CS2010: ALGORITHMS AND DATA STRUCTURES

Lecture 6: Doubly Linked Lists

Vasileios Koutavas

School of Computer Science and Statistics
Trinity College Dublin

Java collections library

public interface **java.util.List**<E>

boolean	**add**(int index, E element)	Inserts the specified element at the specified position in this list (optional operation).
E	**remove(**int index)	Removes the element at the specified position in this list
	…	

Implementations:

- **LinkedList** doubly-linked list implementation
- **ArrayList** resizable-array implementation

Example client

```
public static void main(String[] args)
{
  StackOfStrings buffer = new StackQueue<String>();
  while (!StdIn.isEmpty())
  {
    String s = StdIn.readString();
    if (s.equals("<"))
       StdOut.print(buffer.pop());
    else if (s.equals(">"))
       StdOut.print(buffer.dequeue());
    else
       stack.enqueue(s);
} }
```

pop: return and remove the **most recent** element
dequeue: return and remove the **least recent** element
enqueue: add an element

Example client

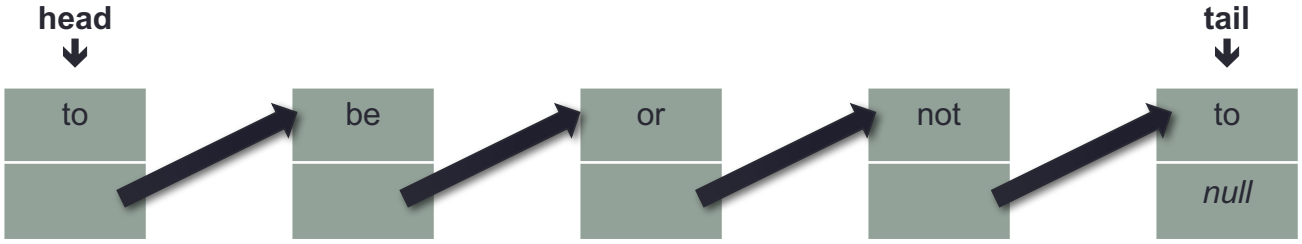
```
public static void main(String[] args)
{
  StackOfStrings buffer = new StackQueue<String>();
  while (!StdIn.isEmpty())
  {
    String s = StdIn.readString();
    if (s.equals("<"))
       StdOut.print(buffer.pop());
    else if (s.equals(">"))
       StdOut.print(buffer.dequeue());
    else
       stack.enqueue(s);
} }
```

Input: "judge me by my size do > you > < < < < < ? >"

Example client

```
public static void main(String[] args)
{
  StackOfStrings buffer = new StackQueue<String>();
  while (!StdIn.isEmpty())
  {
    String s = StdIn.readString();
    if (s.equals("<"))
       StdOut.print(buffer.pop());
    else if (s.equals(">"))
       StdOut.print(buffer.dequeue());
    else
       stack.enqueue(s);
} }
```

Can we implement StackQueue efficiently using a **linked list**?



Example client

```
public static void main(String[] args)
{
  StackOfStrings buffer = new StackQueue<String>();
  while (!StdIn.isEmpty())
  {
    String s = StdIn.readString();
    if (s.equals("<"))
       StdOut.print(buffer.pop());
    else if (s.equals(">"))
       StdOut.print(buffer.dequeue());
    else
       stack.enqueue(s);
} }
```

Can we implement StackQueue efficiently using a **linked list**?

- enqueue and push to the same end of the LL (will run in **Θ(1)**)
- pop from the same end of the LL & dequeue from the other end
  - one of dequeue/pop will run in **Θ(1)** and the other in **Θ(N)**

we can do better

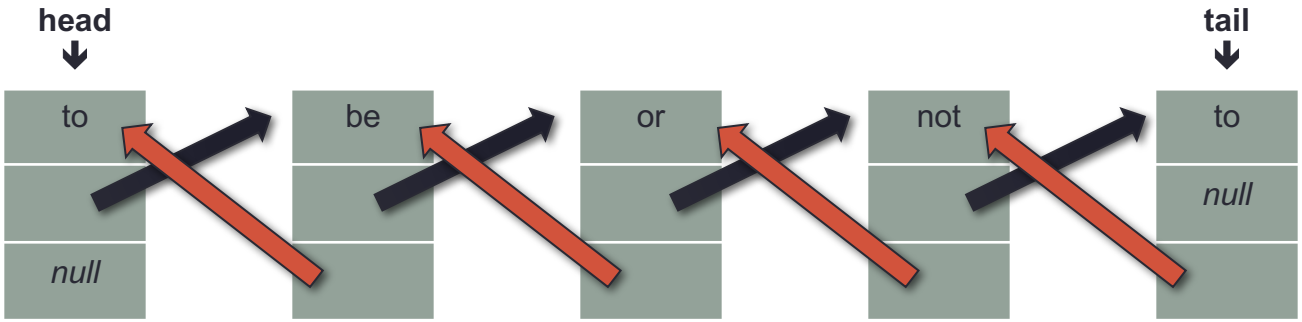
Doubly Linked Lists

(Java implementation in Assignment 2)

https://www.scss.tcd.ie/Vasileios.Koutavas/teaching/cs2010/mt1819/assignment-2/

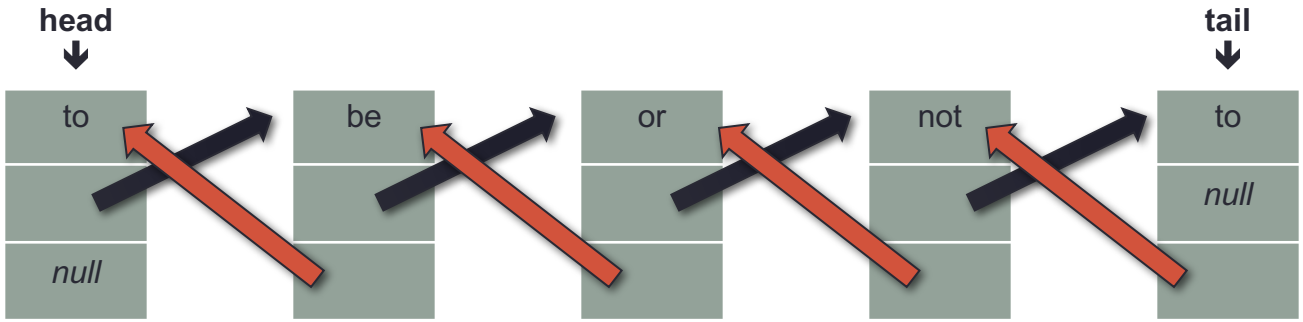
Doubly Linked Lists (DLL)

```
class DLLNode {
  String item;
  DLLNode next;
  DLLNode prev;
}
```

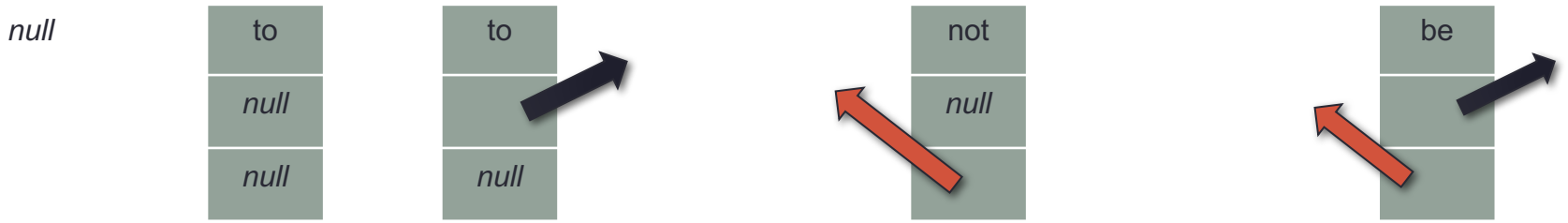


Doubly Linked Lists (DLL)

```
class DLLNode {
  String item;
  DLLNode next;
  DLLNode prev;
}
```



Kinds of nodes inside a DLL:



Empty DLL

The **only node** of a DLL with 1 node

The **first node** of a DLL with >1 node

The **last node** of a DLL with >1 node

A **middle node** of a DLL with >2 items

Doubly Linked Lists

```
class     DLLofString

          DoublyLinkedList()

void      insertFirst(String s)       inserts s at the head of the list
String    getFirst()                  returns string at the head of the list
boolean   deleteFirst()               removes string at the head of the list
void      insertLast(String s)        inserts s at the end of the list
String    getLast(String s)           returns string at the end of the list
boolean   deleteLast()                removes string at the end of the list

void      insertBefore(int pos, String s)      inserts s before position pos
String    get(int pos)                         returns string at position pos
boolean   deleteAt(int pos)                    deletes string at position pos
```

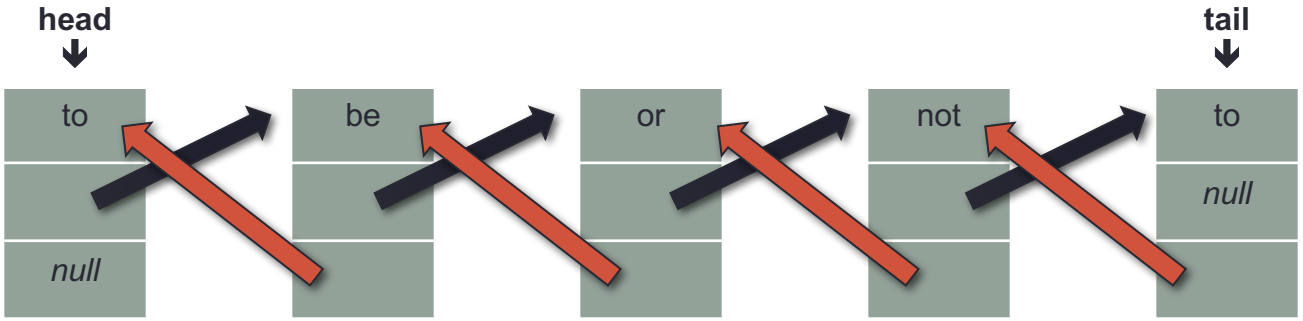
- Interface somewhat different than that of java.util.List.
- How to make interface generic?
  - class DoublyLinkedList<T>

Doubly Linked Lists

class	DLLofString	
	DoublyLinkedList()	
void	insertFirst(String s)	*inserts s at the head of the list*
String	getFirst()	returns string at the head of the list
boolean	deleteFirst()	removes string at the head of the list
void	insertLast(String s)	inserts s at the end of the list
String	getLast(String s)	returns string at the end of the list
boolean	deleteLast()	removes string at the end of the list
void	insertBefore(int pos, String s)	inserts s before position pos
String	get(int pos)	returns string at position pos
boolean	deleteAt(int pos)	deletes string at position pos

- Interface somewhat different than that of java.util.List.
- How to make interface generic?
  - class DoublyLinkedList<T>

Three main cases to deal with



>1 node

1 node

head → null
tail → null

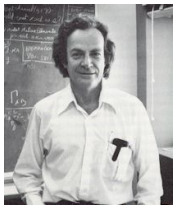
0 nodes

```
/**
     * Inserts an element at the end of the doubly linked list
     * @param data : The new data of class T that needs to be added to the list
     * @return none
     *
     */
    public void insertLast( T data )
```

`void insertLast("be")`

cases for size of the DLL

DLL of **size 0**	?
DLL of **size 1**	?
DLL of **size > 1**	?

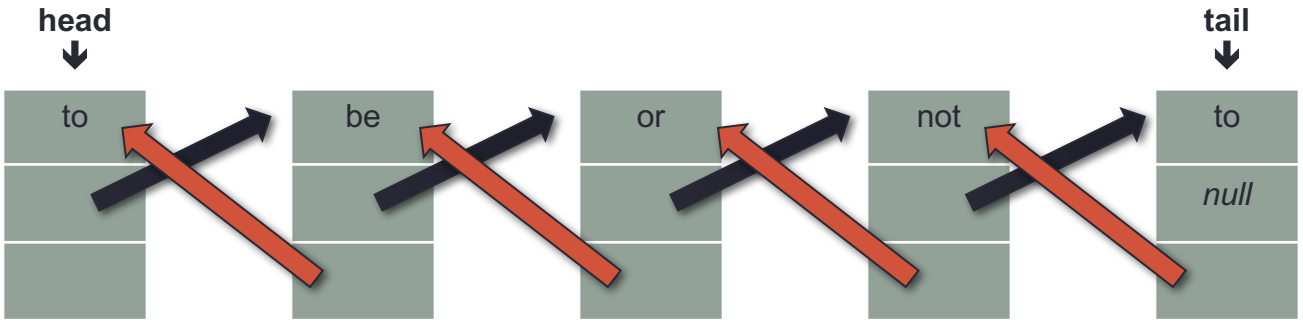


Before you write any code consider simple examples, but not too simple!

(paraphrase of Richard Feynman)

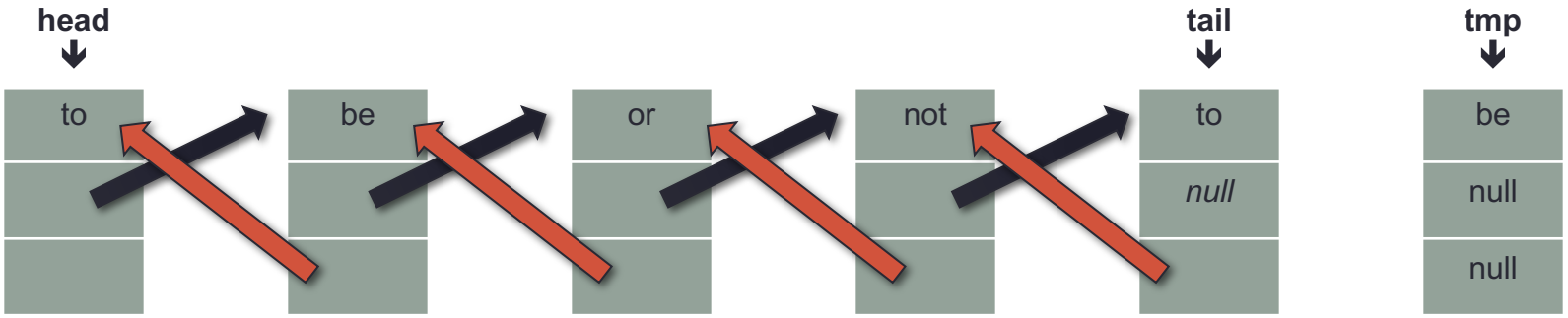
insertLast(“be”)

>1 node



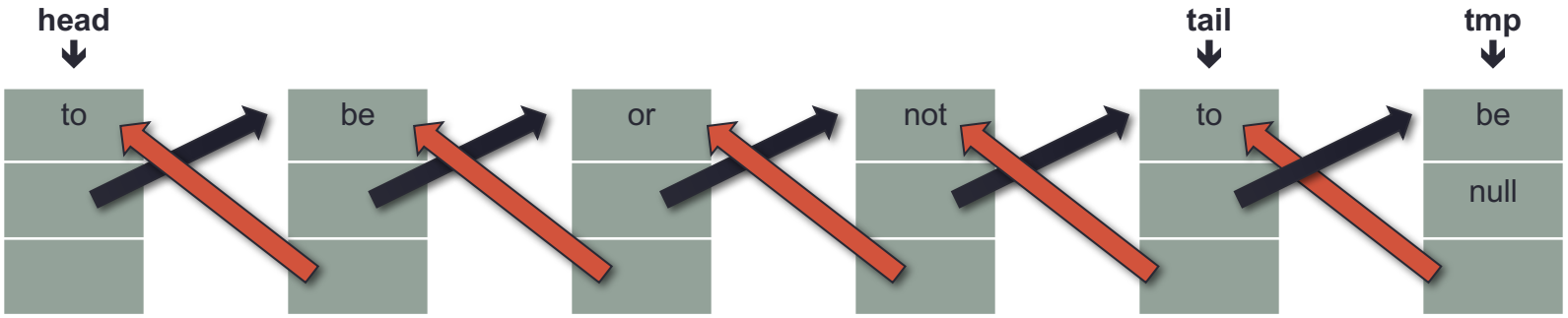
insertLast("be")

>1 node



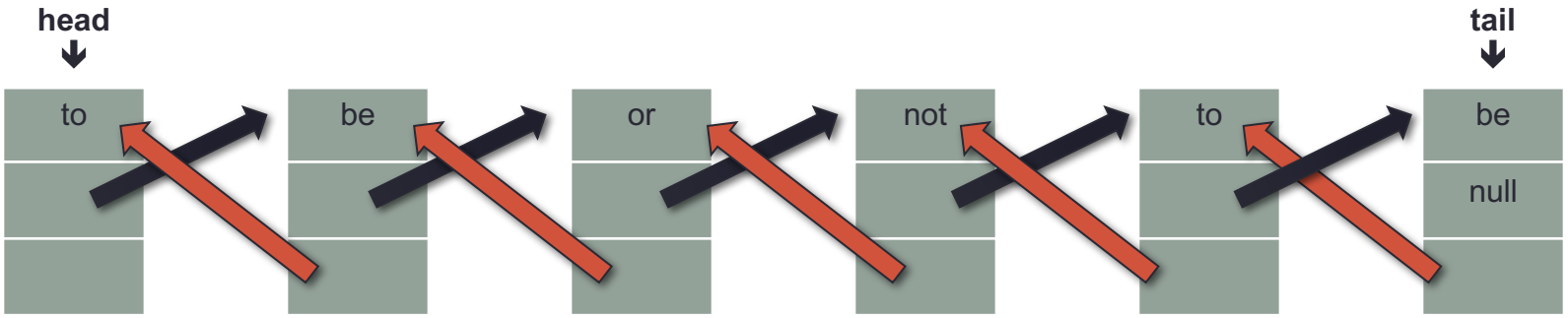
insertLast("be")

>1 node



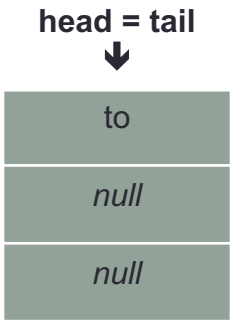
insertLast("be")

>1 node



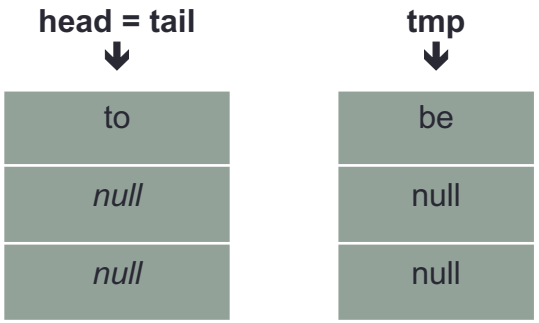
`insertLast("be")`

1 nodes



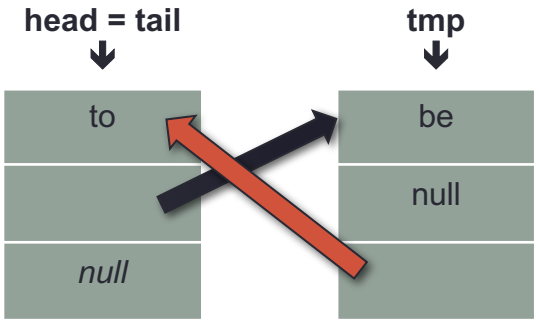
insertLast("be")

1 node



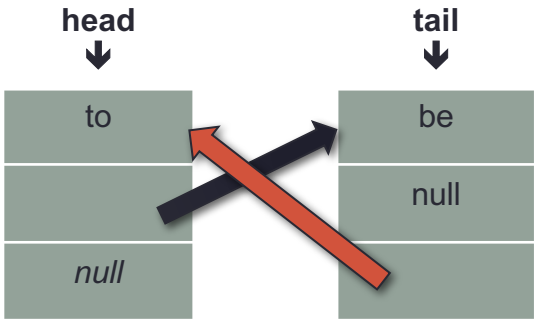
insertLast("be")

1 node



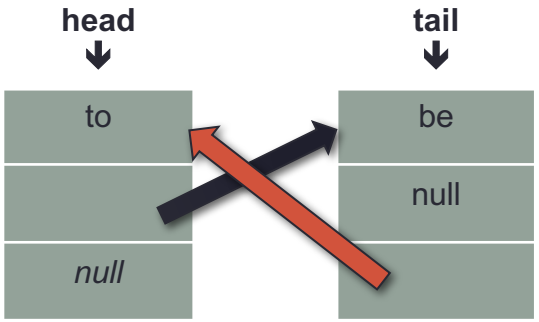
insertLast("be")

1 node



insertLast("be")

1 node



* Same as the case with >1 node

`insertLast("be")`

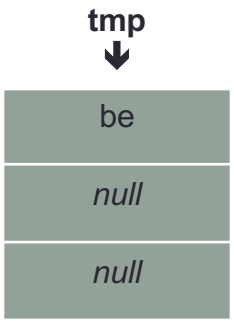
0 nodes

head → null **tail → null**

`insertLast("be")`

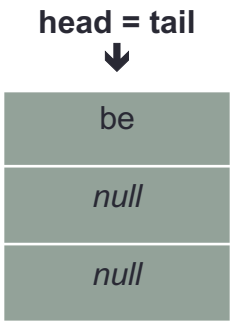
0 nodes

head → null **tail → null**



`insertLast("be")`

0 nodes



```
/**
    * Inserts an element at the beginning of the doubly linked list
    * @param data : The new data of class T that needs to be added to the list
    * @return none
    *
    */
   public void insertFirst( T data )
```

how can we implement this?

void insertFirst("be")

cases for size of the DLL	
DLL of **size 0**	?
DLL of **size 1**	?
DLL of **size > 1**	?

write down an example for each case

```
/**
    * Inserts an element in the doubly linked list
    * @param pos : The integer location at which the new data should be
    *      inserted in the list. We assume that the first position in the list
    *      is 0 (zero). If pos is less than 0 then add to the head of the list.
    *      If pos is greater or equal to the size of the list then add the
    *      element at the end of the list.
    * @param data : The new data of class T that needs to be added to the list
    * @return none
    *
    */
   public void insertBefore( int pos, T data )
```

void insertBefore(pos,“be”)

	pos == 0 (insert at the head of the DLL)	**0 < pos < DLL.size -1** (insert in the middle of the DLL)	**pos == DLL.size -1**	**pos < 0** (insert at the head of the DLL)	**pos ≥ DLL.size** (insert at the end of the DLL)
DLL of **size 0**					
DLL of **size 1**					
DLL of **size > 1**					

void insertBefore(pos,“be”)

	pos == 0 (insert at the head of the DLL)	**0 < pos < DLL.size -1** (insert in the middle of the DLL)	**pos == DLL.size -1**	**pos < 0** (insert at the head of the DLL)	**pos ≥ DLL.size** (insert at the end of the DLL)
DLL of **size 0**	insertFirst(“be”)				
DLL of **size 1**	insertFirst(“be”)				
DLL of **size > 1**	insertFirst(“be”)				

void insertBefore(pos,“be”)

	pos == 0 (insert at the head of the DLL)	**0 < pos < DLL.size -1** (insert in the middle of the DLL)	**pos == DLL.size -1**	**pos < 0** (insert at the head of the DLL)	**pos ≥ DLL.size** (insert at the end of the DLL)
DLL of **size 0**	insertFirst(“be”)	*not possible* *0 < pos < -1*			
DLL of **size 1**	insertFirst(“be”)	*not possible* *0 < pos < 0*			
DLL of **size > 1**	insertFirst(“be”)	?			

void insertBefore(pos,“be”)

	pos == 0 (insert at the head of the DLL)	**0 < pos < DLL.size -1** (insert in the middle of the DLL)	**pos == DLL.size -1**	**pos < 0** (insert at the head of the DLL)	**pos ≥ DLL.size** (insert at the end of the DLL)
DLL of **size 0**	insertFirst(“be”)	*not possible* *0 < pos < -1*	insertFirst(“be”) (here pos == -1)		
DLL of **size 1**	insertFirst(“be”)	*not possible* *0 < pos < 0*	insertFirst(“be”) (here pos == 0)		
DLL of **size > 1**	insertFirst(“be”)	?	?		

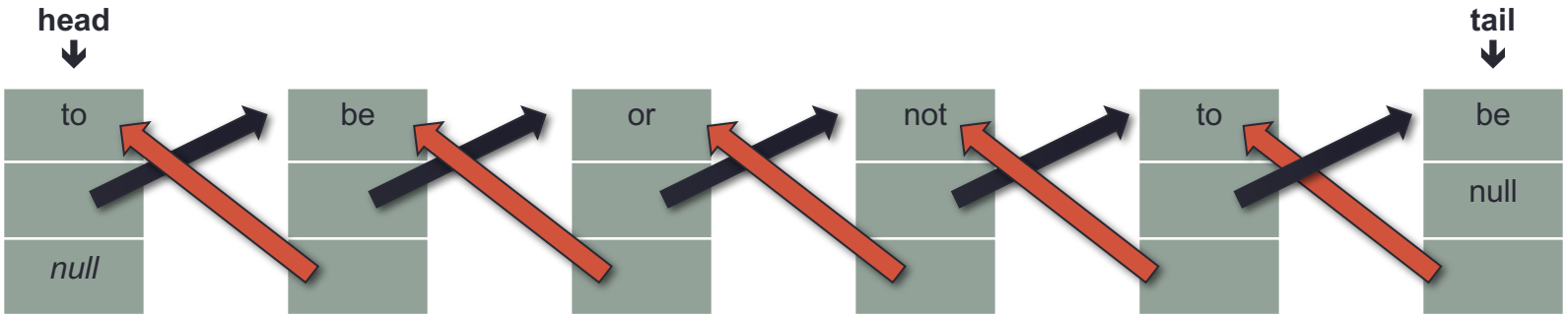
void insertBefore(pos,“be”)

	pos == 0 (insert at the head of the DLL)	**0 < pos < DLL.size -1** (insert in the middle of the DLL)	**pos == DLL.size -1**	**pos < 0** (insert at the head of the DLL)	**pos ≥ DLL.size** (insert at the end of the DLL)
DLL of **size 0**	insertFirst(“be”)	*not possible* *0 < pos < -1*	insertFirst(“be”) (here pos == -1)	insertFirst(“be”)	InsertLast(“be”)
DLL of **size 1**	insertFirst(“be”)	*not possible* *0 < pos < 0*	insertFirst(“be”) (here pos == 0)	insertFirst(“be”)	InsertLast(“be”)
DLL of **size > 1**	insertFirst(“be”)	?	?	insertFirst(“be”)	InsertLast(“be”)

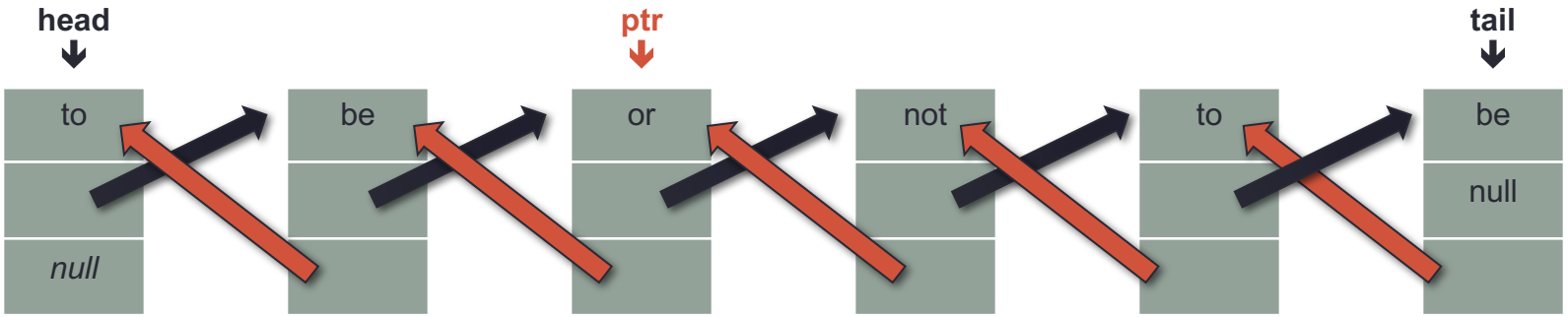
void insertBefore(pos,“be”)

	pos == 0 (insert at the head of the DLL)	**0 < pos < DLL.size -1** (insert in the middle of the DLL)	**pos == DLL.size -1**	**pos < 0** (insert at the head of the DLL)	**pos ≥ DLL.size** (insert at the end of the DLL)
DLL of **size 0**	insertFirst(“be”)	*not possible* *0 < pos < -1*	insertFirst(“be”) (here pos == -1)	insertFirst(“be”)	InsertLast(“be”)
DLL of **size 1**	insertFirst(“be”)	*not possible* *0 < pos < 0*	insertFirst(“be”) (here pos == 0)	insertFirst(“be”)	InsertLast(“be”)
DLL of **size > 1**	insertFirst(“be”)	?	same as case to the left	insertFirst(“be”)	InsertLast(“be”)

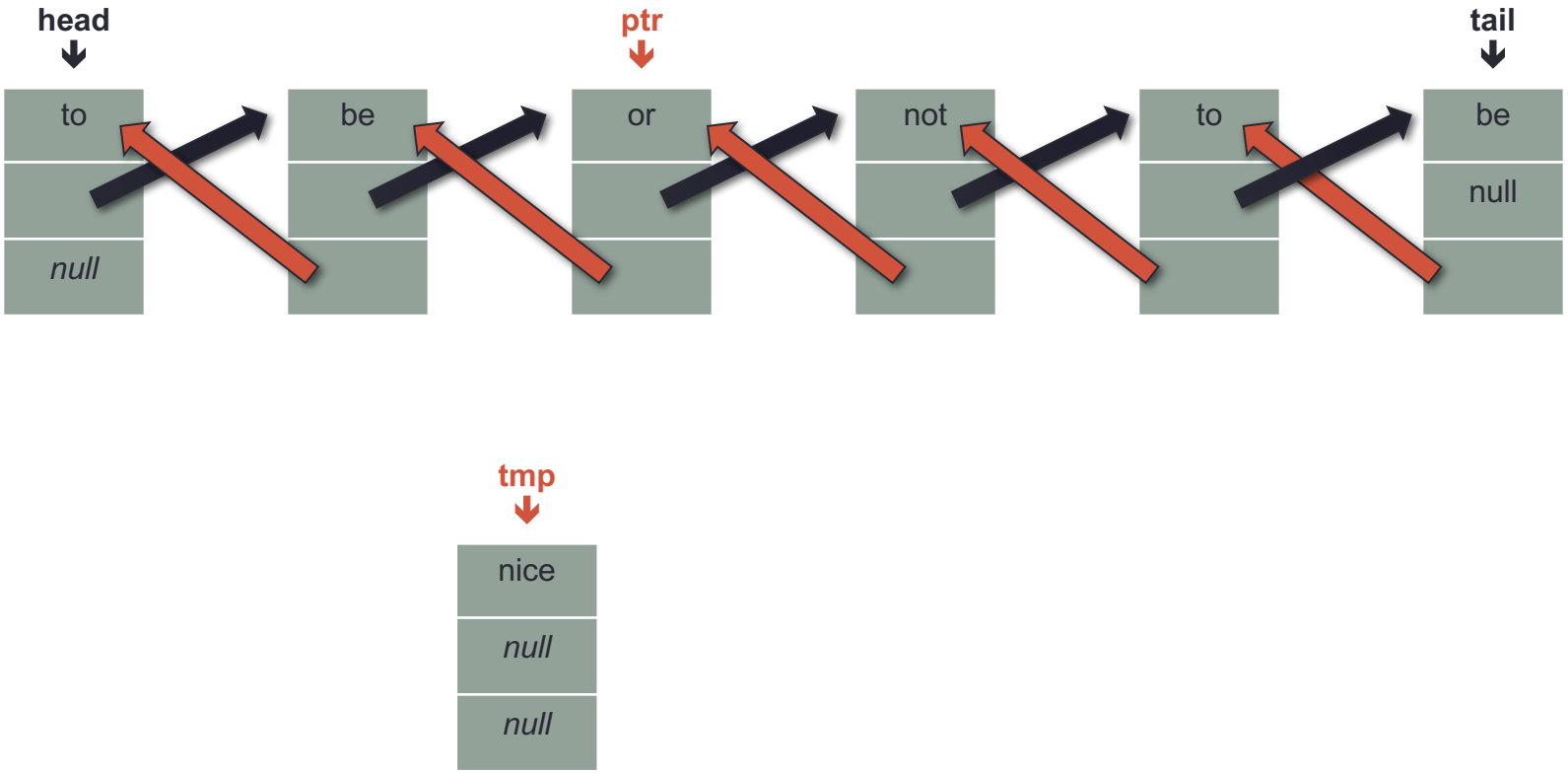
insertBefore(2,“nice”)



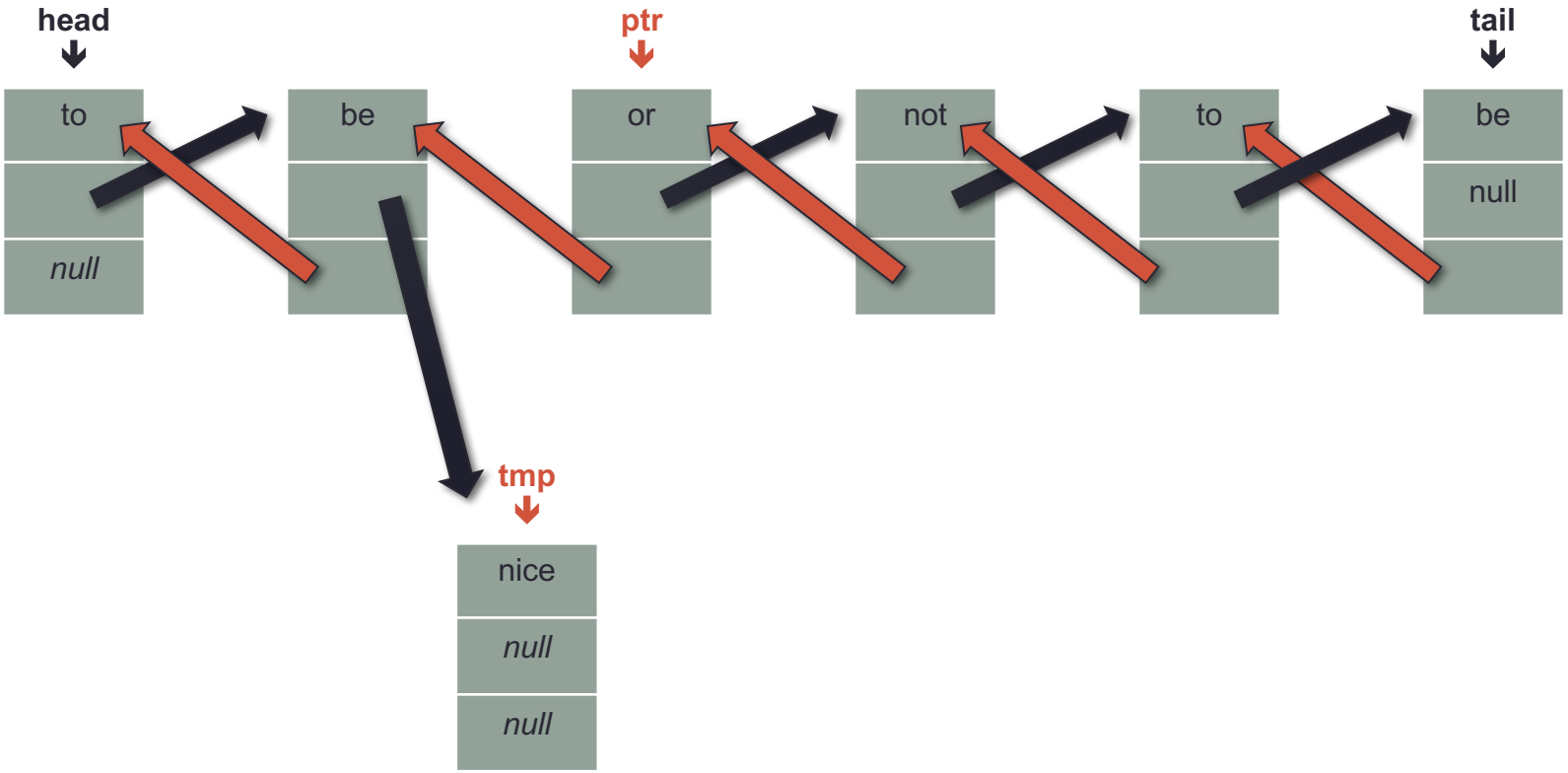
insertBefore(2,“nice”)



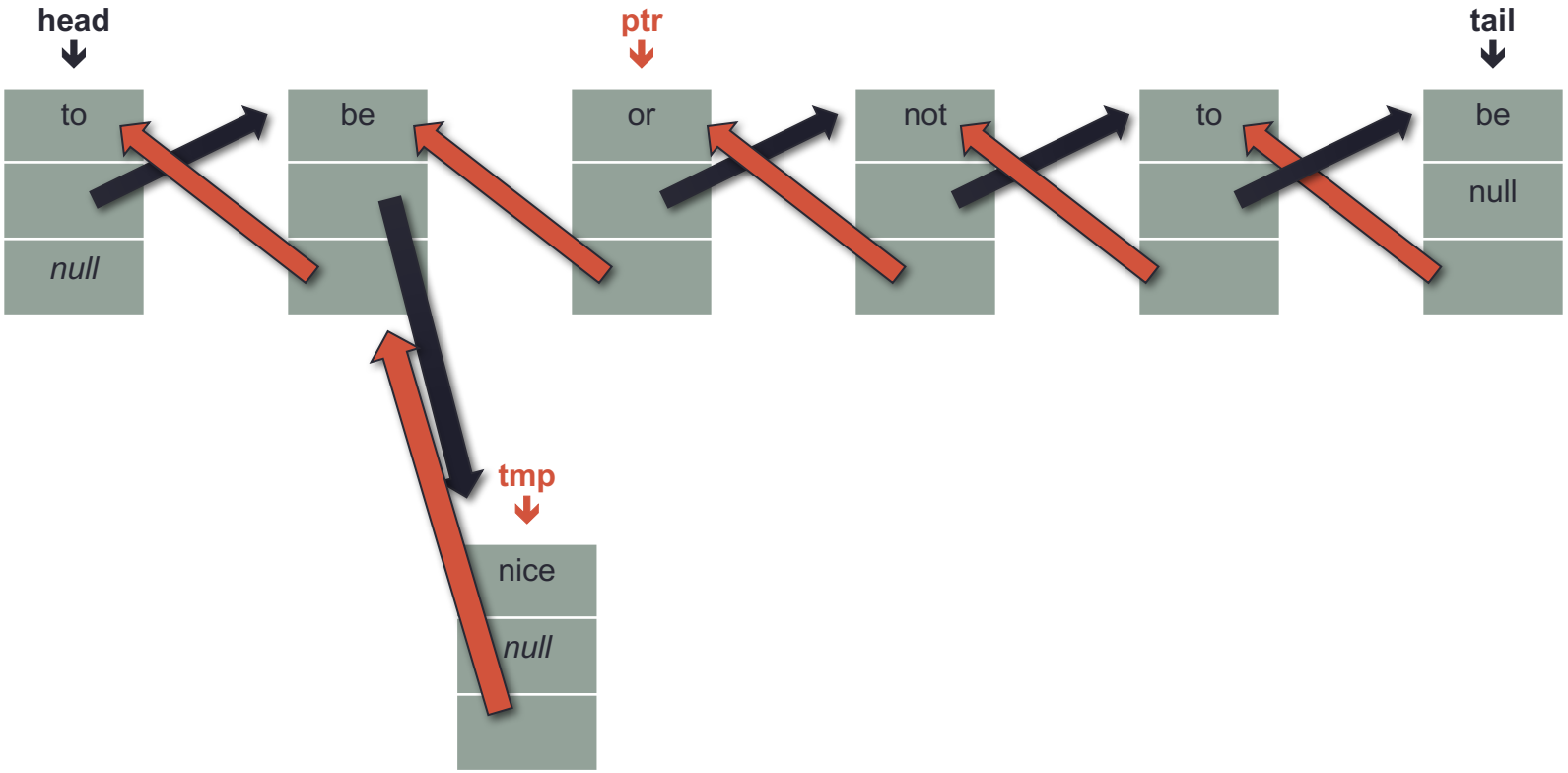
insertBefore(2,“nice”)



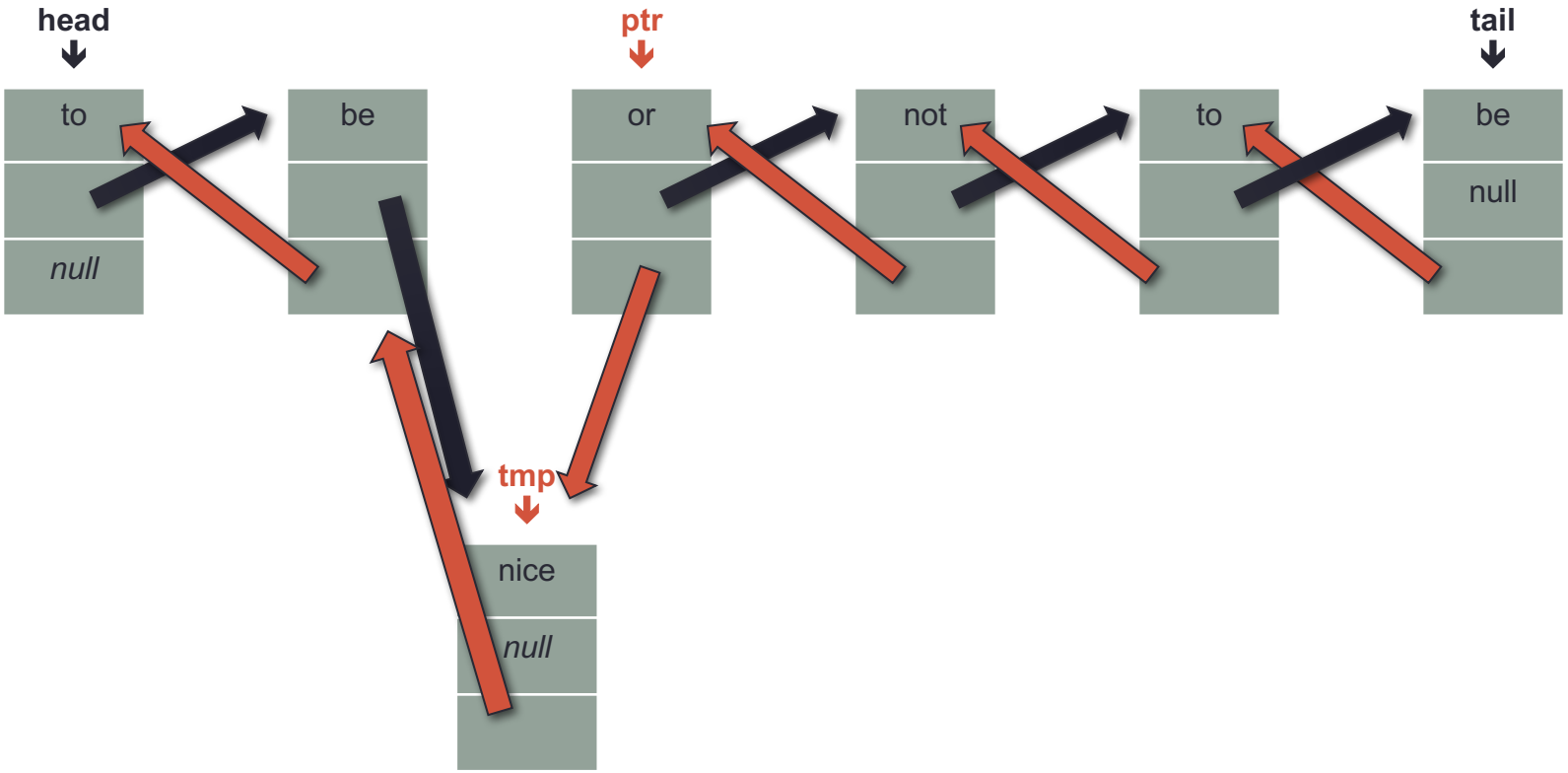
insertBefore(2,“nice”)



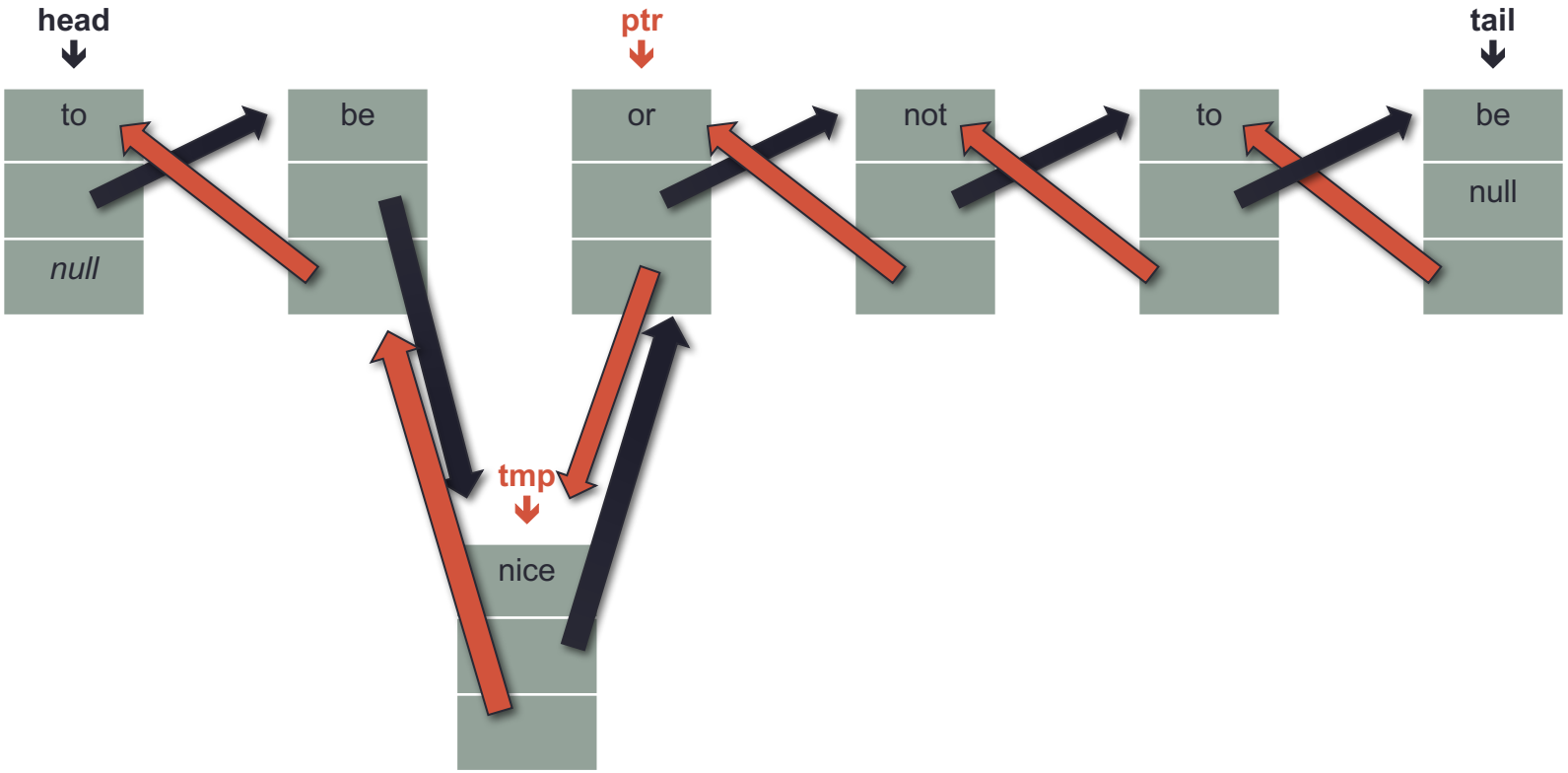
insertBefore(2,“nice”)



insertBefore(2,“nice”)



insertBefore(2,“nice”)



Moral of the story

Before you implement an operation of a complex Data Structure (DS):

- For each different meaningful **state** your DS may be in:
  - For each different meaningful **input** to your DS operation:
    - Create one example **on paper**
    - Work out the example **on paper**
    - **Reuse** code from previous, simpler examples
    - **Group cases** whose examples require the same steps
- **ONLY then write code**