CSE214 Data Structures

Arrays and Singly Linked Lists

YoungMin Kwon

SUNY Korea
The State University of New York

Lists

- List
  - A linear sequence of elements
- Implementations
  - Arrays
  - Singly linked lists
  - Circularly linked lists
  - Doubly linked lists

SUNY Korea

Arrays

- Example 1
  - Storing game entries for a video game in an array
  - Information to store
    - Name of the player
    - Score of the game
  - GameEntry class for the entries

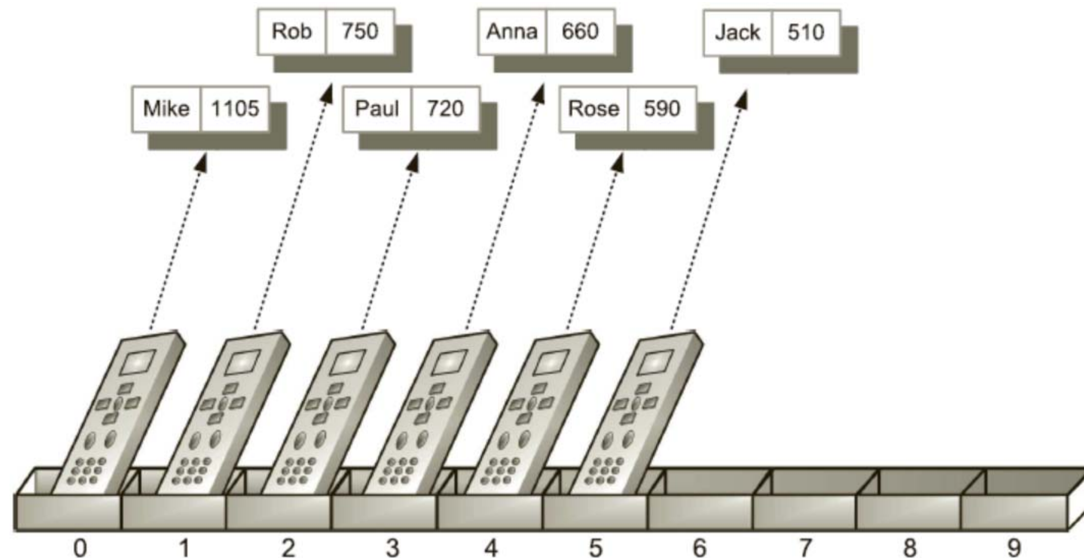
SUNY Korea

```
public class GameEntry {
  private String name; // name of the person earning this score
  private int score;    // the score value
  /** Constructs a game entry with given parameters.. */
  public GameEntry(String n, int s) {
    name = n;
    score = s;
  }
  /** Returns the name field. */
  public String getName( ) { return name; }
  /** Returns the score field. */
  public int getScore( ) { return score; }
  /** Returns a string representation of this entry. */
  public String toString( ) {
    return "(" + name + ", " + score + ")";
  }
}
```

SUNY Korea

Scoreboard

- A class to store the high scores
- Use a fixed length array of GameEntry
  - The entries in the list are sorted from highest score to lowest one



SUNY Korea

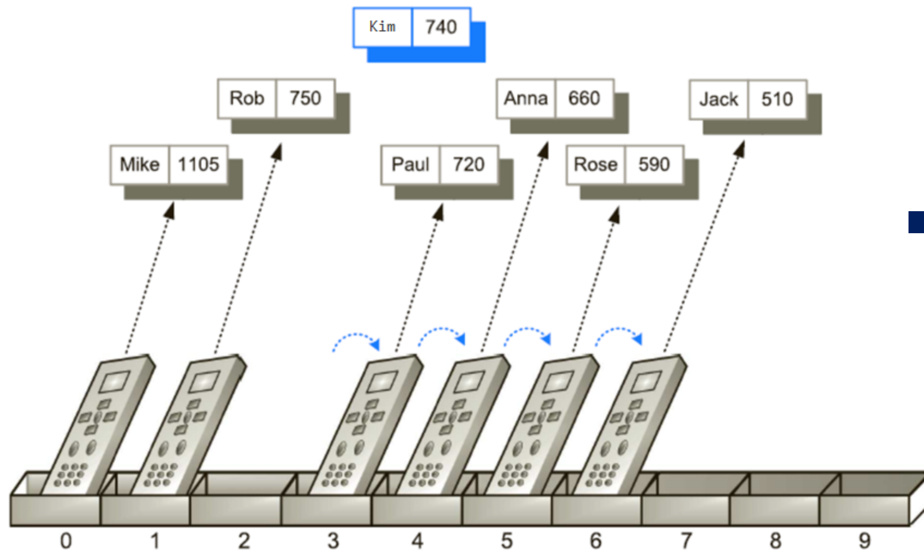
```
/** Class for storing high scores in an array in nondecreasing order. */
public class Scoreboard {
  private int numEntries = 0; // number of actual entries
  private GameEntry[ ] board; // array of game entries (names & scores)
  /** Constructs an empty scoreboard with the given capacity
                                                    for storing entries. */
  public Scoreboard(int capacity) {
    board = new GameEntry[capacity];
  }
  // more methods will go here
}
```

SUNY Korea
The State University of New York

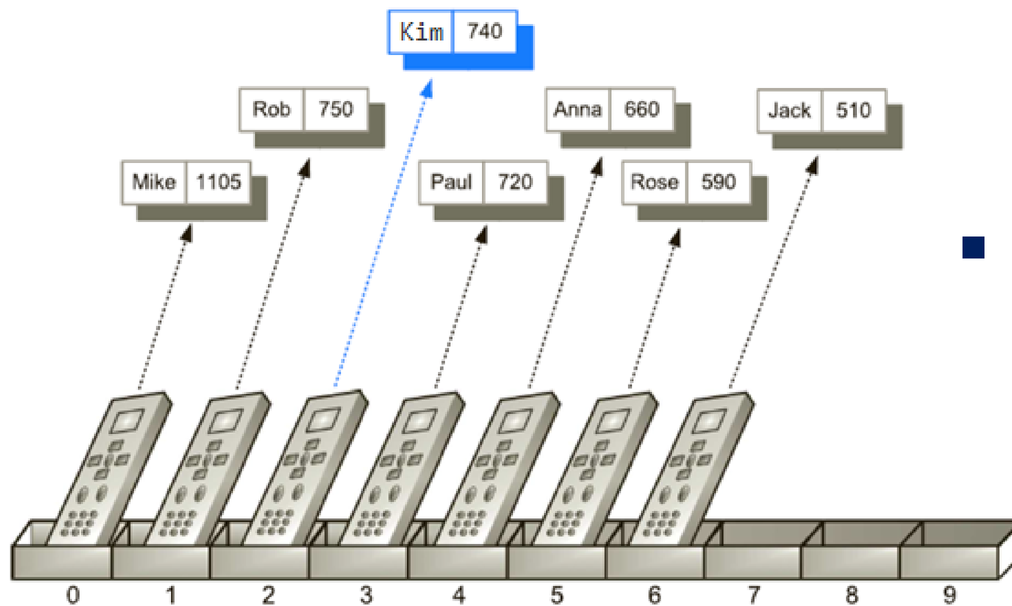
Adding an Entry

- Adding a new entry to a Scoreboard
  - If the board is not full, any new entry will be added
  - If the board is full, a new entry will be added only when it is strictly better than the other scores
    - Better than the last entry
  - To add a new entry
    - To make room, shift downward the entries that have lower score than the new one
    - Add the new entry to the empty space

SUNY Korea



- Make room for a new reference



- Add a reference to Kim

SUNY Korea

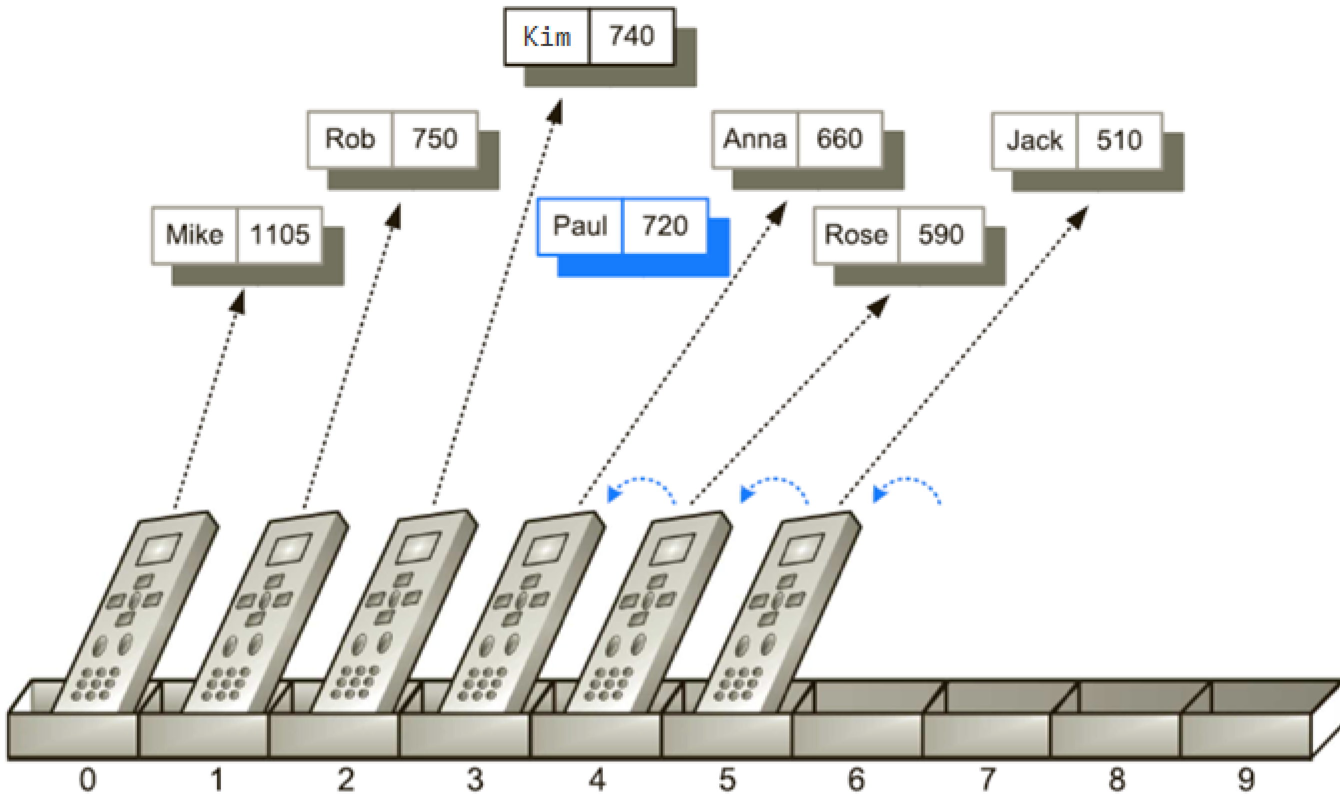
```
/** Attempt to add a new score to the collection (if it is high enough) */
public void add(GameEntry e) {
  int newScore = e.getScore( );

  // is the new entry e really a high score?
  if (numEntries < board.length || newScore > board[numEntries−1].getScore( )) {
    if (numEntries < board.length)                 // no score drops from the board
      numEntries++;                                // so overall number increases

    // shift any lower scores rightward to make room for the new entry
    int j = numEntries − 1;
    while (j > 0 && board[j−1].getScore( ) < newScore) {
      board[j] = board[j−1];                       // shift entry from j-1 to j
      j−−;                                         // and decrement j
    }

    // when done, add new entry
    board[j] = e;
  }
}
```

SUNY Korea
The State University of New York

Removing an Entry

- Removing an entry at index *i* from a Scoreboard
  - When an entry is removed, any lower scores should be shifted upward to fill the empty space
  - If *i* is outside of the current entries throw an IndexOutOfBoundsException
  - Return the entry that was removed

SUNY Korea



SUNY Korea

```
/** Remove and return the high score at index i. */
public GameEntry remove(int i) throws IndexOutOfBoundsException {
  if (i < 0 || i >= numEntries)
    throw new IndexOutOfBoundsException("Invalid index: " + i);

  GameEntry temp = board[i];                  // save the object to be removed

  for (int j = i; j < numEntries − 1; j++)    // count up from i (not down)
    board[j] = board[j+1];                    // move one cell to the left

  board[numEntries −1 ] = null;               // null out the old last score
  numEntries−−;

  return temp;                                // return the removed object
}
```

SUNY Korea
The State University of New York

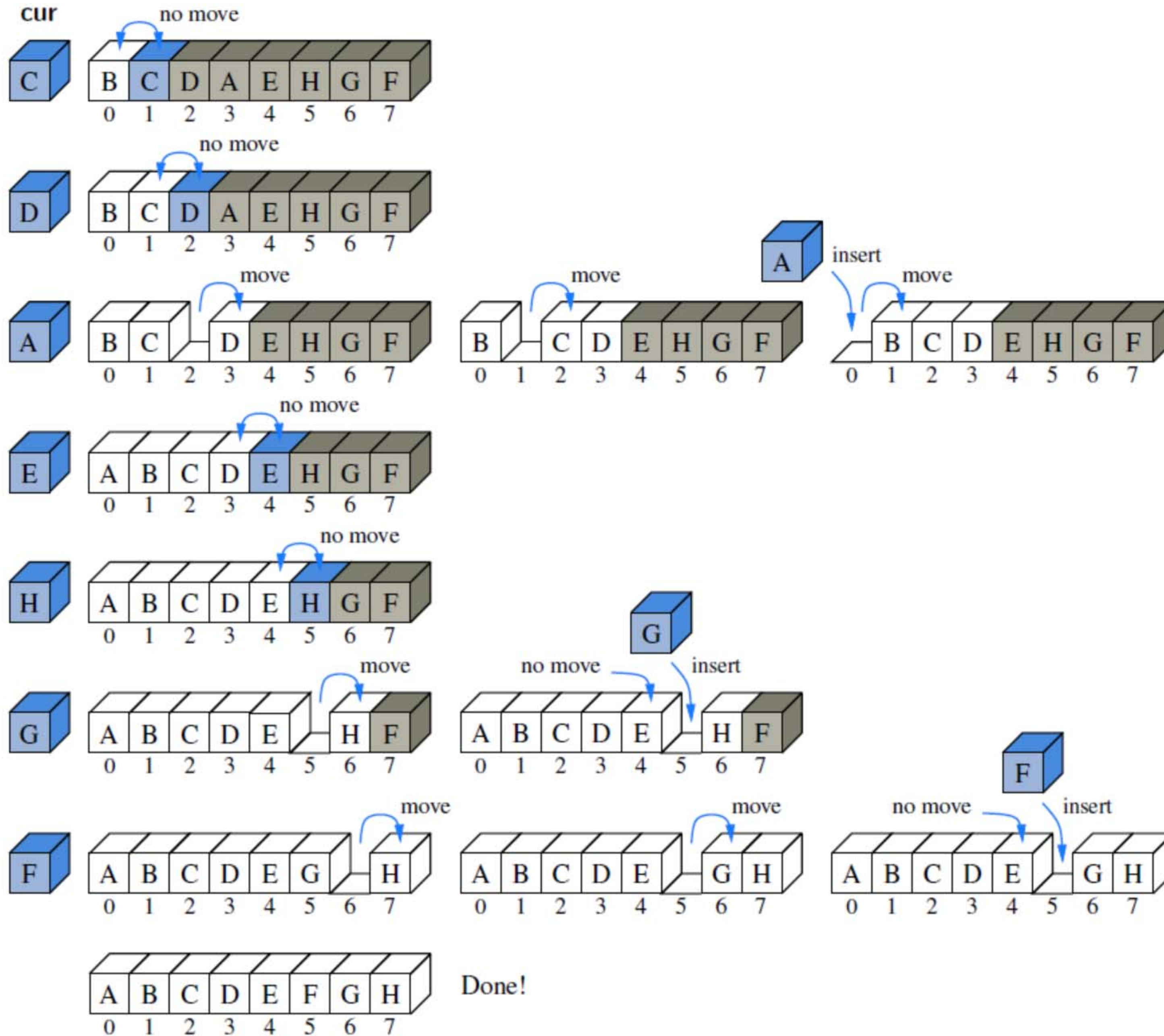
Sorting an Array

- The insertion-sort algorithm
  - Works similarly as the Scoreboard example
  - Given an array of length n
  - Consider only the first k elements of the array at a time while changing k from 1 to n-1
  - Insert the kth element to the first k sorted elements

SUNY Korea
The State University of New York

Insertion-sort

Algorithm InsertionSort(A):
  Input: An array A of n comparable elements
  Output: The array A with elements rearranged in nondecreasing order
  for k from 1 to $n-1$ **do**
    Insert $A[k]$ at its proper location within $A[0], A[1], \ldots, A[k]$.



SUNY Korea

Insertion-sort

```
/** Insertion-sort of an array of characters into nondecreasing order */
public static void insertionSort(char[ ] data) {
  int n = data.length;
  for (int k = 1; k < n; k++) {              // begin with second character

    char cur = data[k];                      // time to insert cur=data[k]

    int j = k;                               // find correct index j for cur
    while (j > 0 && data[j−1] > cur) {       // thus, data[j-1] must go after cur
      data[j] = data[j−1];                   // slide data[j-1] rightward
      j−−;                                   // and consider previous j for cur
    }

    data[j] = cur;                           // this is the proper place for cur
  }
}
```

SUNY Korea
The State University of New York

Insertion-sort (using swap)

```
//swap data[i] and data[j]
public static void swap(char[] data, int i, int j) {
    char tmp = data[i];
    data[i]  = data[j];
    data[j]  = tmp;
}
//insertion sort (using swap)
public static void insertionSort(char[] data) {
    int n = data.length;
    for(int k = 1; k < n; k++) {
        for(int j = k; j > 0; j--) {
            if(data[j-1] > data[j]) {
                swap(data, j-1, j);
            }
        }
    }
}
public static void main(String[] args) {
    char[] data = new char[] {'s', 'a', 'm', 'p', 'l', 'e'};
    insertionSort(data);
    for(char c : data)
        System.out.print (c + " ");
}
```

SUNY Korea

Insertion-sort

- Insertion sort demo
  - https://www.youtube.com/watch?v=ROalU379l3U

Singly Linked Lists

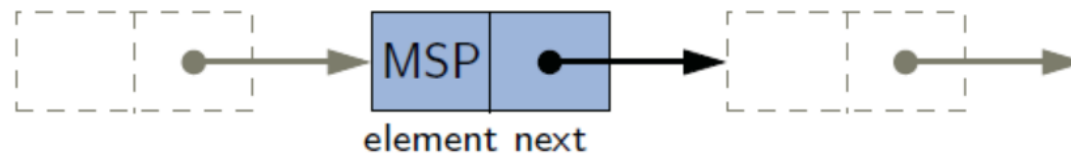
- Drawbacks of using arrays
  - The capacity of an array must be fixed when it is created
    - In many applications, we cannot predict the required capacity
    - Allocate for the possible maximum $\rightarrow$ waste of memory
  - Insertion and deletion are time consuming
    - Shifting many elements

SUNY Korea

Singly Linked Lists

- Linked List
  - A collection of nodes that form a linear sequence

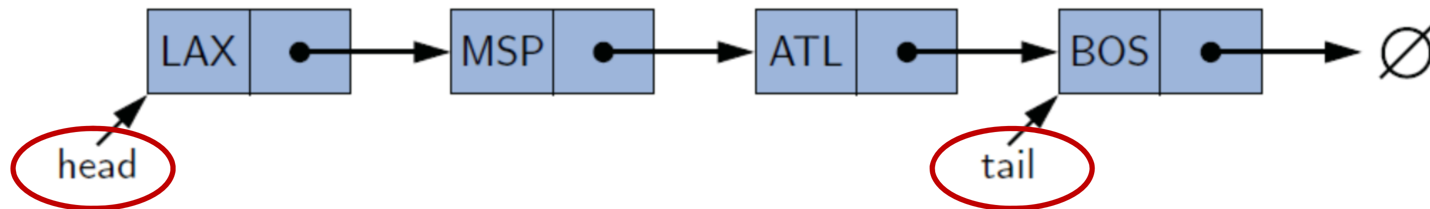
- Node
  - Has a reference to its element object
  - Has a reference to the next node



SUNY Korea

Singly Linked Lists

- Linear structure



- Head: points to the first node
- Tail: points to the last node
  - Tail node has null in its next reference variable
  - Tail node can be found by following the nodes from the head
- Count: the number of elements in the list
  - Count can be computed while traversing along the nodes

SUNY Korea

Inserting an Element

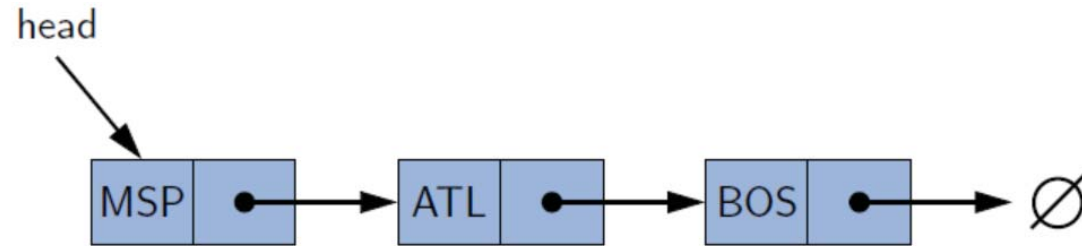
- Inserting a node at the head
  - Create a new node with a reference to the element
  - Set its next link to reference the node where the current head is referencing
  - Make the head reference the new node

Algorithm addFirst(e):

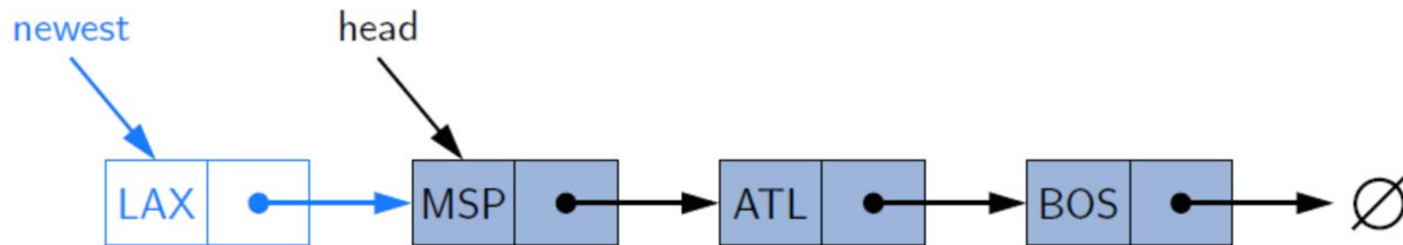
newest = Node(e) {create new node instance storing reference to element e}
newest.next = head {set new node's next to reference the old head node}
head = newest {set variable head to reference the new node}
size = size + 1 {increment the node count}

SUNY Korea

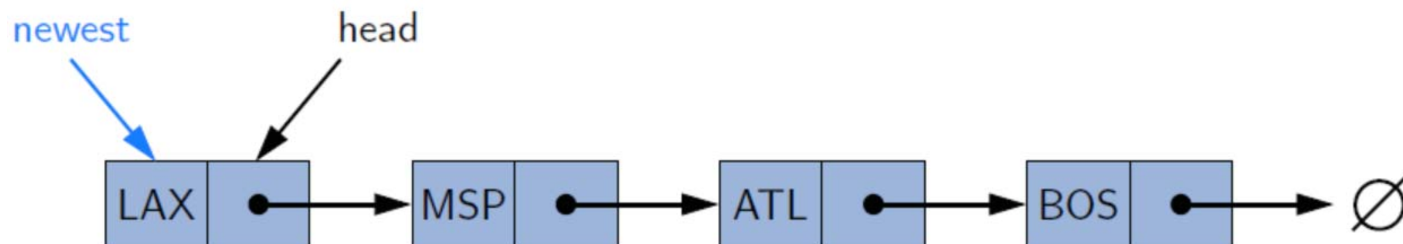
Inserting an Element



(a)



(b)



(c)

SUNY Korea

Inserting an Element

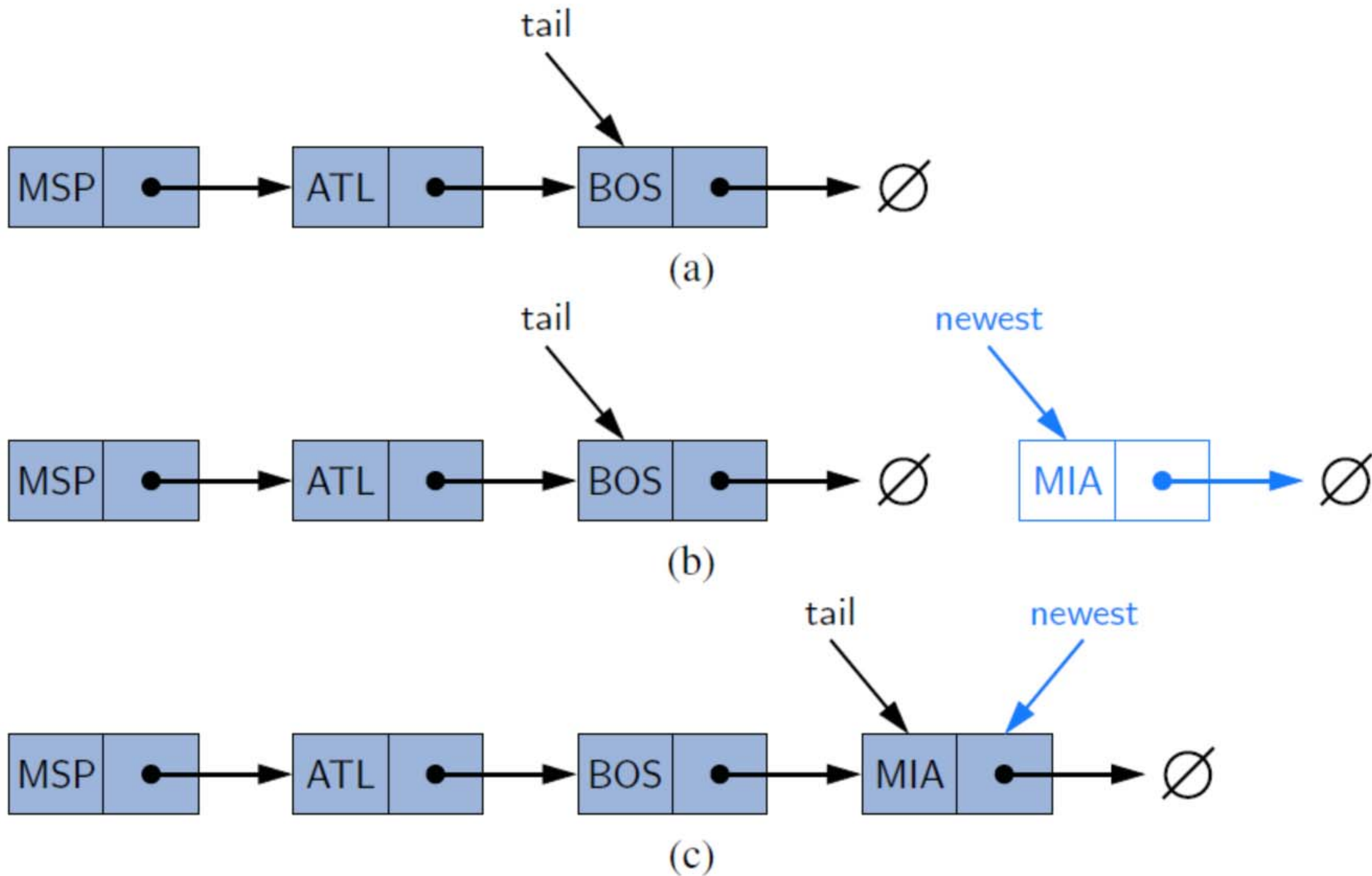
- Inserting a node at the tail
  - Create a new node with a reference to the element
  - Set its next link to null
  - Make the tail node's next reference the new node
  - Make the tail reference the new node

Algorithm addLast(e):

```
newest = Node(e)           {create new node instance storing reference to element e}
newest.next = null         {set new node's next to reference the null object}
tail.next = newest         {make old tail node point to new node}
tail = newest              {set variable tail to reference the new node}
size = size + 1            {increment the node count}
```

SUNY Korea

Inserting an Element



SUNY Korea

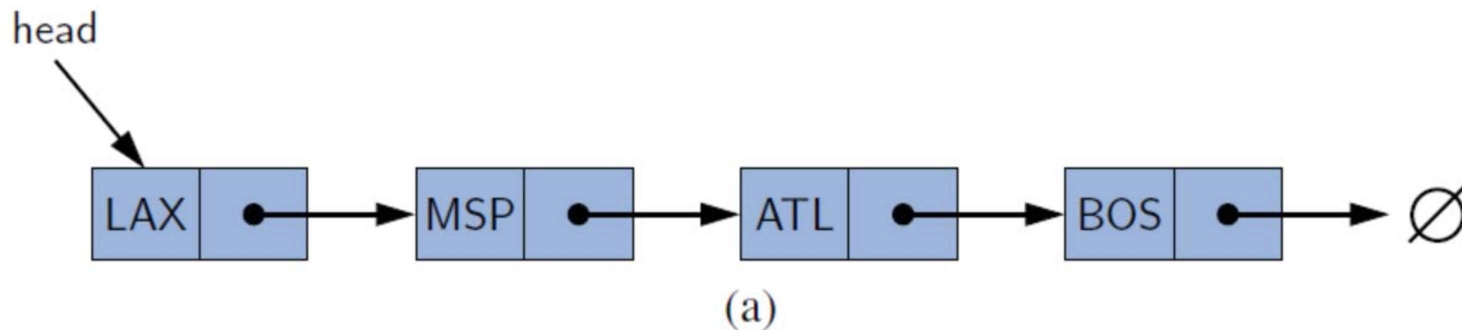
Removing an Element

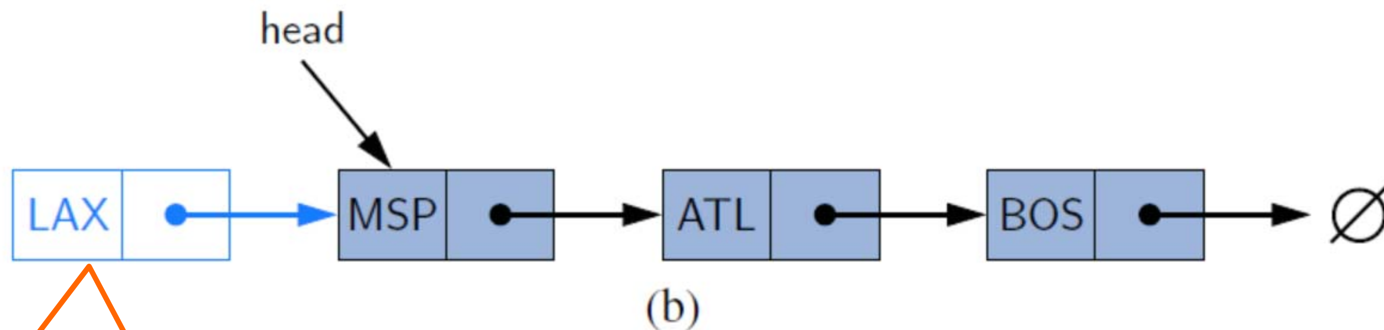
- Removing a node from the head
  - Set the head to the head's next link

```
Algorithm removeFirst():
    if head == null then
        the list is empty.
    head = head.next        {make head point to next node (or null)}
    size = size − 1         {decrement the node count}
```

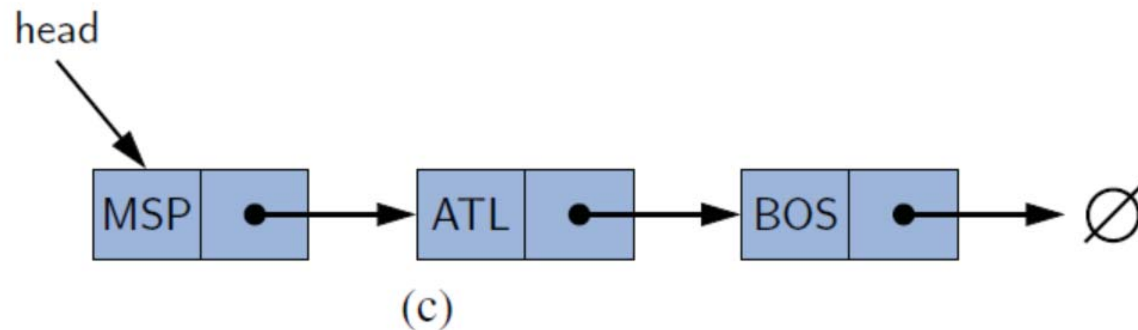
SUNY Korea
The State University of New York

Removing an Element





No one is referencing this node
⇒ it will be garbage collected



SUNY Korea

Removing an Element

- Removing a node from the tail
  - Unfortunately, it is not easy to remove a node from the tail
  - Need to find the node before the tail
  - Need to scan from the head to the node before the tail
- This task can be easily achieved using doubly linked lists

SUNY Korea

Implementing SinglyLinkedList

```
public class SinglyLinkedList<E> {
    //Node of the list
    private static class Node<E> {
        private E e;               //element
        private Node<E> next;      //next node

        public Node(E e, Node<E> n)     { this.e = e; this.next = n; }
        public E getElement()           { return e; }
        public Node<E> getNext()        { return next; }
        public void setNext(Node<E> n) { next = n; }
    }

    private Node<E> head, tail;
    private int size;

    public SinglyLinkedList() {}
    public int size()         { return size; }
    public boolean isEmpty()  { return size == 0; }
```

SUNY Korea

```java
public E first() {
    return isEmpty() ? null : head.getElement();
}

public E last() {
    return isEmpty() ? null : tail.getElement();
}

public void addFirst(E e) {
    head = new Node<>(e, head);
    if(isEmpty())  //special handling for empty case
        tail = head;
    size++;
}

public void addLast(E e) {
    if(isEmpty())  //special handling for empty case
        addFirst(e);
    else {
        tail.setNext(new Node<>(e, null));
        tail = tail.getNext();
        size++;
    }
}
```

SUNY Korea

```java
public E removeFirst() {
    if(isEmpty())
        return null;

    //return value
    E ret = head.getElement();

    //make head point to the next node
    head = head.getNext();
    size--;

    //if empty, make tail null
    if(size == 0)
        tail = null;

    return ret;
}

public E removeLast() {
    //TODO: implement this method
}
```

SUNY Korea
The State University of New York

```
    private static void onFalseThrow(boolean b) {
        if(!b)
            throw new RuntimeException("Error: unexpected");
    }

    public static void main(String[] args) {
        SinglyLinkedList<Integer> list =
                                      new SinglyLinkedList<>();

        list.addLast(2);
        list.addLast(3);
        list.addLast(4);
        list.addFirst(1);

        onFalseThrow(list.removeLast()  == 4);
        onFalseThrow(list.removeLast()  == 3);
        onFalseThrow(list.removeFirst() == 1);
        onFalseThrow(list.removeLast()  == 2);
        System.out.println("Success!");
    }
}
```

SUNY Korea