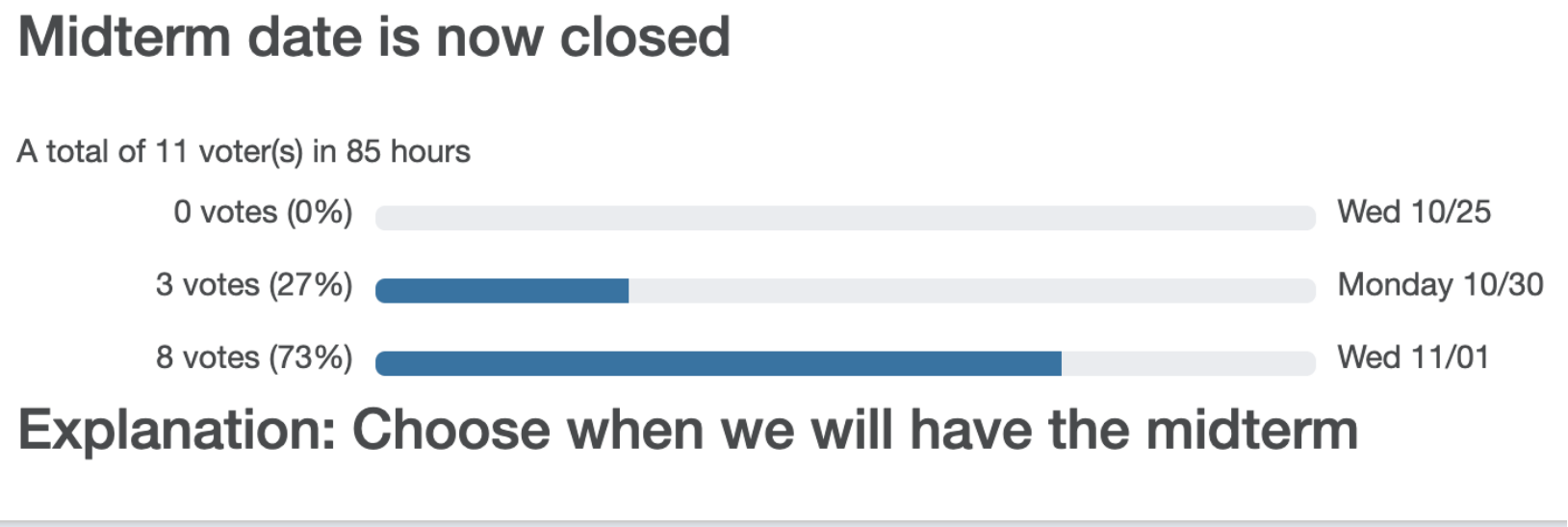
CS151 Intro to Data Structures

LinkedLists

Announcements

- Midterm: Wednesday November 1st



If you have a conflict, let me know

Announcements

Goldengate:

- Don’t work directly on goldengate
- From goldengate, ssh into another machine
- List of available machines:
    - https://cs.brynmawr.edu/~gtowell/crp.html

Announcements

- HW01 due Tuesday (09/19)
  - Will be released later tonight
  - Will be using your ExpandableArray from today's lab

- Lab checkoff, deadline is when corresponding HW is due

instanceof

- An operator that tests to see if an object is an instance of a specified type
- Every subclass object is an instance of its super class – not true the other way

```
class A {} class B extends A{} class C extends B{}
A[] as = {new A(), new B(), new C()};
for (int i=0; i<as.length; i++) {
  System.out.print((as[i] instanceof A)+ " ");
  System.out.print((as[i] instanceof B)+ " ");
  System.out.println(as[i] instanceof C);
}
```

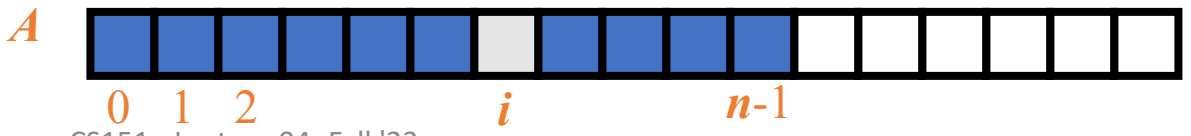
Outline

- Lists vs Arrays
- LinkedLists

Arrays

Arrays

- Fixed size – once created, can not be resized
- Contiguous memory allocation
- Fast random access – $O(1)$
- Slow insertion and deletion – $O(n)$
  - must shift the rest



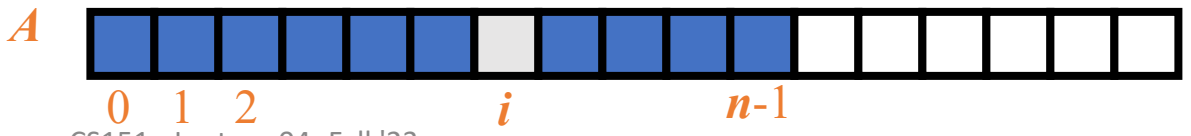
Arrays and Lists

Arrays

- Fixed size – once created, can not be resized
- Contiguous memory allocation
- Fast random access – $O(1)$
- Slow insertion and deletion – $O(n)$ must shift the rest

Lists

- Dynamic size:
  - grows and shrinks
- Non-contiguous memory allocation
- Slow random lookup – $O(n)$ must traverse
- Fast insertion and deletion – $O(1)$



ArrayList

- Dynamically-sized array
- Stores an ordered sequence of objects
- Can grow and shrink when items are added/removed
- Standard array features all supported, but with different syntax
- Part of Java collections framework
- `import java.util.ArrayList`

- `ExpandableArray` is a simple ArrayList

ArrayList - Implementation

implemented with an array

Dynamic sizing based on the current size

- A variable keeps track of the current size
- initially it is equal to the specified size or an estimate (if created empty)
- after deletion, elements are shifted to the left and size is decremented
- after addition, if not enough space, array will be expanded (doubled) and all elements copied

Methods of an ArrayList

`add(o)`	appends o at the end of list
`add(index, o)`	inserting given o at index, shifting list to the right
`get(index)`	returns the object found at index
`remove(index)`	removes the object found at index and returns it, shifting list to the left
`set(index, o)`	replaces object at given index with o
`size()`	returns the number of elements in list
`indexOf(o)`	returns the first index where o is found, or -1
`lastIndexOf(o)`	returns the last index where o is found, or -1
`clear()`	removes all

Linked List

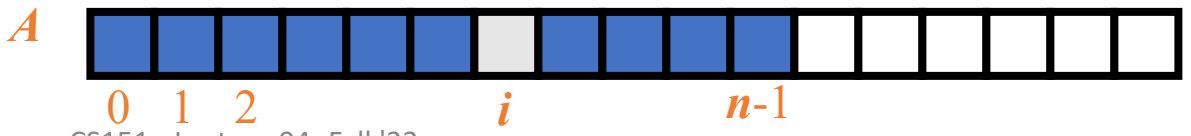
Arrays and Lists

Arrays

- Fixed size – once created, can not be resized
- Contiguous memory allocation
- Fast random access – $O(1)$
- Slow insertion and deletion – $O(n)$ must shift the rest

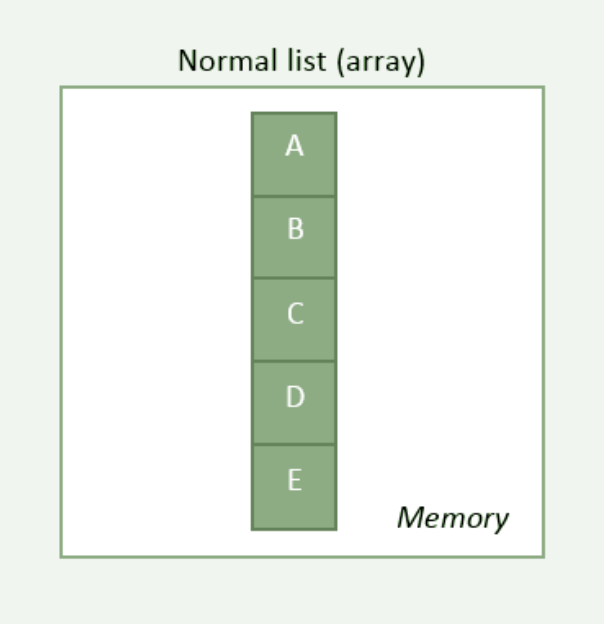
Lists

- Dynamic size:
  - grows and shrinks
- Non-contiguous memory allocation
- Slow random lookup – $O(n)$ must traverse
- Fast insertion and deletion – $O(1)$

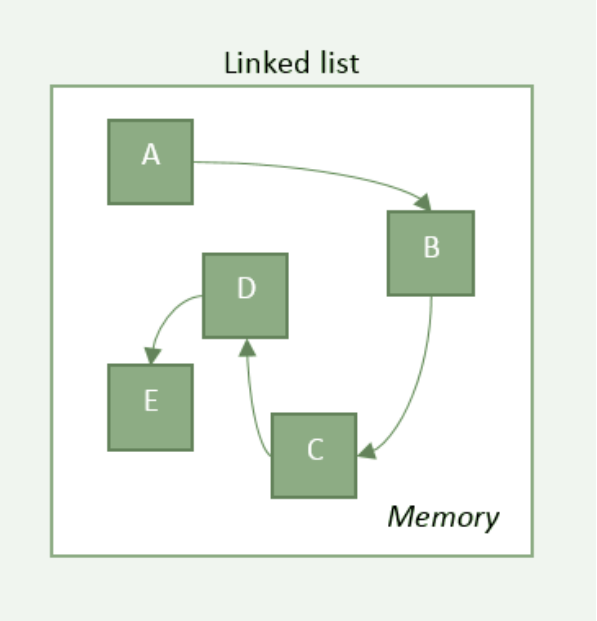


List versus Array - memory

An array is a single consecutive piece of memory

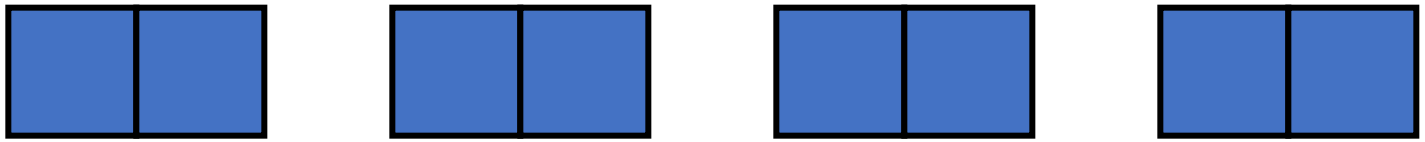


A list can be made of many disjoint pieces



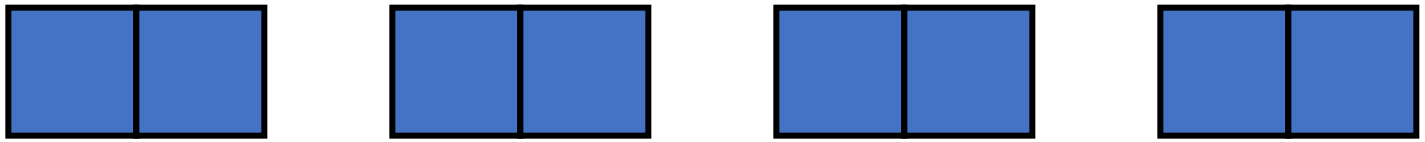
Linked List

- A linked list is a lists of objects
- The objects form a linear sequence
- Linked lists are typically unbounded, that is, they can grow infinitely.



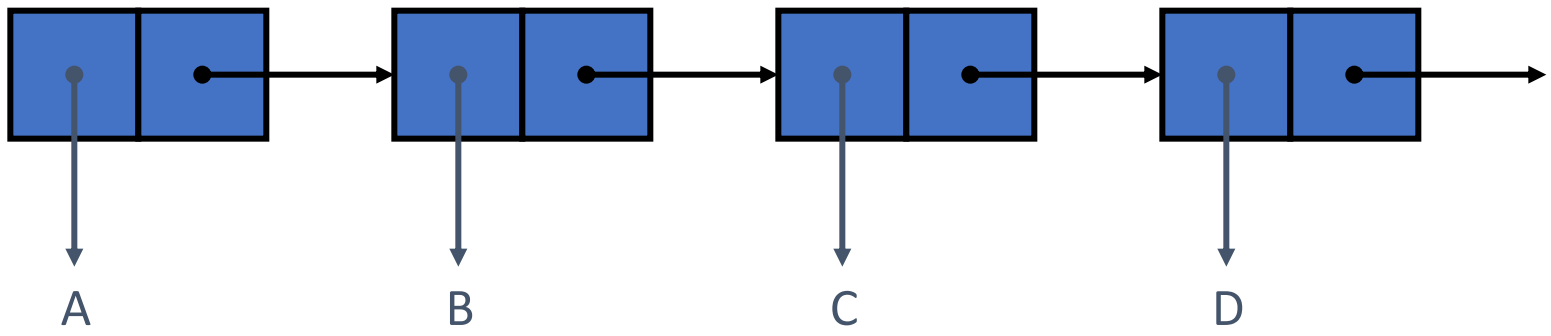
Linked List

- A linked list is a lists of objects (nodes)
- The nodes form a linear sequence
- Linked lists are typically unbounded, that is, they can grow infinitely.



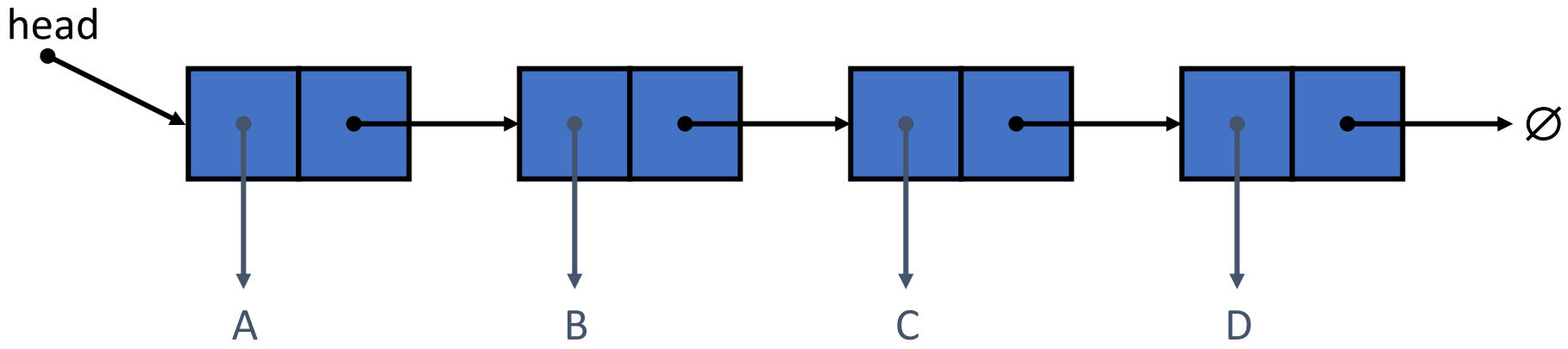
Linked List

- A linked list is a lists of objects (nodes)
- The nodes form a linear sequence.
- Linked lists are typically unbounded, that is, they can grow infinitely.



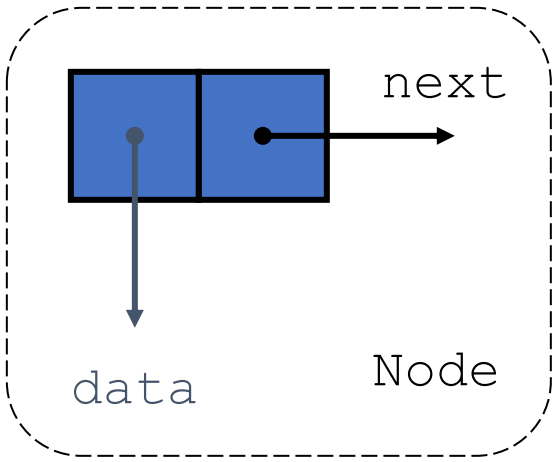
Linked List

- A linked list is a lists of objects (nodes)
- The nodes form a linear sequence.
- Linked lists are typically unbounded, that is, they can grow infinitely.



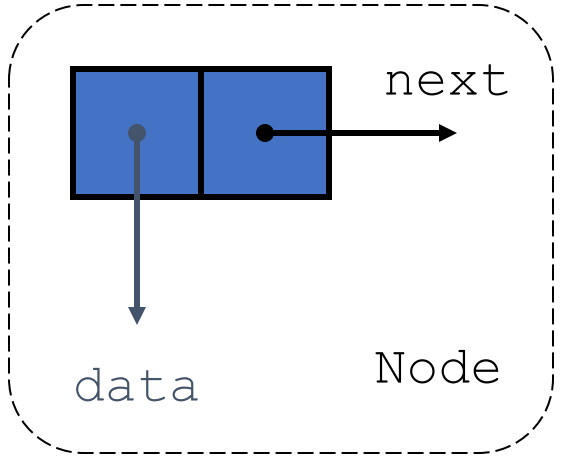
A node

```
public class Node<T> {
   private T data;
   private Node next;
}
```



Self-referential Structures

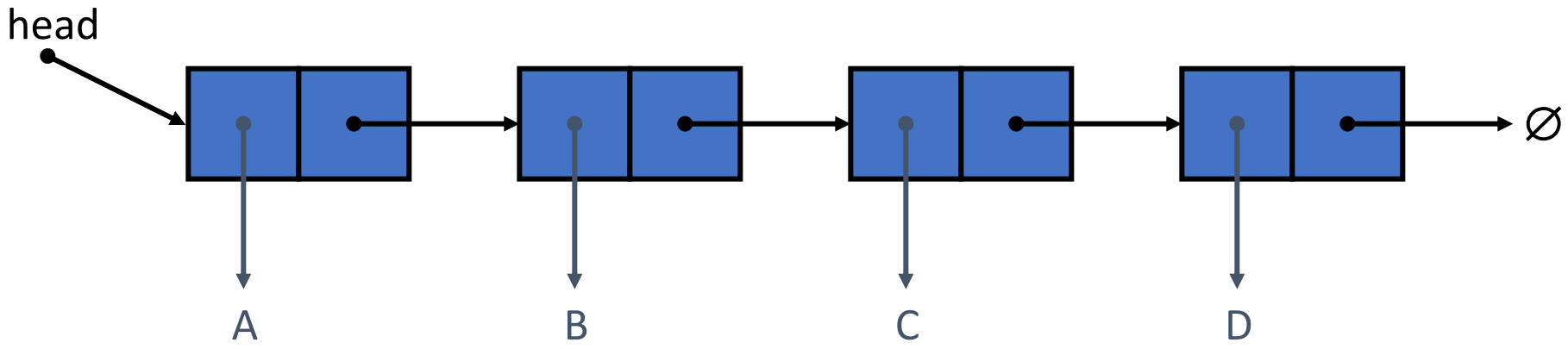
- A class with instance variables that reference itself



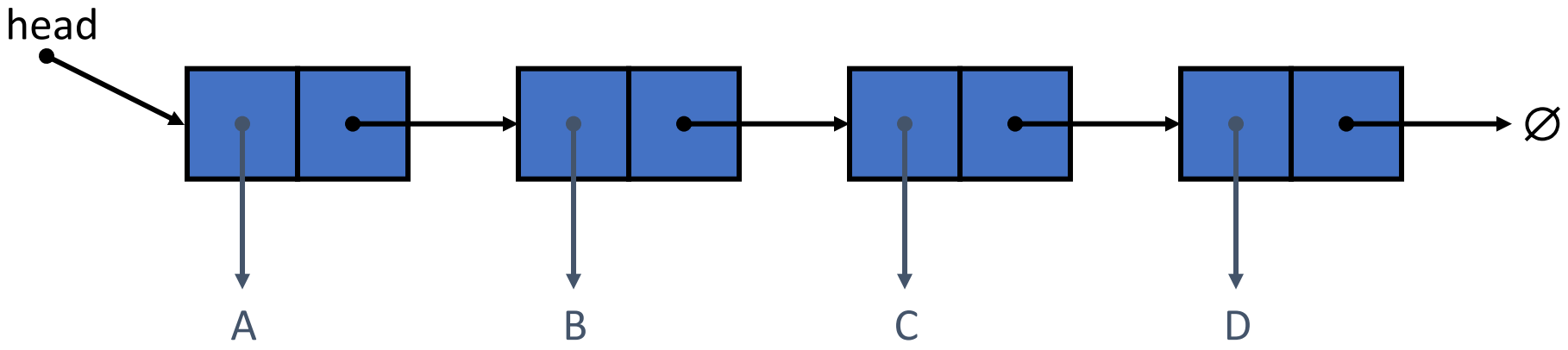

```
public class Node<T> {
  private T data;
  private Node next;
}
```

Linked List

- A linked list is a lists of objects (nodes)
- The nodes form a linear sequence.
- Linked lists are typically unbounded, that is, they can grow infinitely.



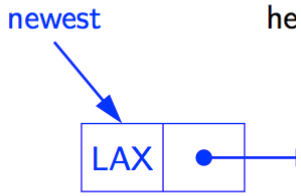
Inserting into a LinkedList



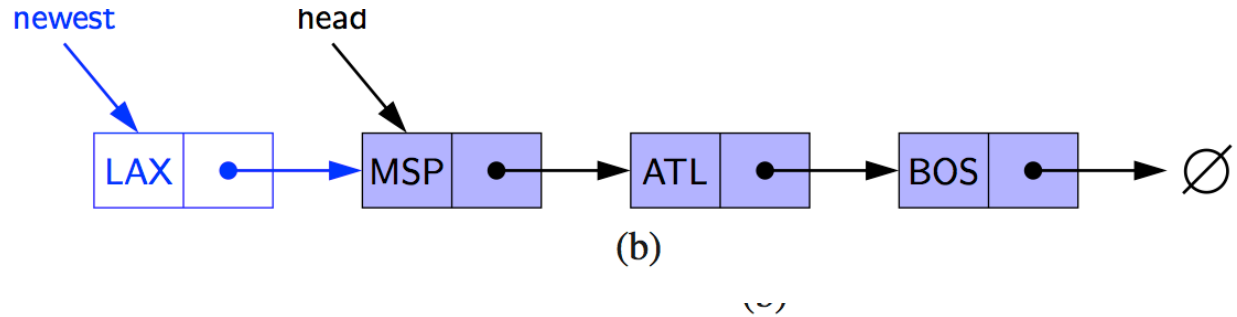
- Inserting in the beginning (tail)
- Inserting at the end (head)
- Inserting before/after a specific node

Inserting at the Head

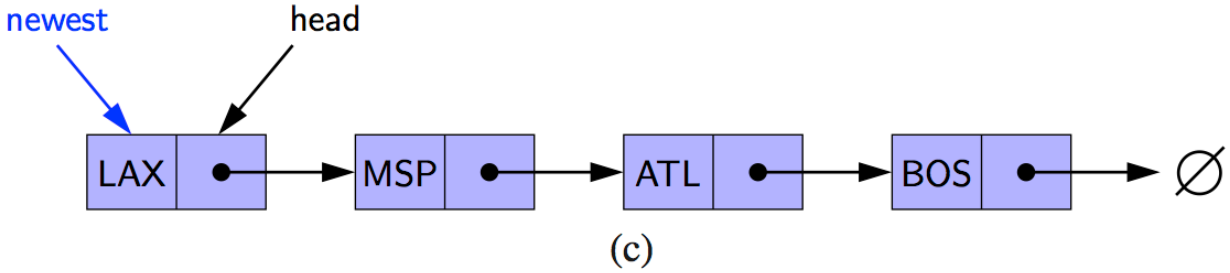
1. create a new node

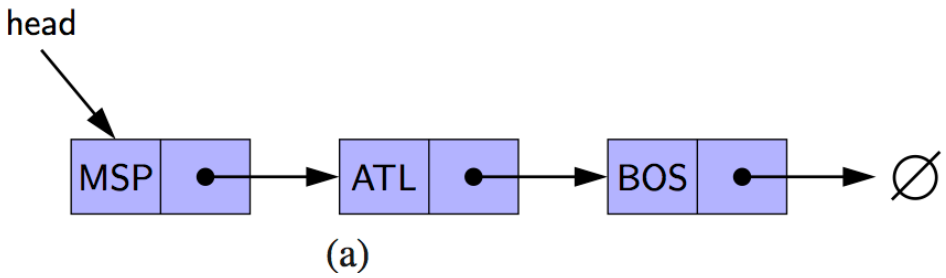


2. have new node point to old head



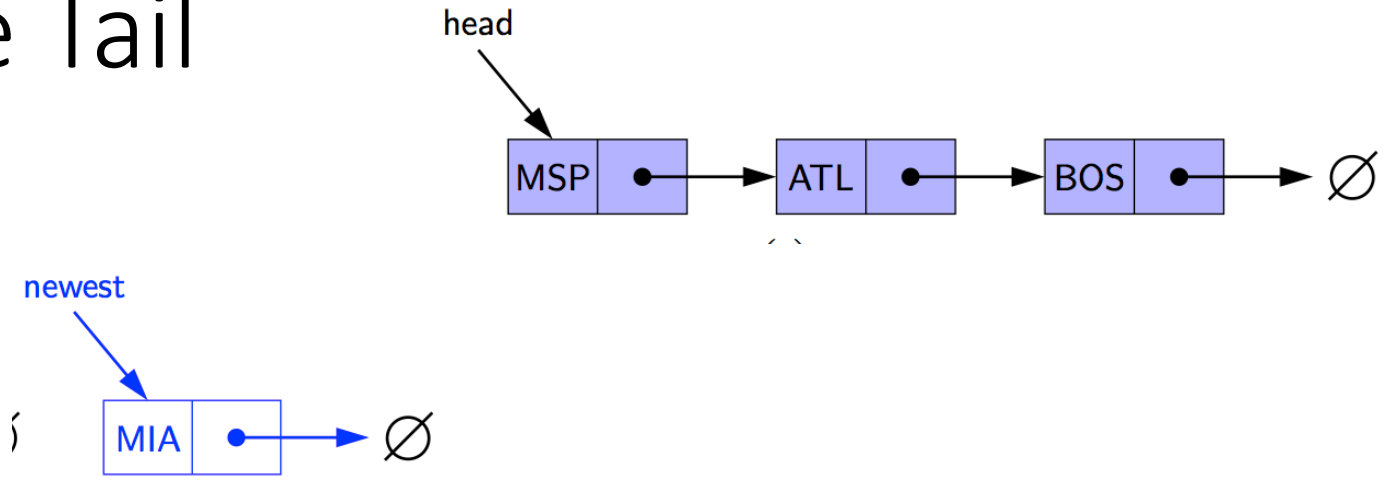
3. update head to point to new node





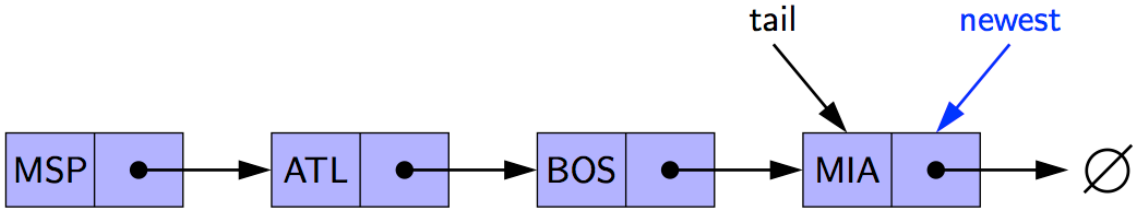
Inserting at the Tail

1. create a new node and have it point to null



2. have old last node point to new node

3. update tail to point to new node

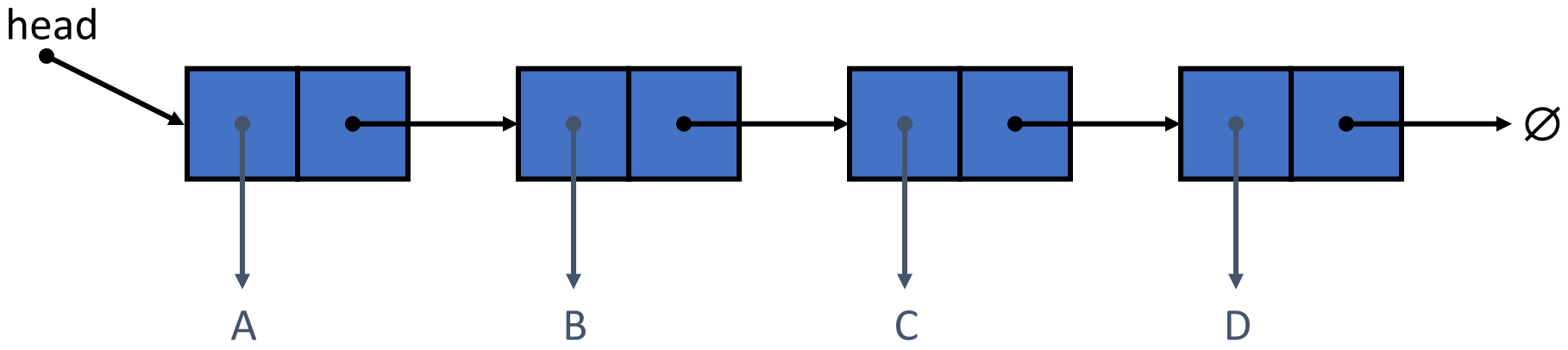


Insertion

```
public void addLast(T data) {
  Node newest = new Node(data, null);
  if (isEmpty()) { head = newest;}
  else {tail.setNext(newest);}
  tail = newest; size++;
 }

 public void addFirst(T data) {
  // exercise
 }
public void addAfter(T data, T after) {
  // exercise
 }
```

Find in a linked list

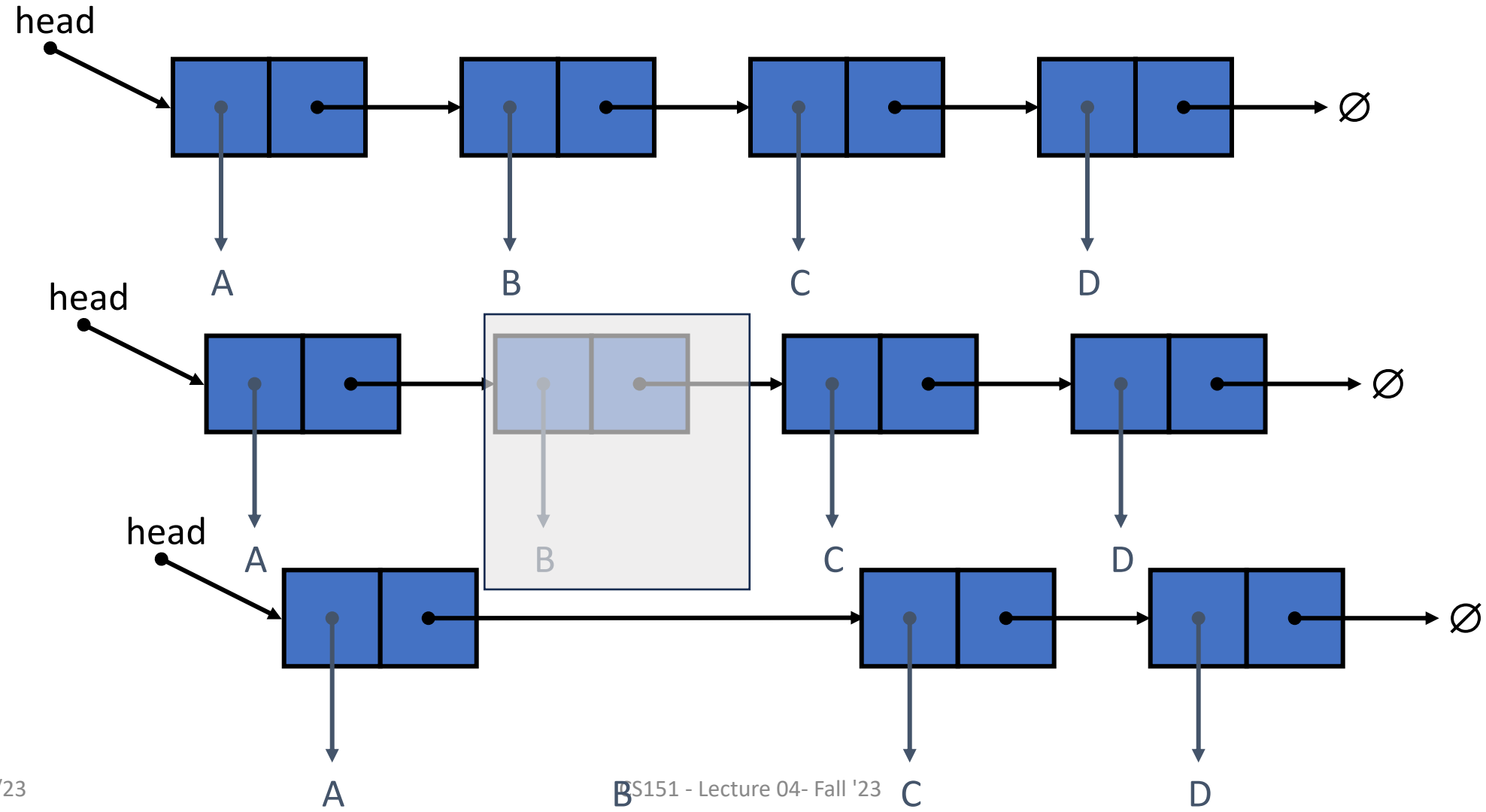


- Check if the head node is what you are looking for
- Iterate through nodes:
  - Stop when found
  - Otherwise return null

Find

```
public T find(T data) {
  Node n = head;
  while (n!=null){
    if (n.getData.equals(name)) {
      return n.getData();
    }
    n=n.getNext();
  }
  return null;
}
```

Removing from a LinkedList



Properties in LinkedList

What do we need to keep track of?

- Head
- Tail (optional)
- Number of elements (optional)
- Is empty (optional)

Nested Class

A class defined inside the definition of another class

When defining a class that is strongly affiliated with another

- help increase encapsulation and reduce undesired name conflicts.

Nested classes are a valuable technique when implementing data structures

- represent a small portion of a larger data structure
- an auxiliary class that helps navigate a primary data structure

Nested Node

```
public class LinkedList {

  private static class Node<E> {
    private E element;
    private Node<E> next;
    public Node(E element, Node<E> next) {
      this.element = element;
      this.next = next;
    }
    public E getElement() {return element;}
    public Node<E> getNext() {return next;}
    public void setNext(Node<E> n) {next = n;}
  }
```