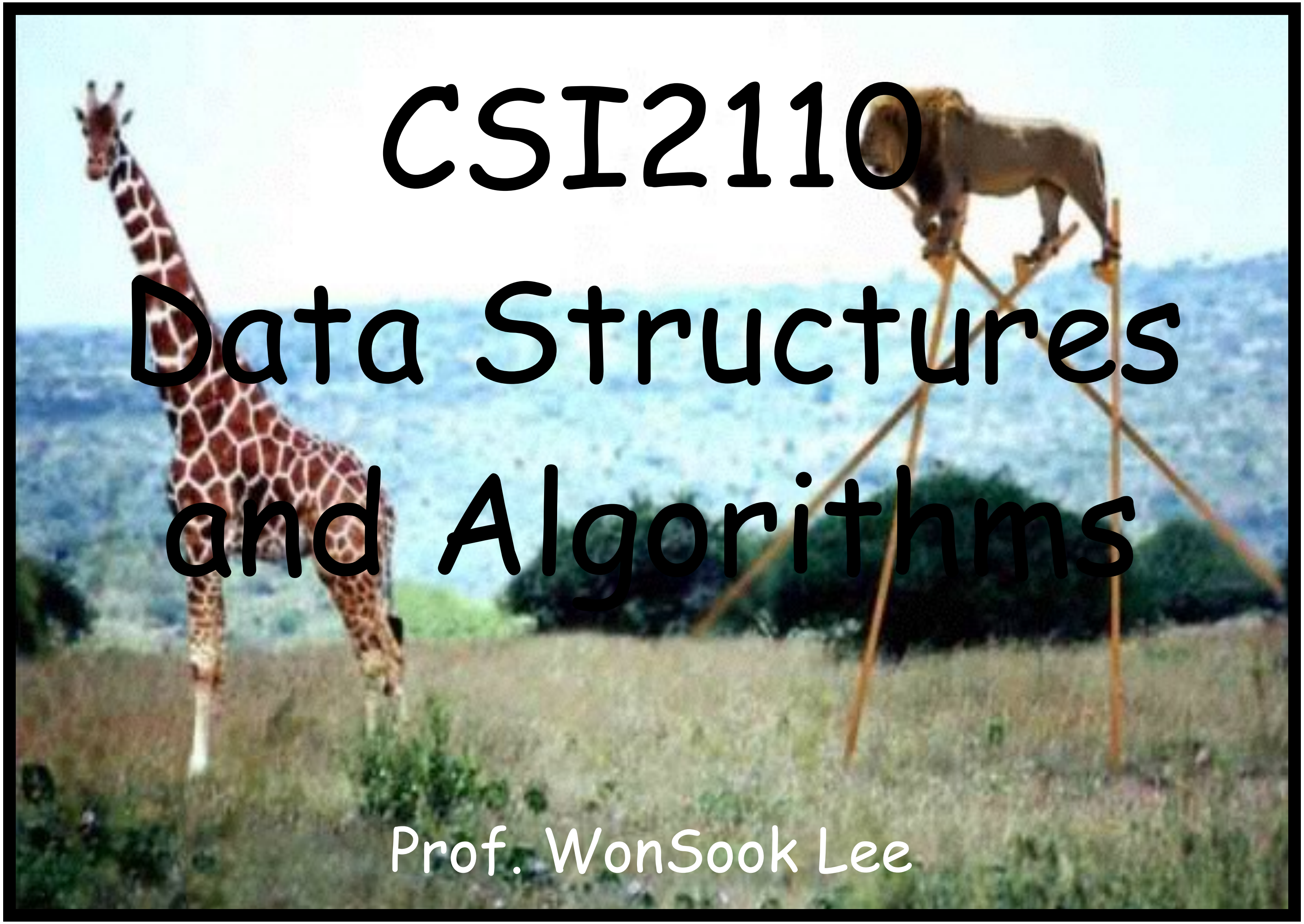


CSI2110 Data Structures and Algorithms

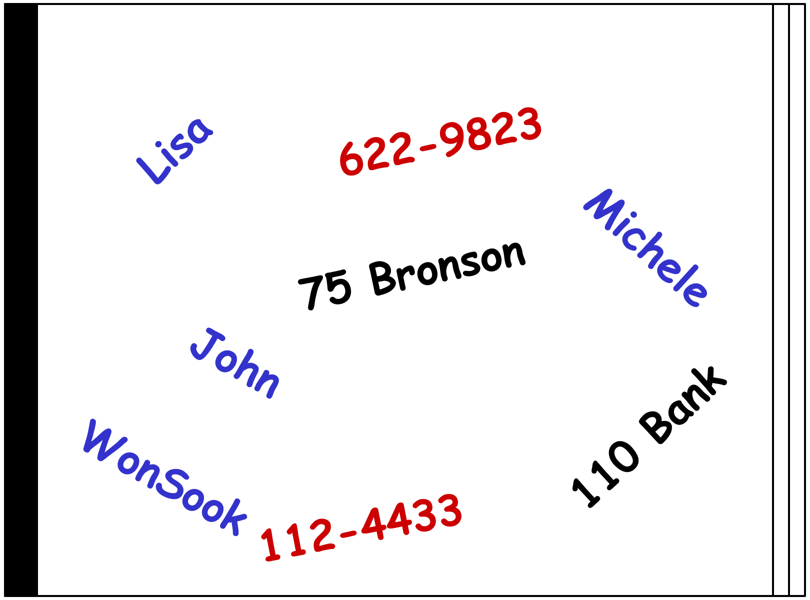
Prof. WonSook Lee

Data Structures ?

Example:
Electronic Phone Book

Contains different **DATA**:

- names
- phone number
- addresses



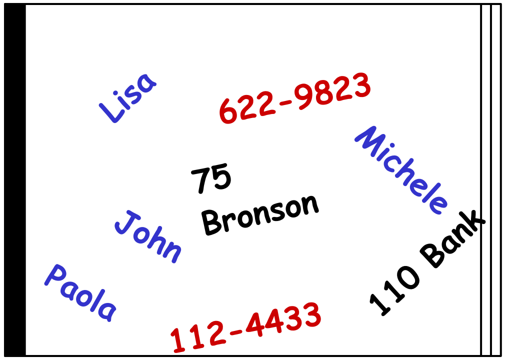
Need to perform certain **OPERATIONS**:

- add
- delete
- look for a phone number
- look for an address

How to organize the data so to optimize the efficiency of the operations

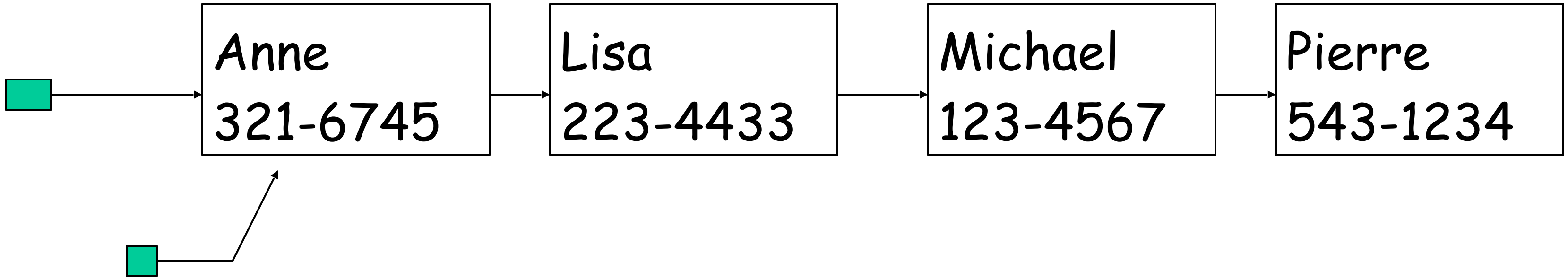
Data Structures ?

Example:



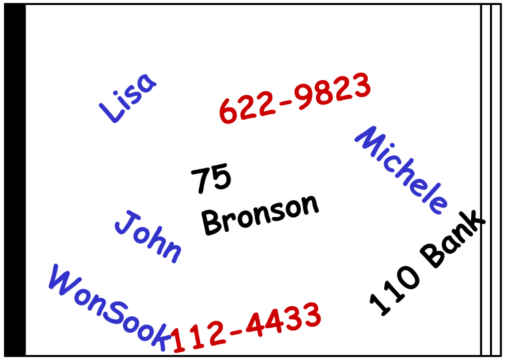
Lisa	Pierre	Michael	Anne
223-4433	543-1234	123-4567	321-6745

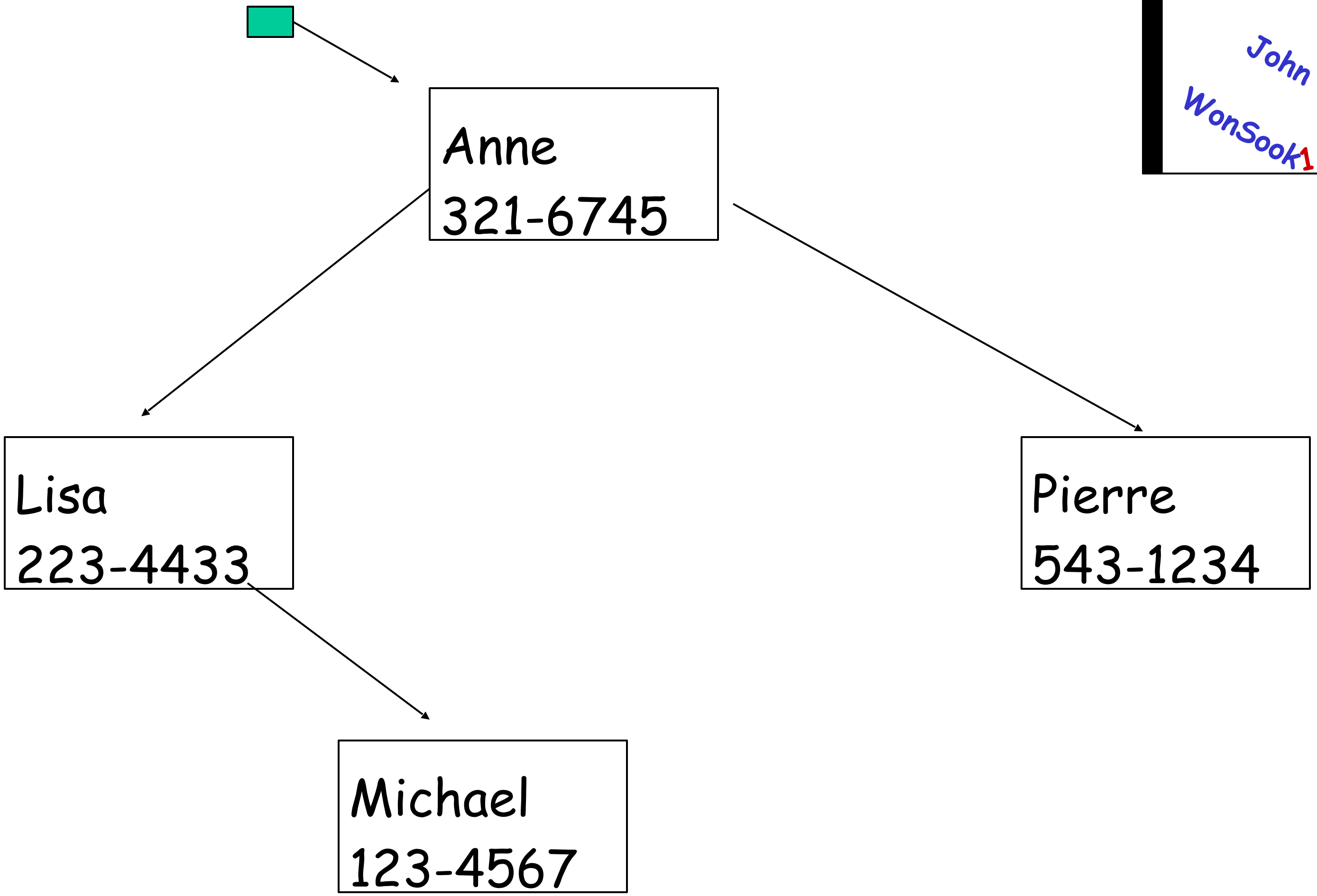
Anne	Lisa	Michael	Pierre
321-6745	223-4433	123-4567	543-1234



Data Structures ?

Example:





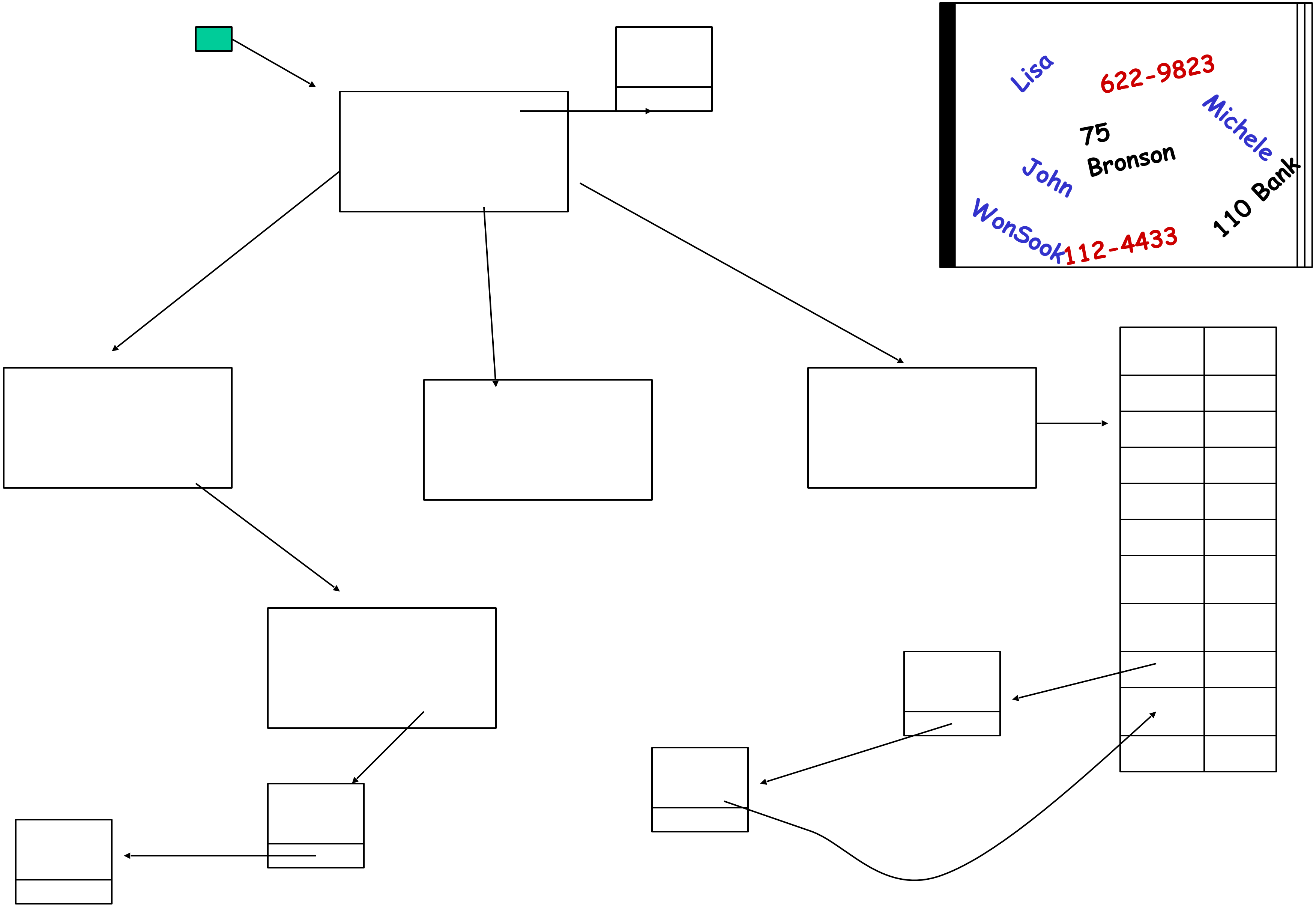
Anne
321-6745

Lisa
223-4433

Pierre
543-1234

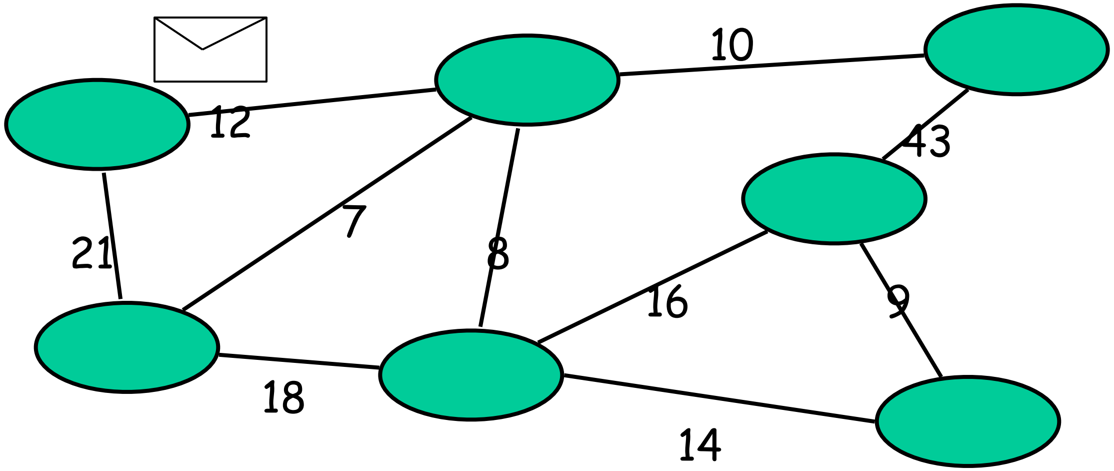
Michael
123-4567

Data Structures ?



Data Structures ?

Example:
Finding the best route
for an email message
in a network



Contains **DATA**:

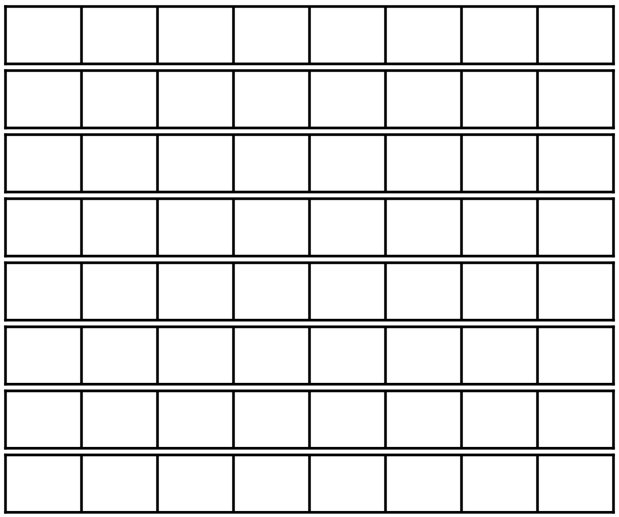
- network + traffic

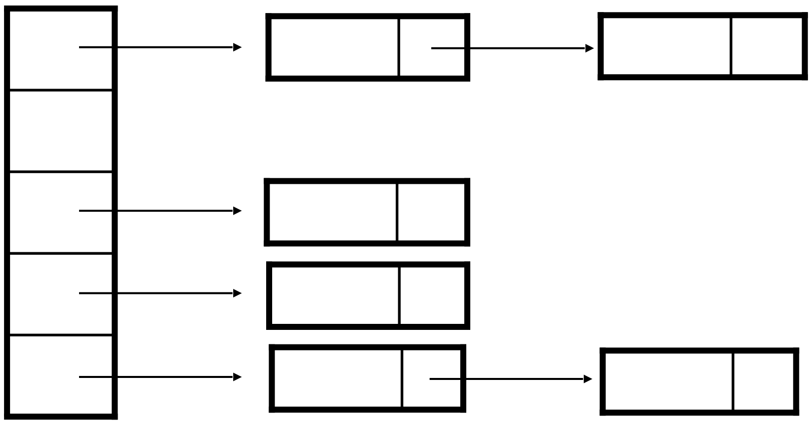
Need to perform certain **OPERATIONS**:

- Find best route

Data Structures ?

How to represent the data





so to perform the operations efficiently

Data Structures ?

Keep in mind the operations you need to perform

Choose the **best** structure for your data

Study different data structures

How to understand if a data structure is good

Objectives of the course

Present in a systematic fashion the most commonly used data structures, emphasizing their abstract properties.

Discuss typical algorithms that operate on each kind of data structure, and analyze their performance.

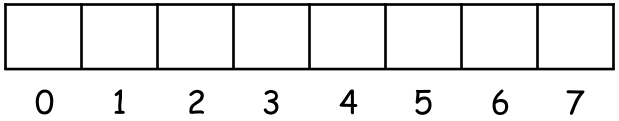
Compare different Data Structures for solving the same problem, and choose the

Overview of the course

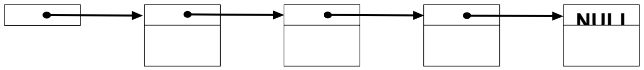
- Stacks, queues, deques
- Algorithm analysis techniques
- Vectors, Lists, Sequences
- Trees
- Heaps
- Dictionaries
- Search trees
- Tries
- Graphs
- Sorting

Review

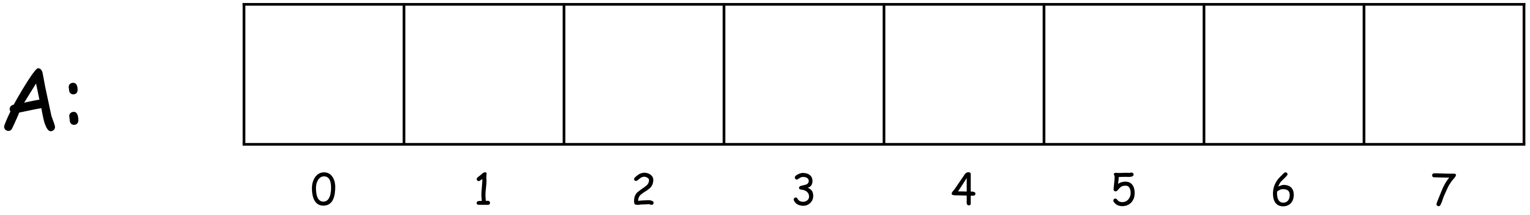
- Arrays



- Linked Structures



Array



Numbered collection of variables of the same type. Fixed length.

- **Static** structure
- Direct access

Insertion ?

Deletion ?

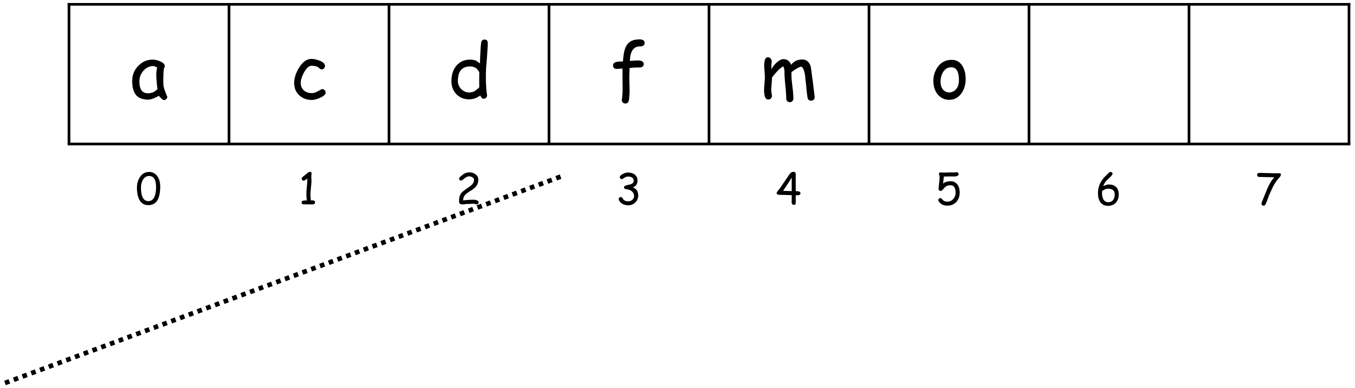
Array

a	c	d	m	o			
0	1	2	3	4	5	6	7

f

example of insertion in a sorted array

Array



a	c	d	f	m	o		
0	1	2	3	4	5	6	7

move “m” and “o”
to make room for “f”

example of insertion in a sorted array

Array

a	c	d	f	m	o	p	z
0	1	2	3	4	5	6	7

1) For insertions and deletions
elements MUST BE MOVED

2) What happens when the array is FULL ?

Operations on Arrays

```
void        addElement(int index,Element e)   // insert
Element     setElement(int index,Element e)
Element     getElement( int index )
Element     remove( int index )
int         size()
```

Supported set of operations on a data structure
define the interface of a data structure

Array Type

Review: Java implementation

```
public class ArrayType {
  protected Element[] array;
  public ArrayType(Element[] in_array) { …}
  public Element setElement(int index, Element e) {…}
  public Element getElement(int index) {…}
  public Element remove(int index ) {…}
  public void addElement(int index, Element e) {…}
  public int size() {…}
}
```

Method Implementations for Array Type I

Insertion

```
public void addElement(int index, Element e) {
  // Shift everything backward
  for (int i=array.length-1; i>index; --i ) {
    array[i] = array[i-1];
  }
  array[index] = e;
  return;
}
```

Method Implementations for Array Type II

Removal:

```
public Element remove(int index ) {
  Element retVal = array[index];
  // Shift everything forward
  for ( int i=index; i<array.length-1; ++i ) {
    array[i] = array[i+1];
  }
  array[array.length-1] = null;
  return retVal;
}
```

Using Array Type

```
ArrayType at = new ArrayType(new Element[10]);
at.setElement(0, new Element("CSI1120", 2008));
at.setElement(1, new Element("CSI1121", 2008));
at.setElement(2, new Element("CSI1110", 2008));
at.addElement(0, new Element("CSI2110", 2009));
    for ( int i=0; i<at.size(); ++i ) {  // print all elements
  Element e = at.getElement(i);
  if ( e == null ) break;
  System.out.println( e );
}
// remove first element
at.remove(0);
```

Storing Elements is fine

What about Apples and Oranges?

Java – Use Object!

```
public class ArrayType {
  protected Object[] array;
  public ArrayTypeObject(Object[] in_array) {…}
  public Object setElement(int index, Object e) {…}
  public Object getElement(int index) {…}
  public Object remove(int index ) {…}
  public void addElement(int index, Object e) {…}
  public int size() {…}
}
```

Change class and methods to use the universal parent class Object.

Now we can store Elements, Apples and Oranges!

Type Safety?

Generics

```
public class ArrayType<E> {
  protected E[] array;
  public ArrayType(E[] in_array) {…}
  public E setElement(int index, E e) {…}
  public E getElement(int index) {…}
  public E remove(int index ) {…}
  public void addElement(int index, E e) {…}
  public int size() {…}
}
```

Now we can store Elements, Apples and Oranges and we have type safety!

```
ArrayType<Element> at = new ArrayType<Element>(new Element[10]);
ArrayType<Apple> at = new ArrayType<Apple>(new Apple[10]);
ArrayType<Orange> at = new ArrayType<Orange>(new Orange[10]);
```

Other Elements

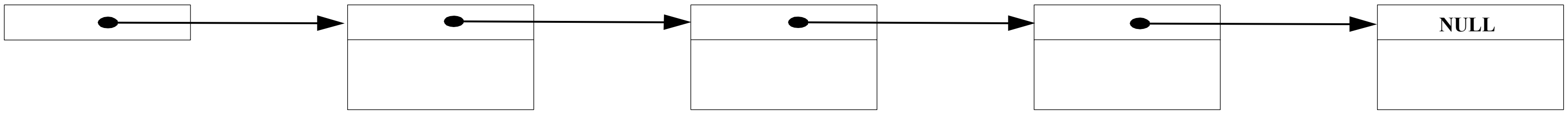
Solution A: Make Element an Interface and force all user of ArrayType to implement it

Solution B: Use the Java Object class as parent class of all classes.

```
void addElement(int index,Object e)   // insert
Object setElement(int index,Object e)
Object getElement( int index )
Object remove( int index )
int size()
```

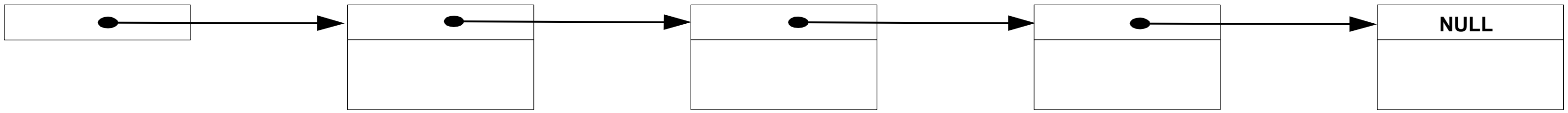
Supported set of operations on a data structure define the data structure

Linked Structures



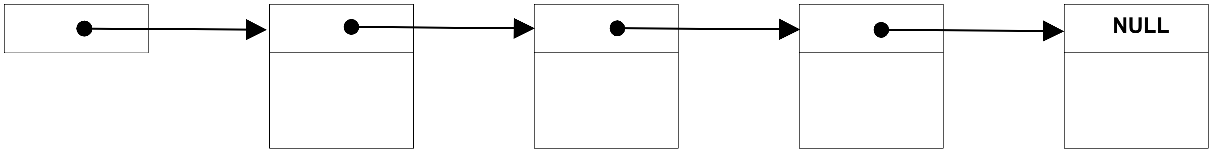
- **Dynamic** structure
- Sequential access
- Insertion and deletion occur without moving elements

Linked Structures



- **Dynamic** structure
- Sequential access
- Insertion and deletion occur without moving elements

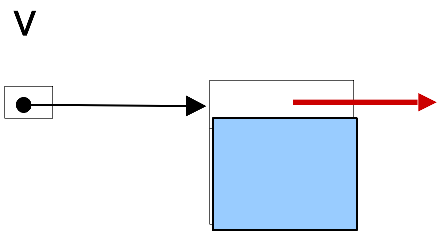
Single Linked Lists



Node v

Object element

Node next



Java Implementation - Singly Linked List

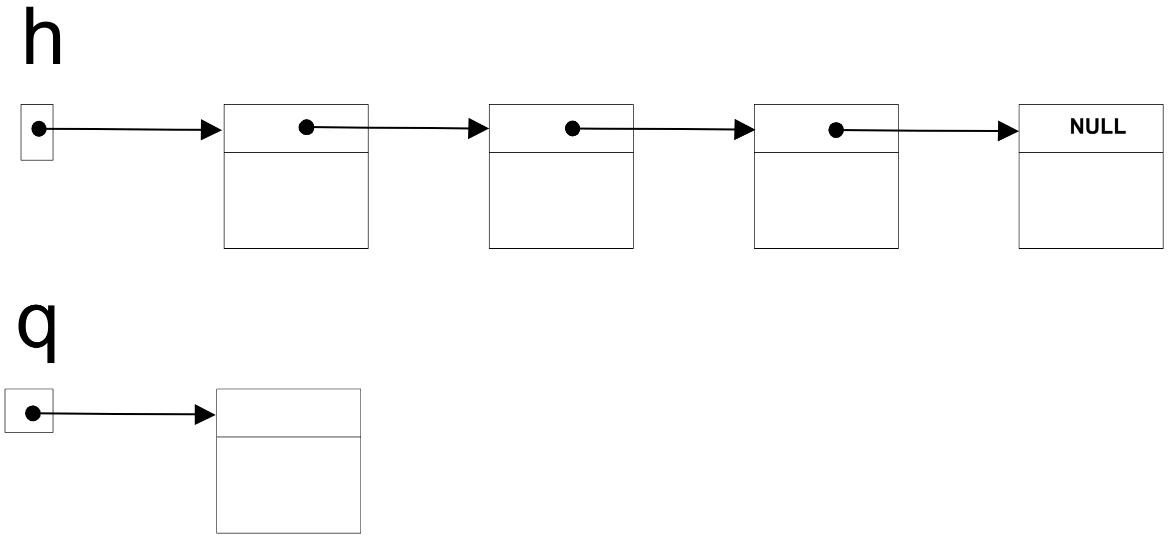
you will review it in the first Lab

Usual Methods (see textbook):

- void setElement(Object e)
- void setNext(Node newNext)
- Object getElement()
- Node getNext()

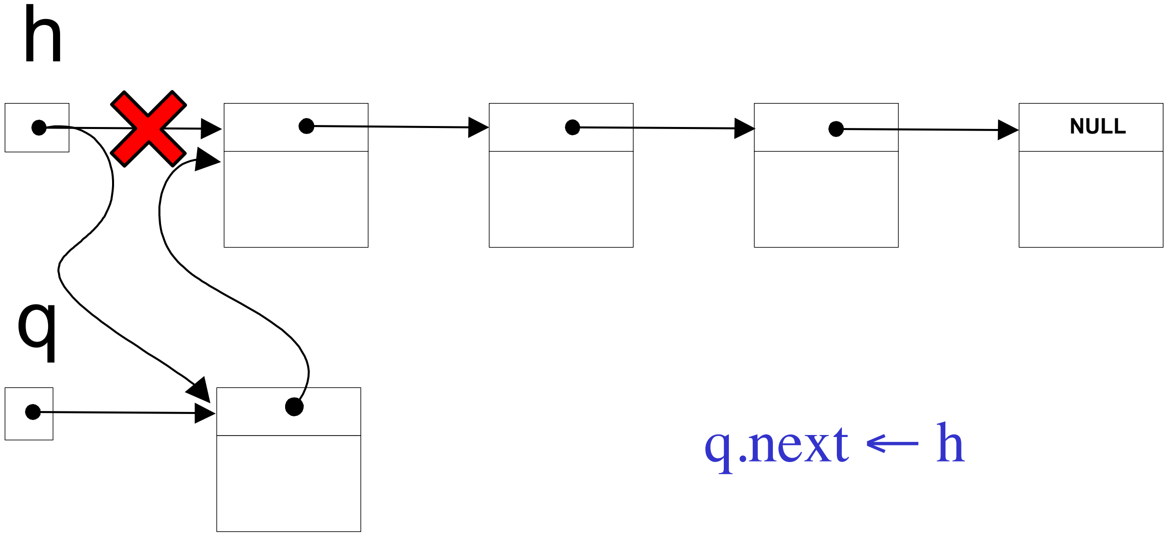
Insertion

Original configuration:



Goal: to insert the element q into list h.

Insertion at the beginning



```
q.next ← h
h ← q
```

(easy)

… we are using pseudocode …

pseudocode

q.next ← h

variable **q.next** gets the value of variable **h**

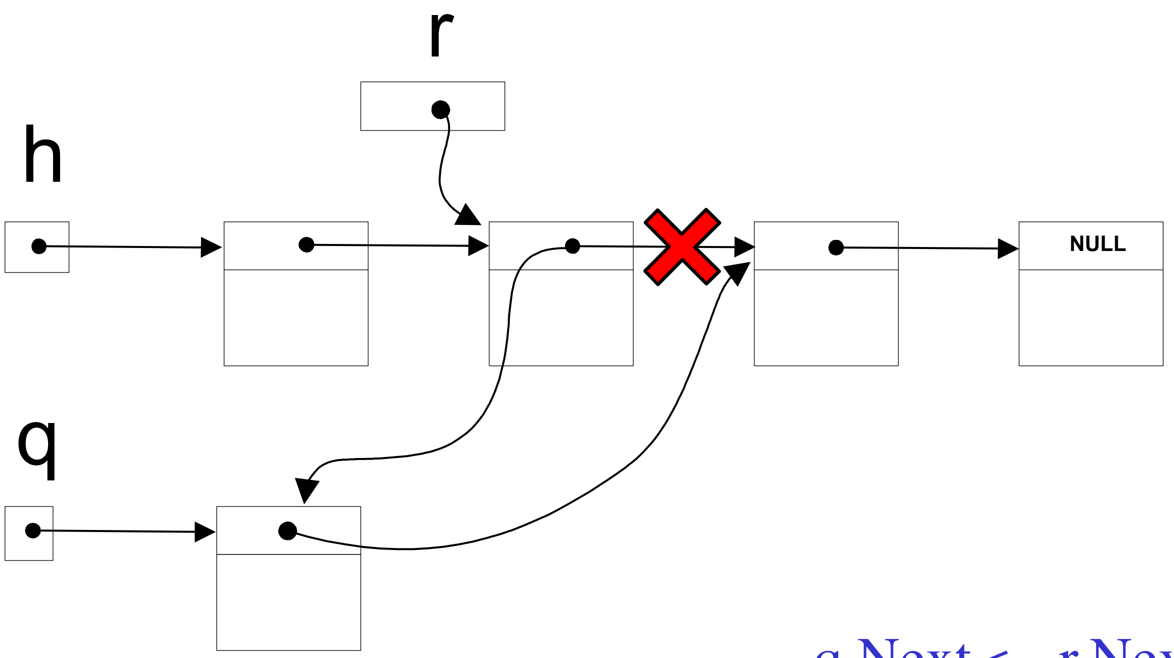
(q.next:= h)

pseudocode

```
$h \leftarrow q$
```

variable **head** gets the value of variable **q**

Insertion after r

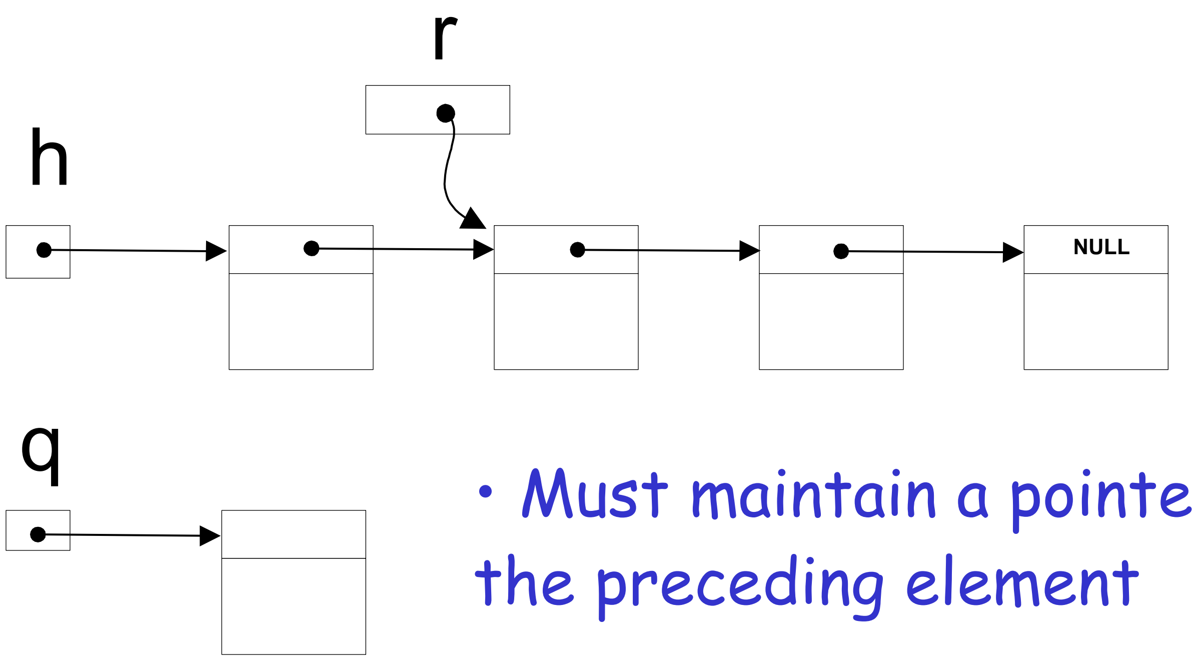


q.Next ← r.Next

r.Next ← q

(easy)

Insertion before r



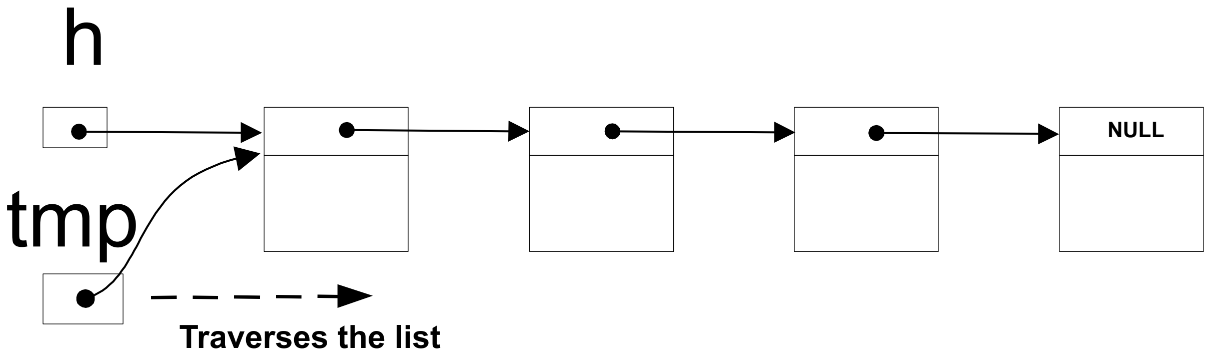
(more difficult)

- Must maintain a pointer to the preceding element

or

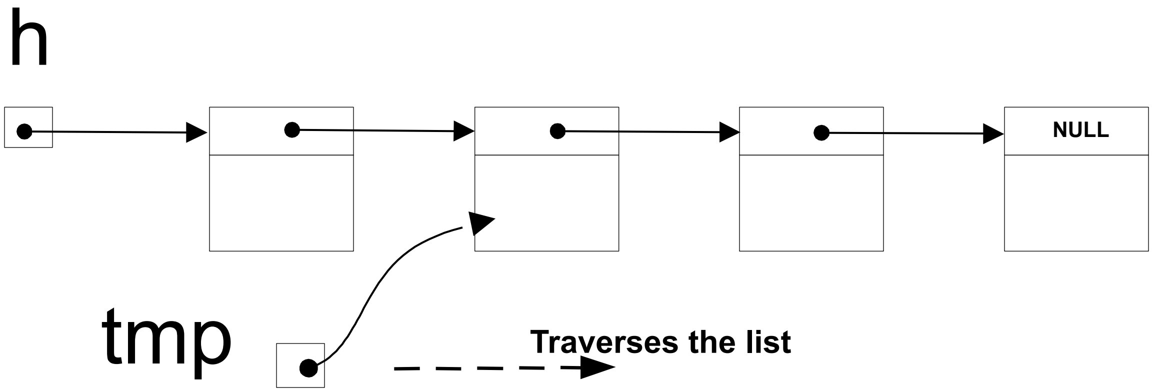
- Exchange the contents pointed to by r and q, and insert q after r.

Search



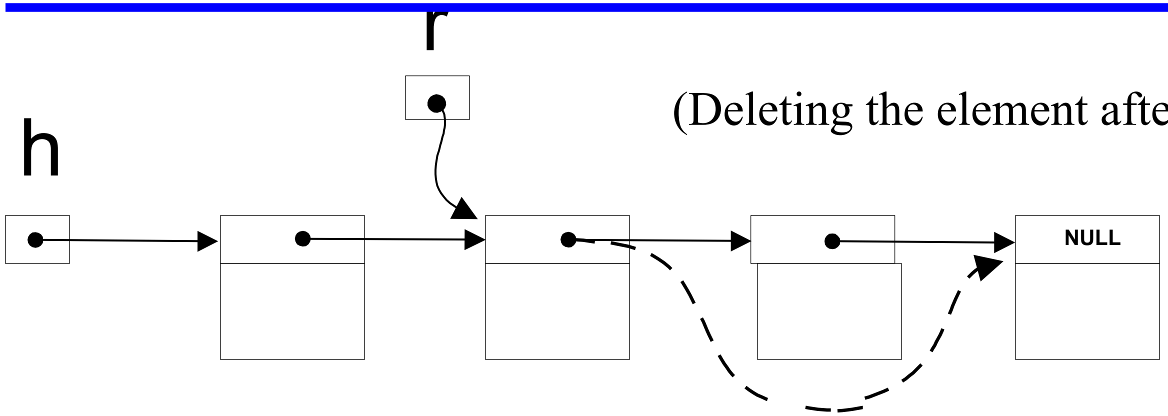
```
Node tmp;
tmp ← p;
while (tmp != null ) {
    if tmp .element  is  ce-que-je-recherche {
        return tmp ; }
    else
```

Search



```
Node tmp;
tmp ← firstnode;
while (tmp != null ) {
    if tmp .element  is  ce-que-je-recherche {
       return tmp n; }
   else {tmp ← tmp .next; }
  }
  return tmp ;
```

Deletion

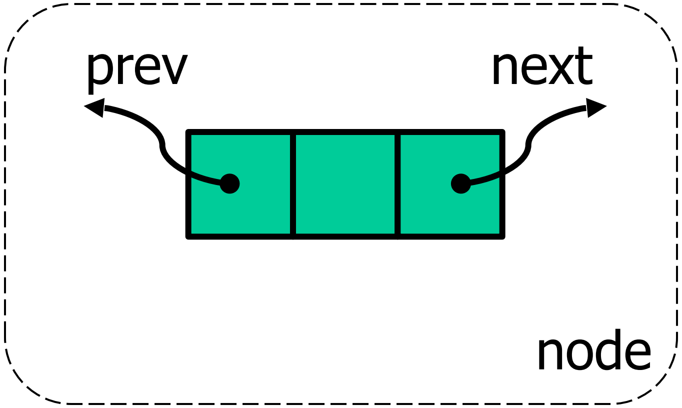


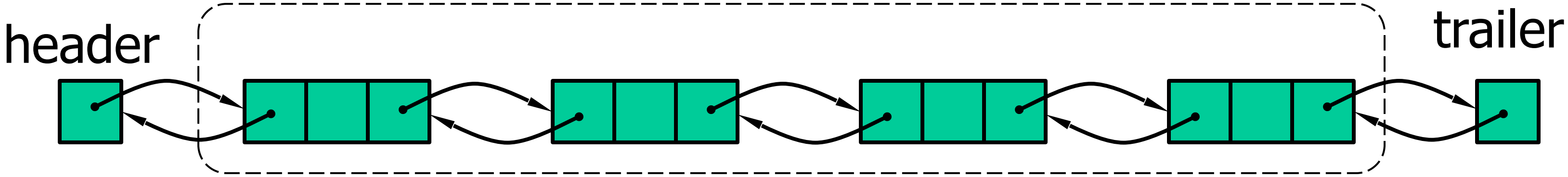
(Deleting the element after r)

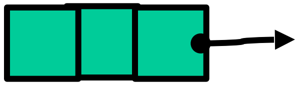
First element (easy)	h ← h.Next
Element after r (easy)	r.Next ← r.Next.Next
Element at r (difficult)	• Use a pointer to the preceding element, or • Exchange the contents of the element at r with the contents of the element following r, and delete The element after r. **Very difficult if r points to the last element!

Doubly Linked List

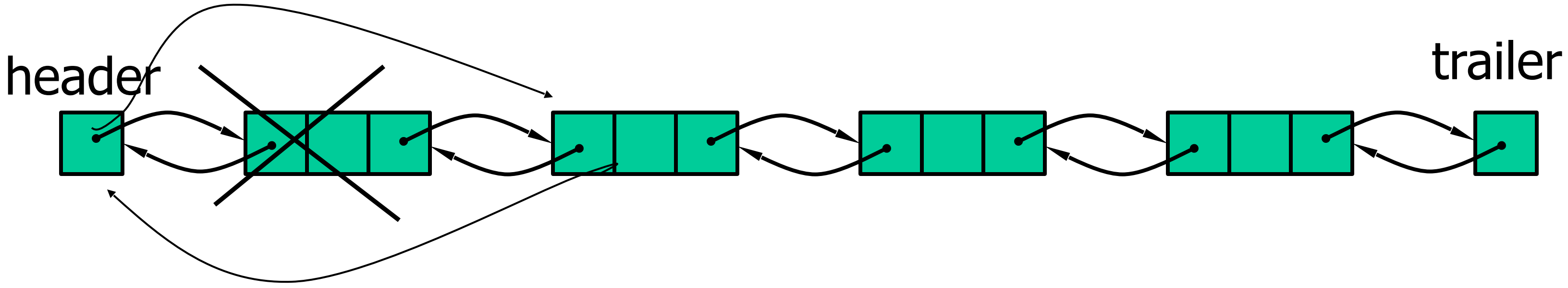
- Nodes store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes







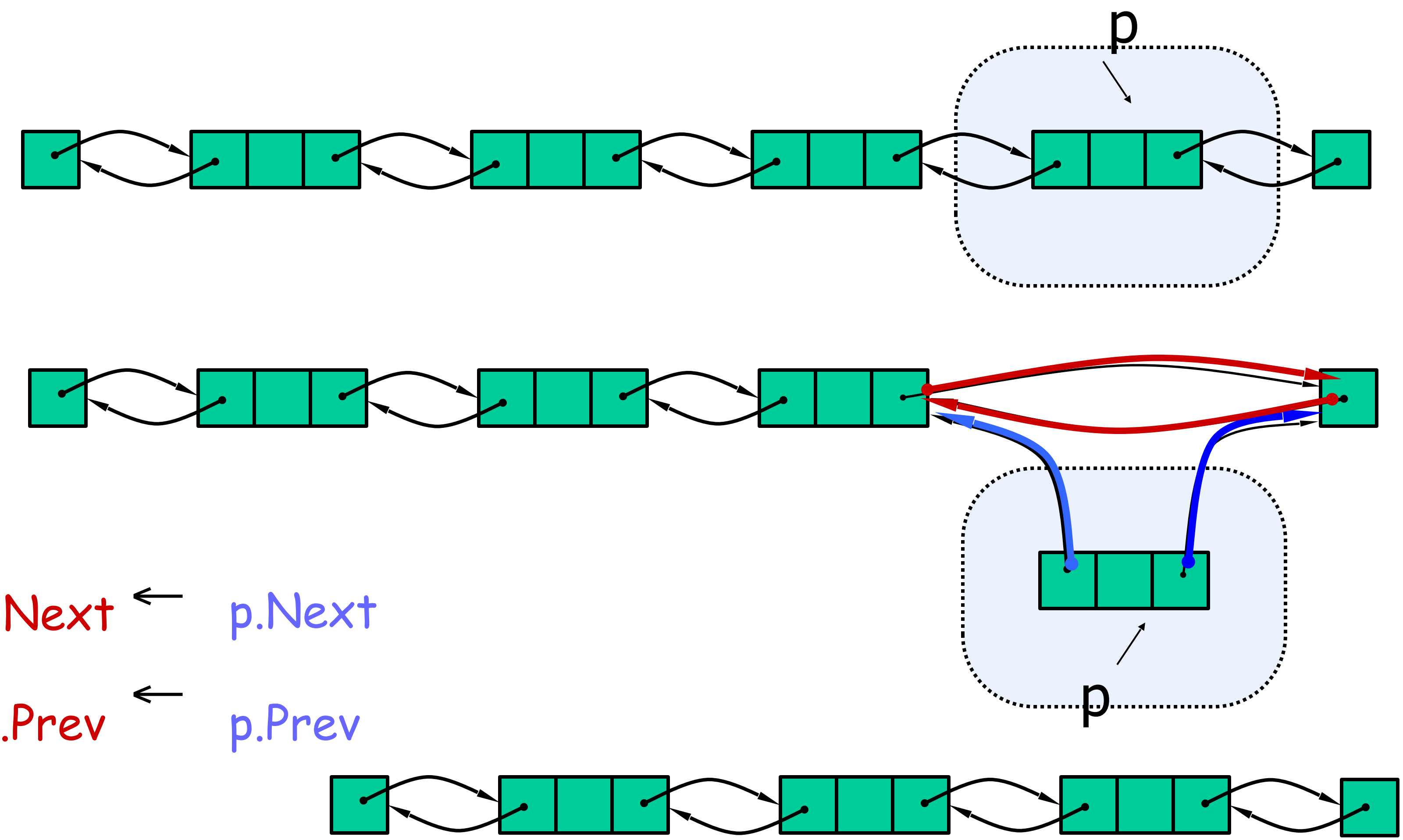
Deletion (first element)



header.next.next.prev ← header

header.next ← header.next.next

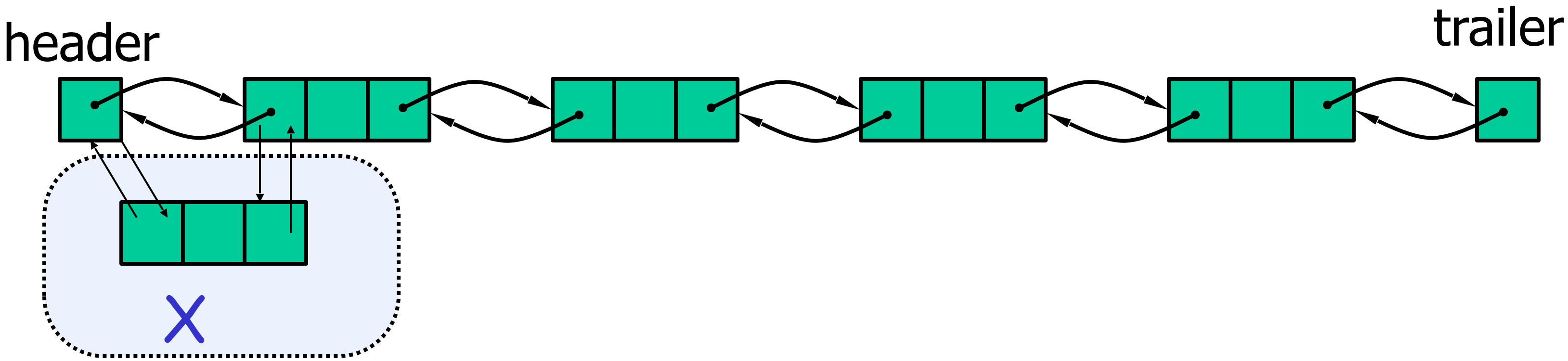
Deletion (element p)



p.Prev.Next ← p.Next

p.Next.Prev ← p.Prev

Insertion (beginning)



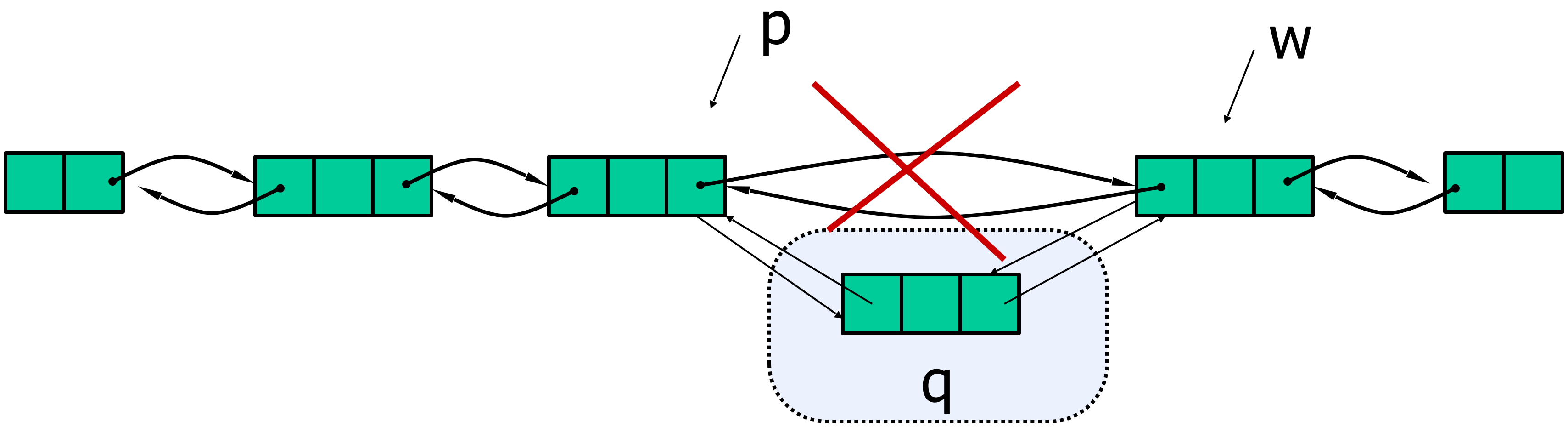
```
X.next ← header.next

header.next ← X

X.prev ← header


X.next.prev ← X
```

Insertion (after p)

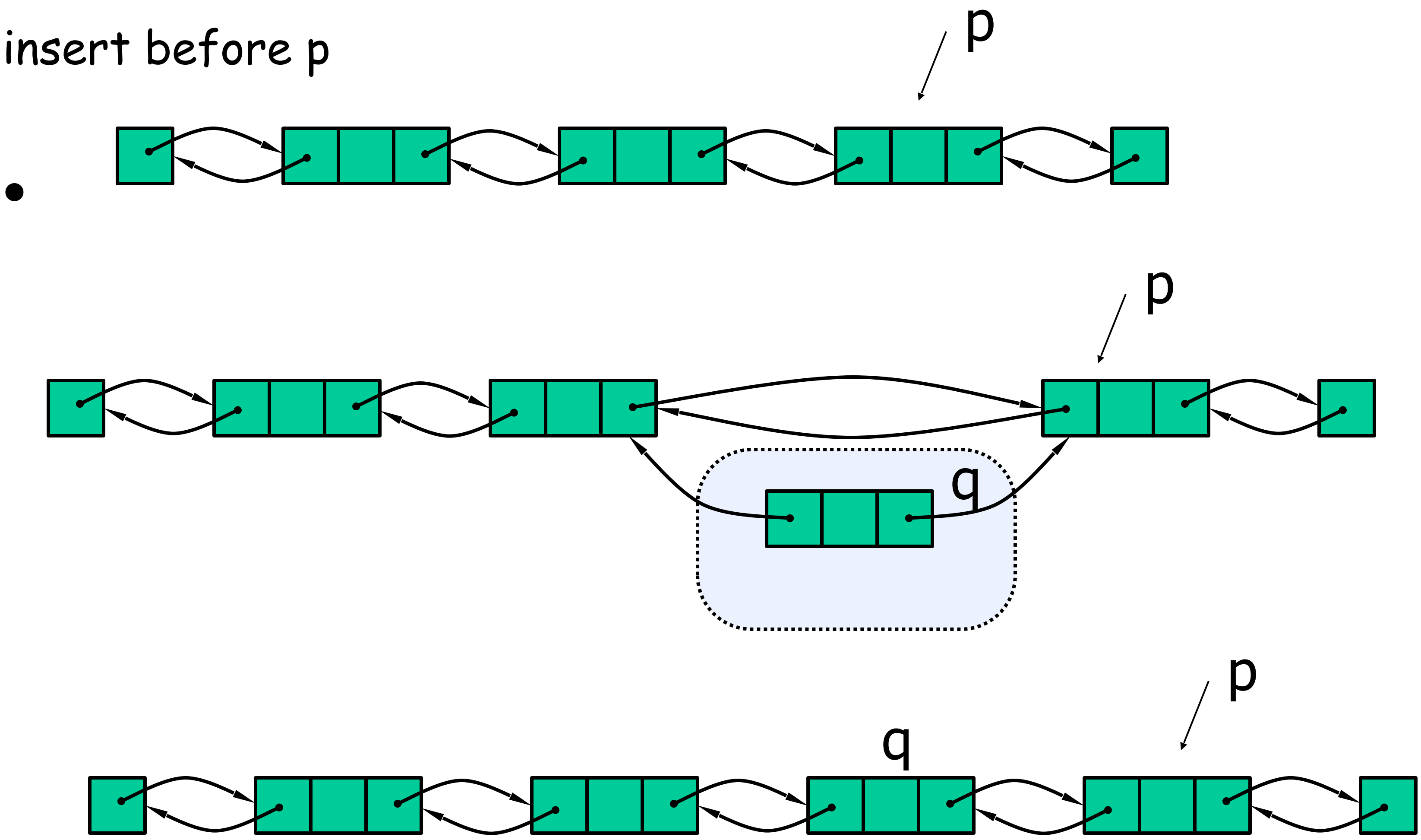


addAfter(p,q)

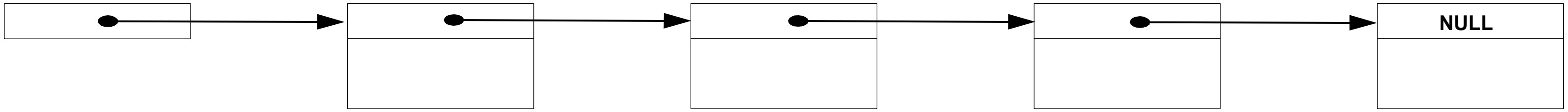
```
    w ← p.getNext()
    q.setPrev(p)
    q.setNext(w)
    w.setPrev(q)
    p.setNext(q)
```

Insertion (before p)

- insert before p



Linked Structures



Dynamic structure: it is never full

No movements of elements

but

There is no DIRECT ACCESS to an element
the list has to be traversed

Java implementation - you will see it in the Labs

A node of a doubly linked list has a next and a prev link.

The doubly linked list supports methods like these:

- setElement(Object e)
- setNext(Object newNext)
- setPrev(Object newPrev)
- getElement()
- getNext()
- getPrev()