kết đơn, mỗi node chỉ có một phần tử kề)
- Other variations
  - Doubly Linked List (Danh sách liên kết đôi)
  - Circular Linked List (Danh sách liên kết vòng)
  - Tailed Linked List (Danh sách liên kết đuôi)
  - Circular Doubly-Linked List (Danh sách liên kết đôi vòng)
  - Etc.

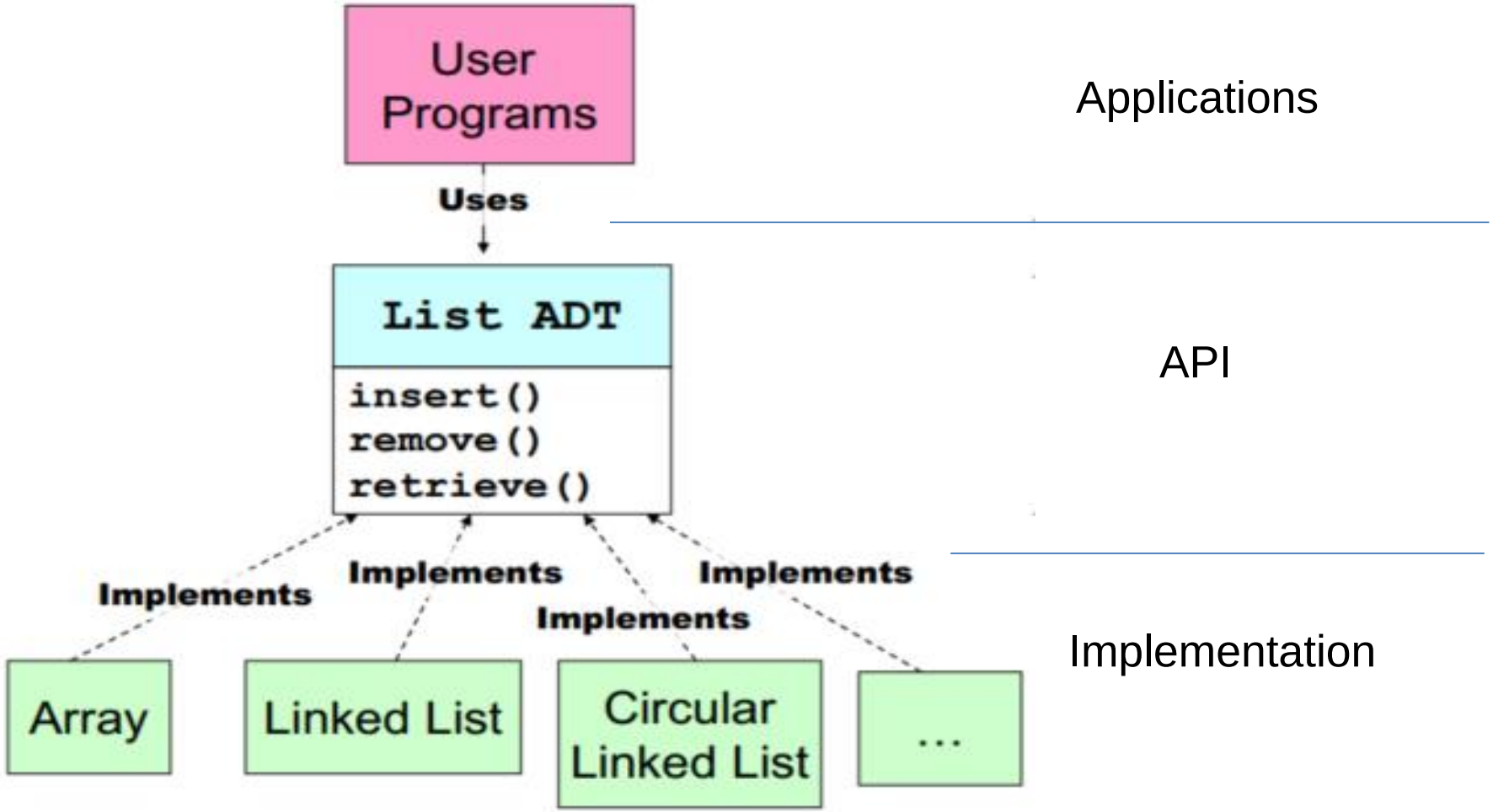
Linked lists: Application

- Storing large number (size > 10^9) – Try!
- For implementing other ADTs, such as:
  - Stack
  - Queue
  - Graph

List ADT

Summary

Summary


User Programs
Uses
List ADT
insert()
remove()
retrieve()
Implements
Implements
Implements
Implements
Array
Linked List
Circular Linked List
...
Applications
API
Implementation