CS206 Data Structures

Lists & Iterators

Sung-eui Yoon (윤성의)

Department of Computer Science
KAIST

http://sglab.kaist.ac.kr/~sungeui

Class Objectives (Ch. 7.2 & 7.3)

- ☐ Can implement doubled linked list
- ☐ Access lists with iterators

Position ADT

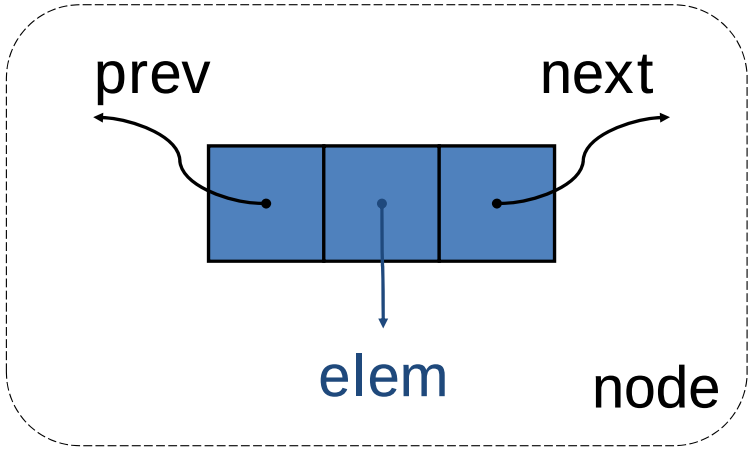
- The Position ADT models the notion of place within a data structure where a single object is stored
- It gives a unified view of diverse ways of storing data, such as
  - a cell of an array
  - a node of a linked list
- Just one method:
  - object element(): returns the element stored at the position

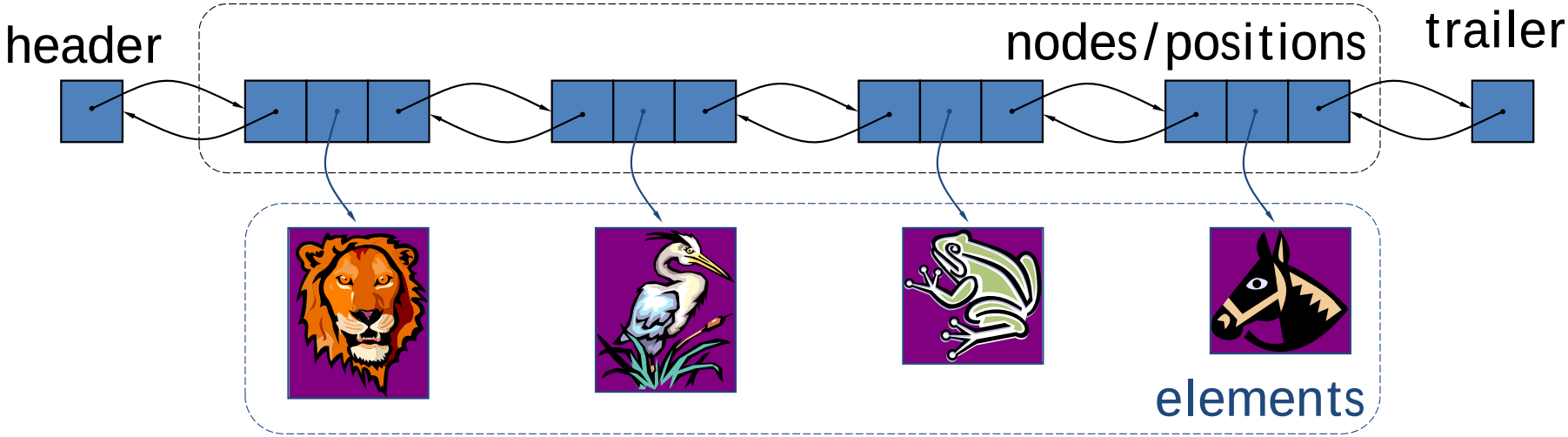
(Node) List ADT

- The List ADT models a sequence of positions storing arbitrary objects
- It establishes a before/after relation between positions
- Generic methods:
  - size(), isEmpty()
- Accessor methods:
  - first(), last()
  - prev(p), next(p)
- Update methods:
  - replace(p, e)
  - insertBefore(p, e), insertAfter(p, e),
  - insertFirst(e), insertLast(e)
  - remove(p)
- Note: we have already covered singly linked list

Doubly Linked List

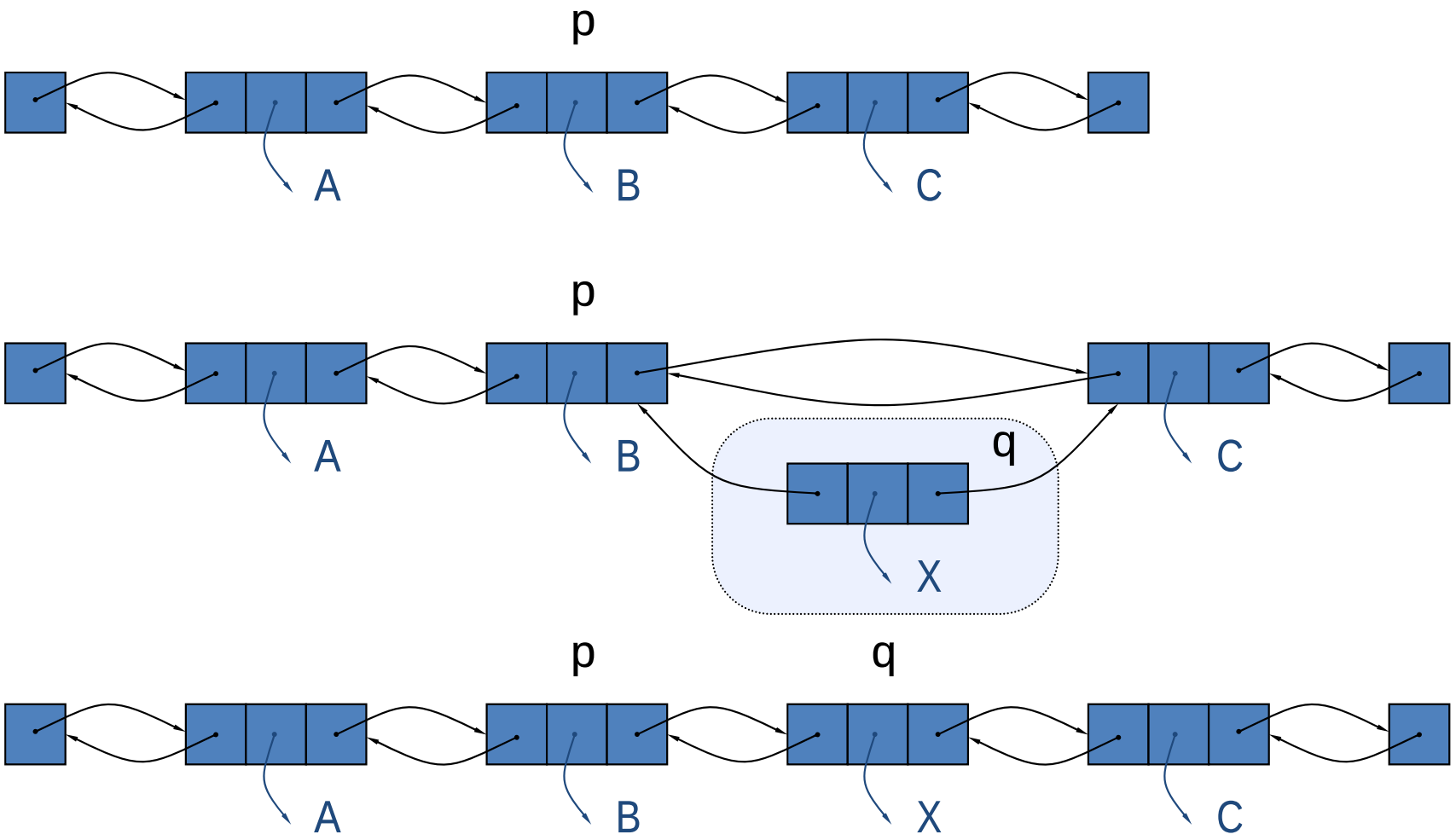
- A doubly linked list provides a natural implementation of the List ADT
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes





Insertion

- We visualize operation insertAfter(p, X), which returns position q

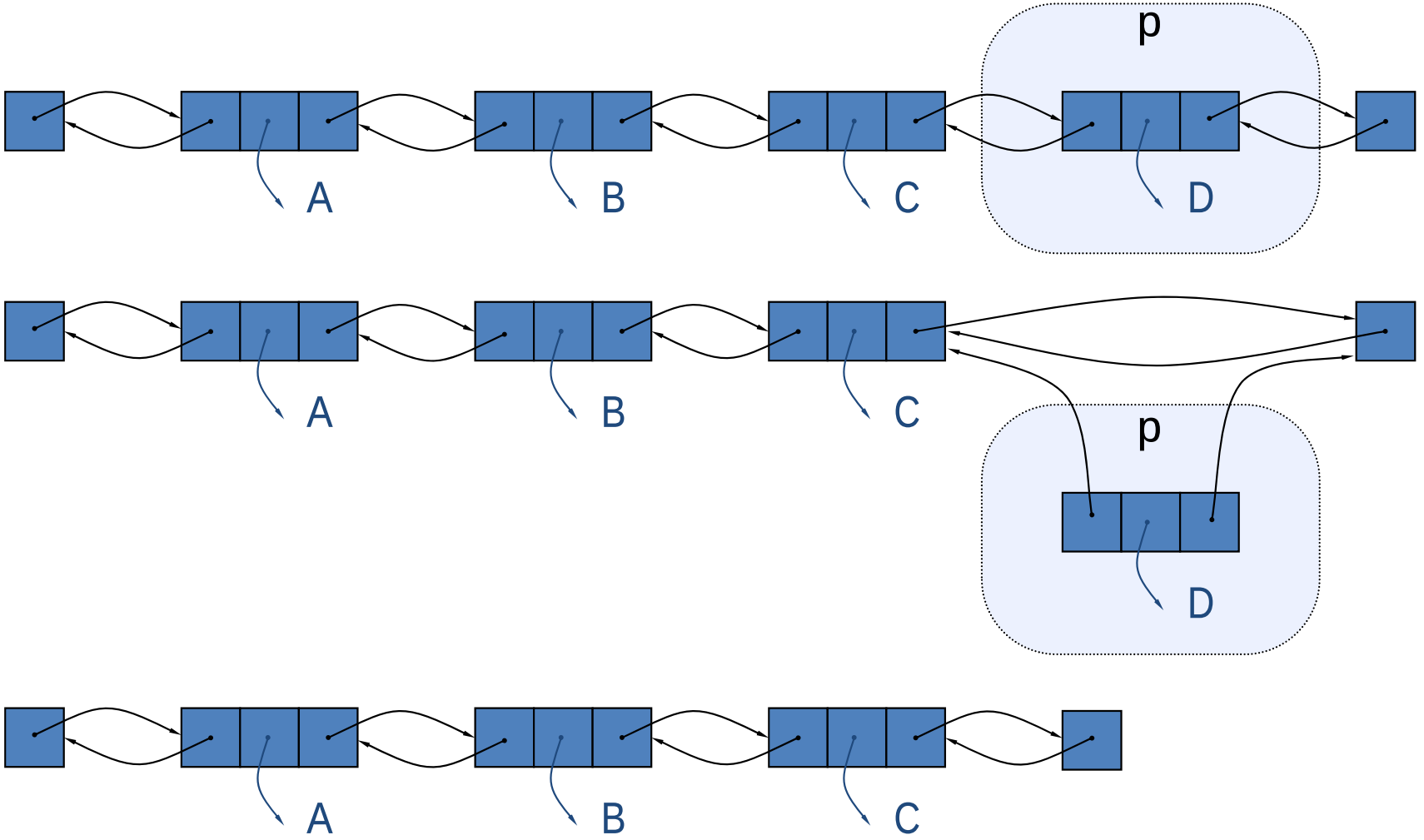


Insertion Algorithm

```
Algorithm insertAfter(p,e):
    Create a new node v
    v.setElement(e)
    v.setPrev(p)   {link v to its predecessor}
    v.setNext(p.getNext())  {link v to its successor}
    (p.getNext()).setPrev(v){link p's old successor to v}
    p.setNext(v)   {link p to its new successor, v}
    return v    {the position for the element e}
```

Deletion

☐ We visualize remove(p), where p = last()



Deletion Algorithm

```
Algorithm remove(p):
    t = p.element {a temporary variable to hold the
          return value}
    (p.getPrev()).setNext(p.getNext())    {linking out p}
    (p.getNext()).setPrev(p.getPrev())
    p.setPrev(null)    {invalidating the position p}
    p.setNext(null)
    return t
```

Performance

- In the implementation of the List ADT by means of a doubly linked list
  - The space used by a list with n elements is O(n)
  - The space used by each position of the list is O(1)
  - All the operations of the List ADT run in O(1) time
  - Operation element() of the Position ADT runs in O(1) time

The Iterator ADT

- The Iterator ADT abstracts the process of scanning through a collection of elements one at a time
  - Sequence S
  - Current element in S
  - A way of stepping to the next element in S (and making it the current element)
- Main array list operations:
  - hasNext(): test whether there are elements left in the iterator
  - next(): return the next element in the iterator
- Extends the concept of Position ADT by adding a traversal capability
- Typically associated with another data structure

java.lang.Iterable

- Only one method defined in the interface
  - iterator(): return an iterator of the elements in the collection
- Using the interface:

```
public interface PositionList<E> extends Iterable<E>
{
    // all other methods of the list ADT
    /** returns an iterator of all the elements in the list */
    public Iterator<E> iterator();
}
```

```
 /** Returns a textual representation of a given node list */
    public static <E> String toString(PositionList<E> l)
    {
        Iterator<E> it = l.iterator();
        String s = "[";
        while (it.hasNext())
        {
            s += it.next(); // implicit cast of the next element to String
            if (it.hasNext())
                s += ", ";
        }
        s += "]";
        return s;
    }
```

Implementing Iterator

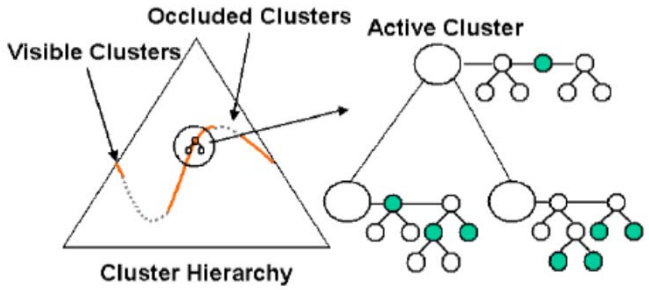
☐ You rarely need to implement iterator when using JDK

```
public class ElementIterator<E> implements Iterator<E>
{
    protected PositionList<E> list; // the underlying list
    protected Position<E> cursor; // the next position

    /** Creates an element iterator over the given list. */
    public ElementIterator(PositionList<E> L)
    {
        list = L;
        cursor = (list.isEmpty()) ? null : list.first();
    }

    public boolean hasNext()
    {
        return (cursor != null);
    }

    public E next() throws NoSuchElementException
    {
        if (cursor == null)
            throw new NoSuchElementException("No next element");
        E toReturn = cursor.element();
        try
        {
            cursor = (cursor == list.last()) ? null : list.next(cursor);
        } catch (Exception e)
        {
            cursor = null;
        }
        return toReturn;
    }
```

```
 /** Returns an iterator of all the elements in the list. */
  public Iterator<E> iterator() { return new ElementIterator<E>(this); }
```

An Example of Using Lists

- View-dependent rendering
  - Yoon et al., IEEE Visualization, '04
  - Received a best paper nomination, top-5



Class Objectives were:

- ☐ Can implement doubled linked list
- ☐ Access lists with iterators

Next Time

☐ Trees

☐ HW:

- Go over the next lecture slides before the class
- Just 10 min ~ 20 min should be okay

Any Questions?

- ☐ Come up with one question on what we have discussed in the class and submit at the end of the class
  - 1 for typical questions
  - 2 for questions with thoughts or that surprised me

- ☐ Write questions at least 4 times

- ☐ You can type at KLMS