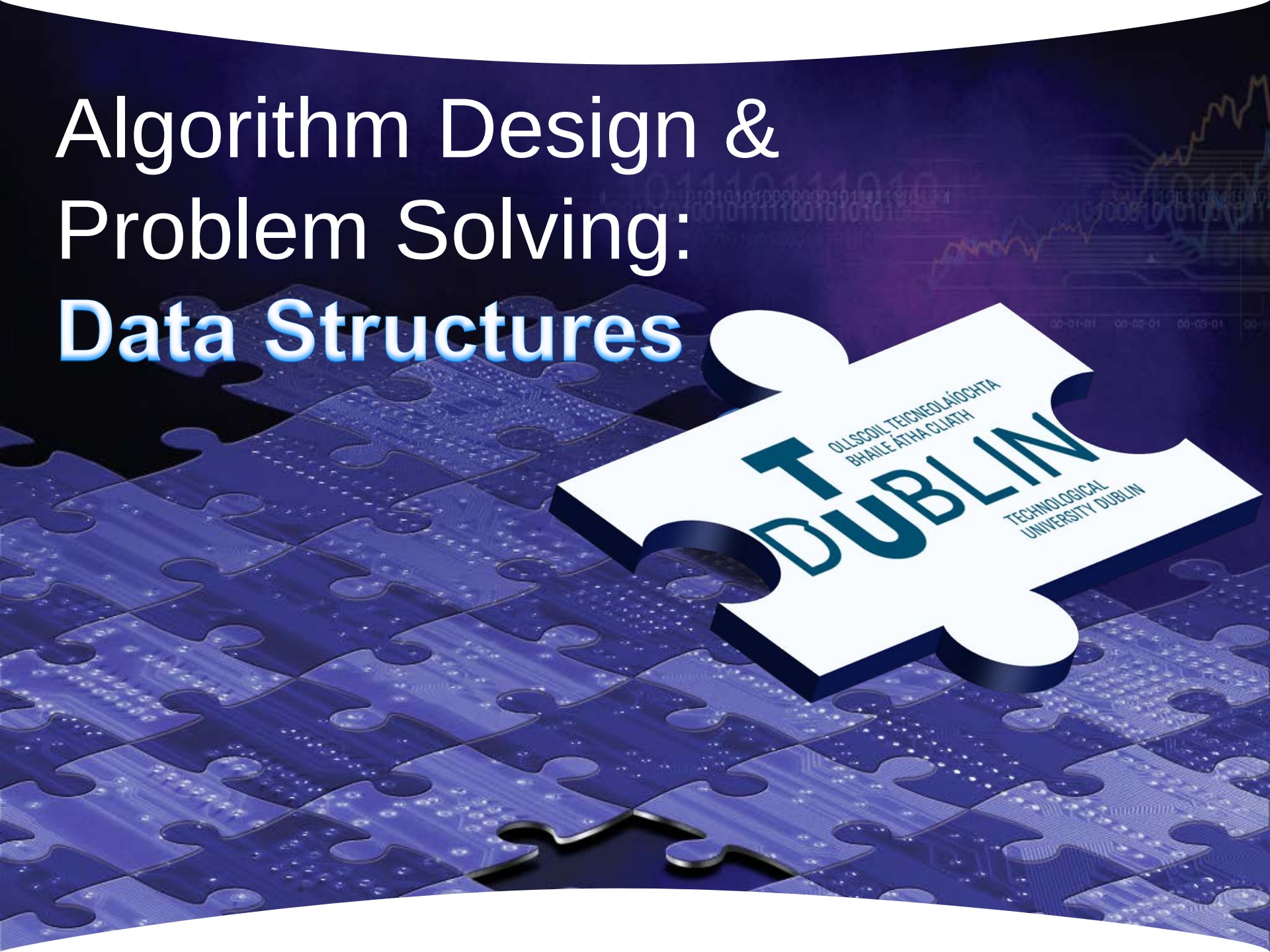
Algorithm Design & Problem Solving:
Data Structures



Contents

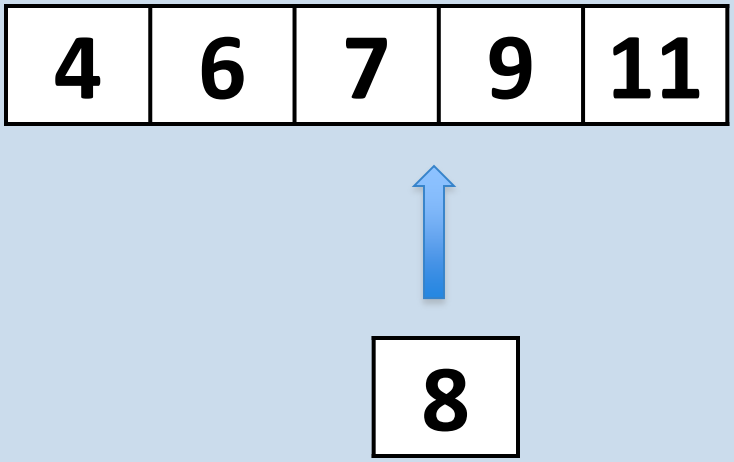
Abstract Data Types

"A data type can be considered **abstract** when it is defined in terms of operations on it, and its implementation is hidden"

Abstract Data Type 1

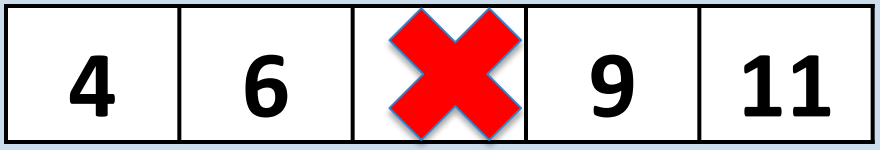
1. What is a **List**
   - How do I insert into a list?
   - How do I delete an item in a list?
   - How do I search a list?

Lists: Inserting



- Check that the list is not full
- Check that the requested position exists
- Shift items up to make room for the new item
- Make the insertion
- Update list size value

Lists: Deleting



- Check that the requested position exists
- Shift items back to delete item
- Update list size value

Lists: Searching

4	6	7	9	11

- In what ways can I search?

How can lists be implemented?

1. **List**
   - Arrays/Records
   - Linked Lists

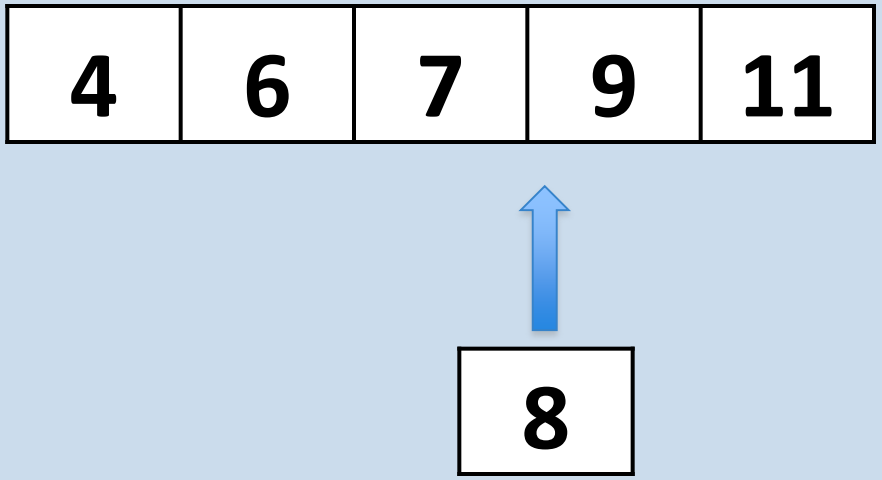
Data Structures: Records

What is a record?

Surname	Byrne	Murphy	Carroll	
First Name	Mary	Tom	Jack	
Sex	Female	Male	Male	
Age	50	45	35	

List: Arrays (same process as records)

1. Inserting



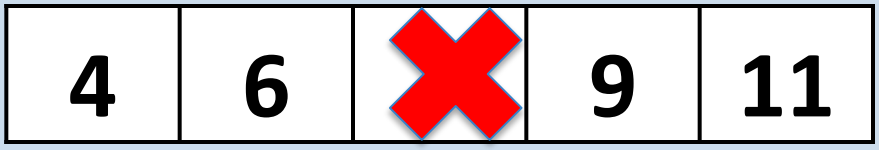
Check that the list is not full
Check that the requested position exists
Shift items up to make room for the new item
Make the insertion
Update list size value

Lists: Arrays-inserting

```
If sizeoflist = sizeofarray
        exit
Else if reqindexnumber is not in the array
        exit
Else
        for i=sizeoflist to i>=reqindexnumber do
                A[i+1]=A[i]
                i--
        A[reqindexnumber]=value
        increment sizeoflist
```

Data Structures: Arrays

2. Deleting



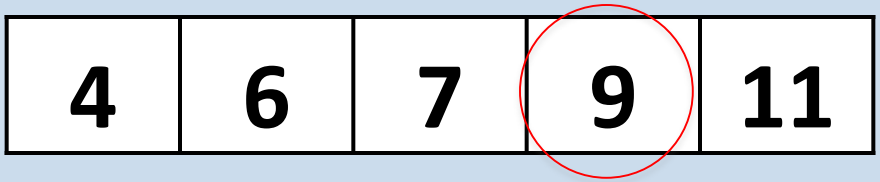
Check that the requested position exists
Shift items back to delete item
Update list size value

Lists: Arrays-deleting

```
if reqindexnumber is not in the array
        exit
Else
        for i=reqindexnumber to i<=sizeoflist do
                A[i]=A[i+1]
        decrement sizeoflist
```

Data Structures: Arrays

3. Searching



Write the code for both types of searches

Reminder: Linear Search Algorithm

```
for i = 0, i < n do
    if A[i] = searchKey
        return i
return -1
```

Reminder: Iterative Binary Search Algorithm

```
While low <= high
    Middle = (low + high)/2
    If searchKey = A[middle]
        return middle
    Else if searchKey < A[middle]
        high = middle - 1
    Else if searchKey > A[middle]
        low = middle + 1
```

Reminder: Recursive Binary Search Algorithm

```
binary_search(A, searchKey, low, high)
    Middle = (low + high)/2
    If searchKey = A[middle]
        return middle
    Else if searchKey < A[middle]
        binary_search(A, searchKey, low, middle – 1)
    Else if searchKey > A[middle]
        binary_search(A,searchKey,middle+1, high)
```

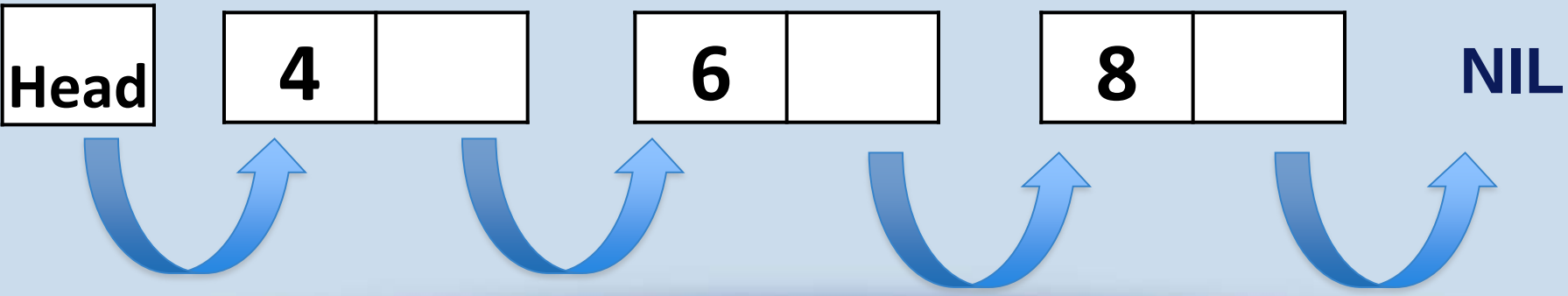
Data Structures: Linked Lists

A **linked list** consists of any number of linked nodes such that each node has at most one predecessor and at most one successor => linear structure

Data Structures: Linked Lists

Each linked list node is actually a record variable with at least two fields:

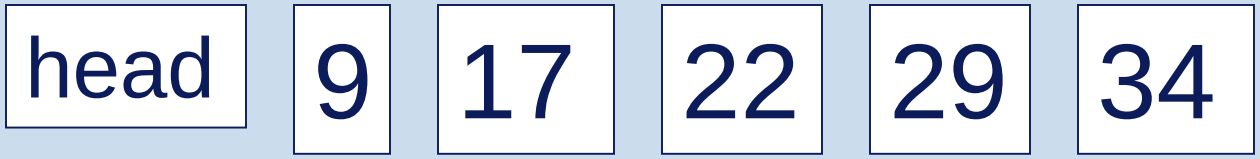
- *Data* field
- *Next* location field, a pointer to the next node



Data Structures: Linked Lists

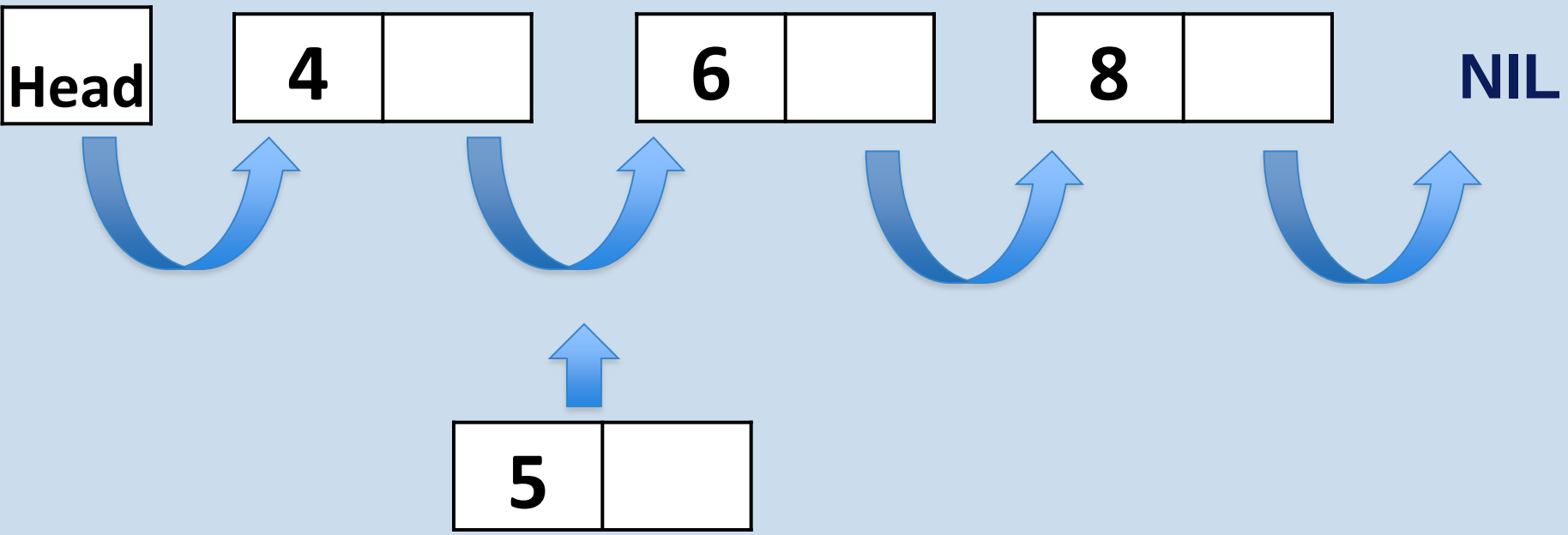
1. Inserting
2. Deleting
3. Searching

Linked List: A Demo



Inserting: Linked Lists

1. Inserting

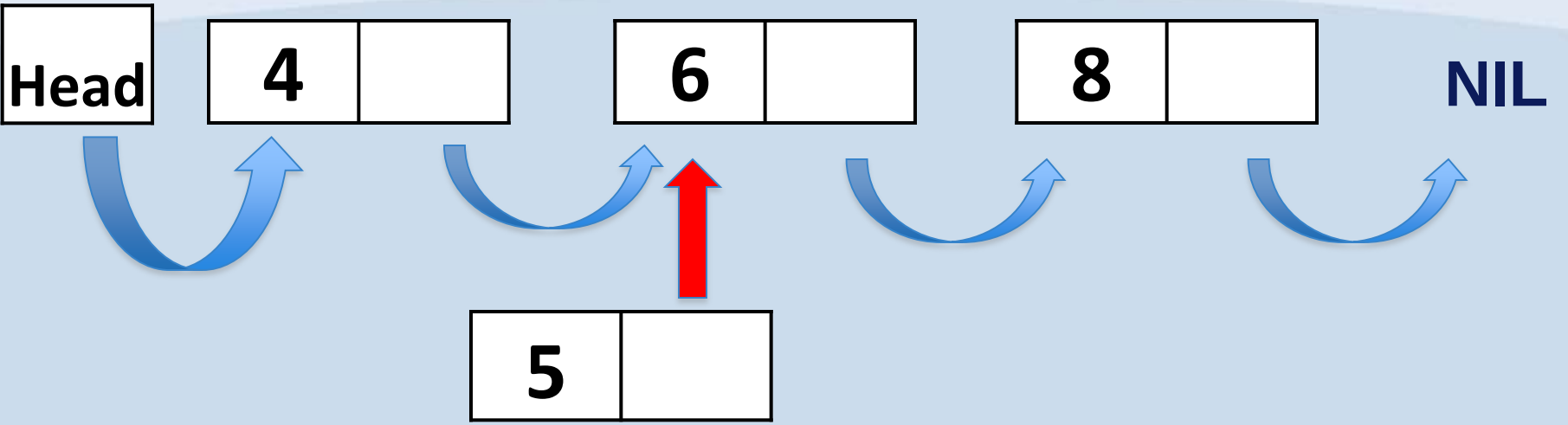


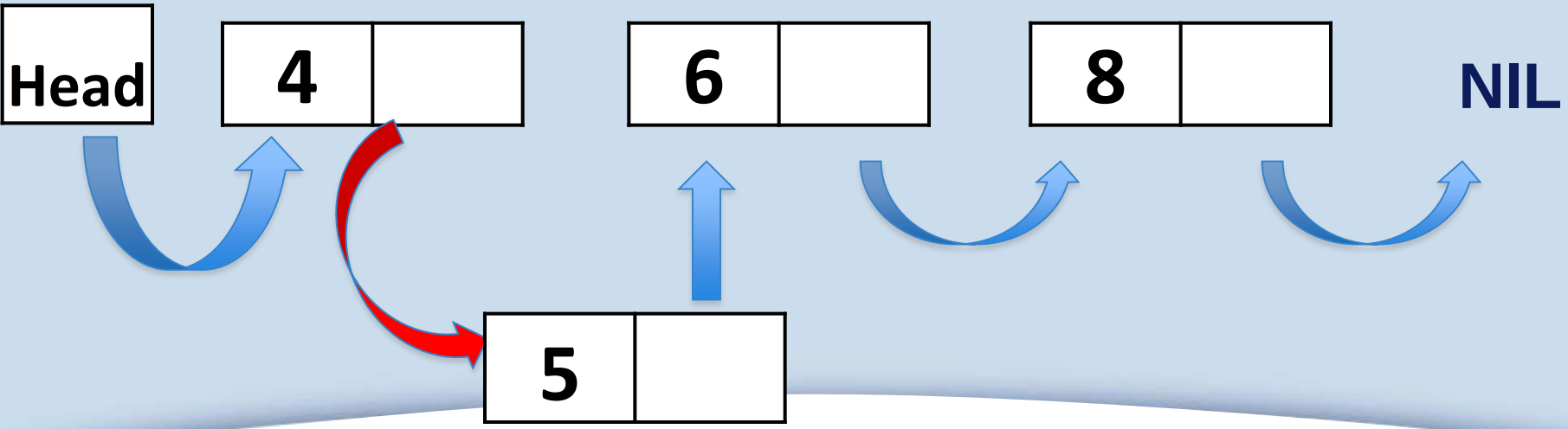
Inserting: Linked List

Involves two steps:

- **Finding the correct location**
- **Doing the work to add the node**

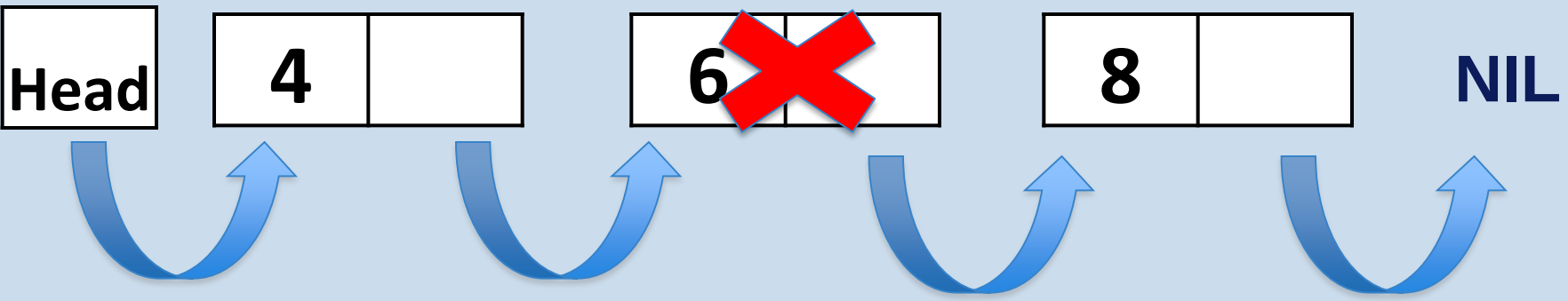
Inserting: Linked Lists





Deleting: Linked Lists

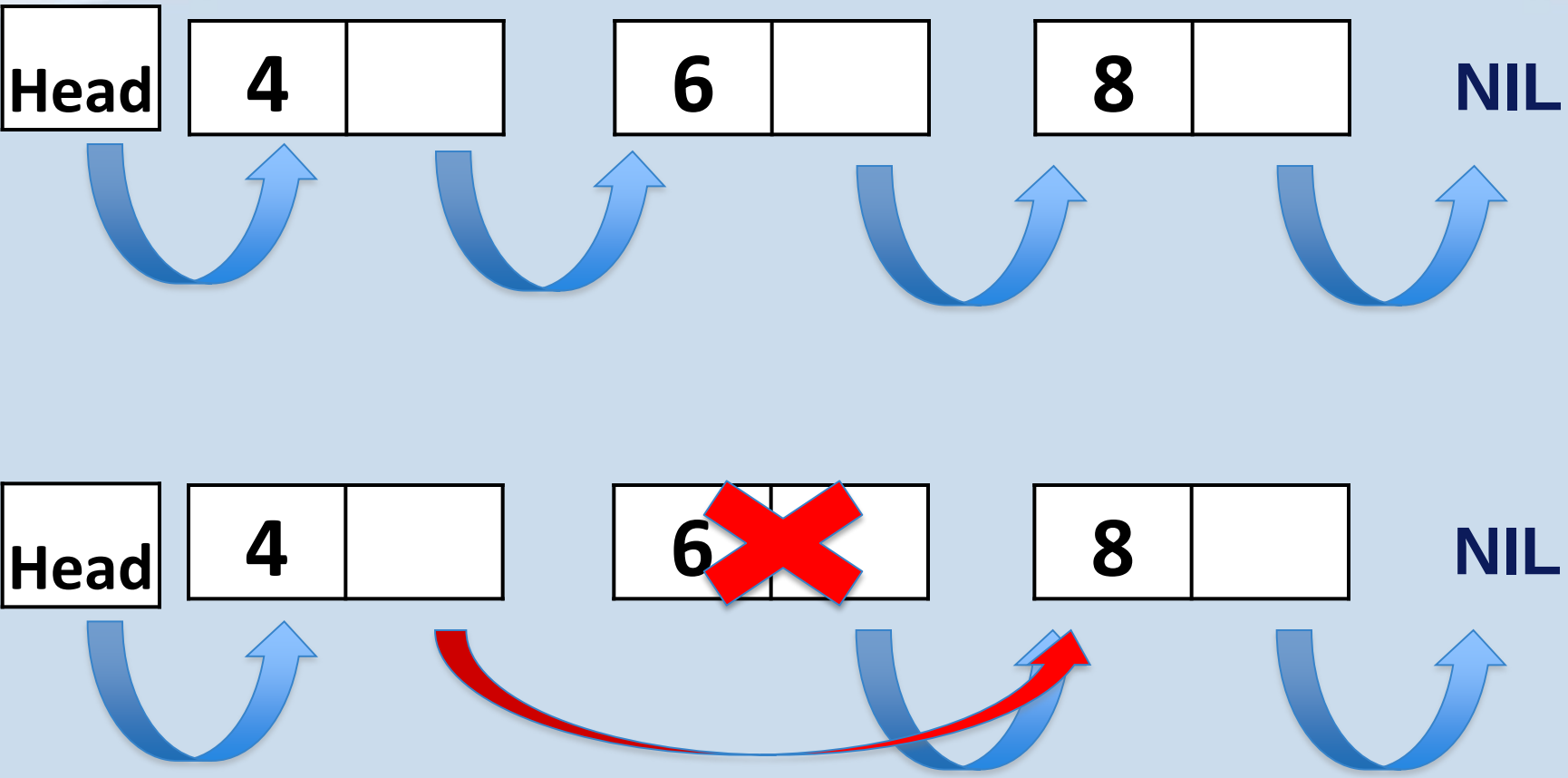
2. Deleting



Deleting: Linked List

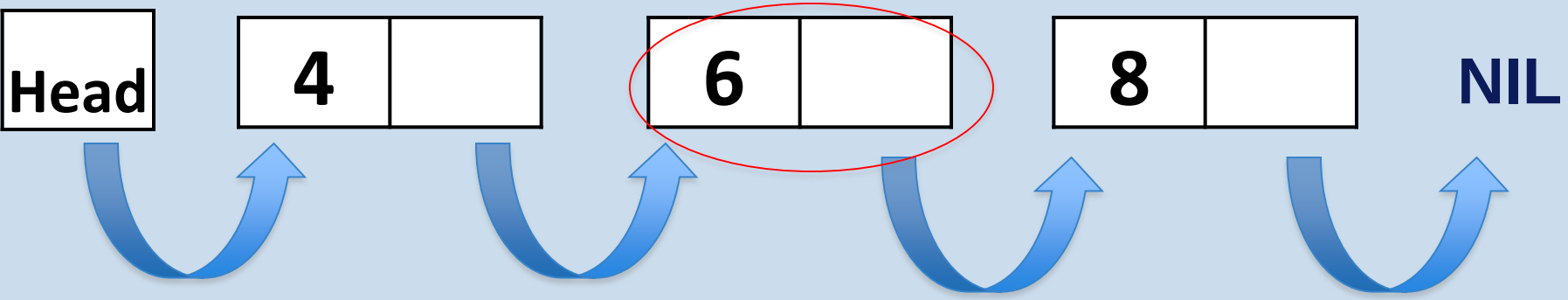
- **Deletion from a linked list involves two steps:**
  - Find a match to the element to be deleted (traverse until nil *or* found)
  - Perform the action to delete
- **Performing the deletion is trivial:**
  - This removes the element, since nothing will point to the node

Deleting: Linked Lists



Searching: Linked Lists

3. Searching



Other Abstract Data Types

1. **Stacks –** works like a physical stack

2. **Queues –** works like a physical queue

What is the difference?

Stacks

LIFO: Last In First Out

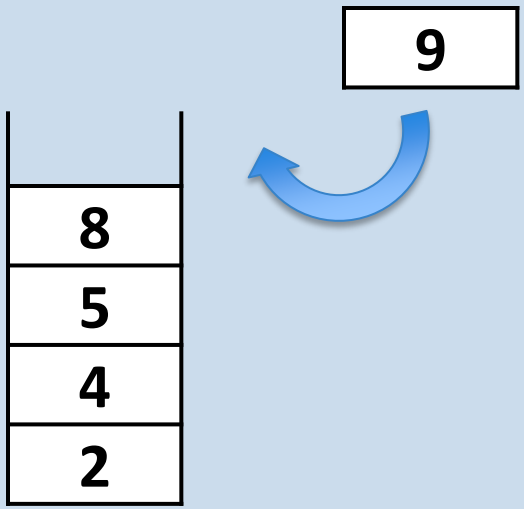
Abstract Data Type 2: Stacks

1. **Push:** put new element on to the stack
2. **Pop:** take an element off the stack

How can this be implemented using an array?

Abstract Data Type 2: Stacks

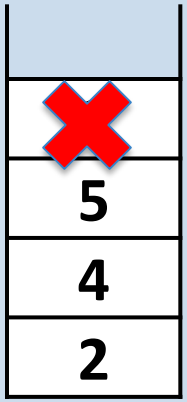
1. Push



If stack is full
exit
else
A[freelocation]= new element
increment freelocation

Abstract Data Type 2: Stacks

2. Pop



If stack is empty
exit
else
decrement freelocation

Queues

FIFO: First In First Out

Abstract Data Type 3: Queues

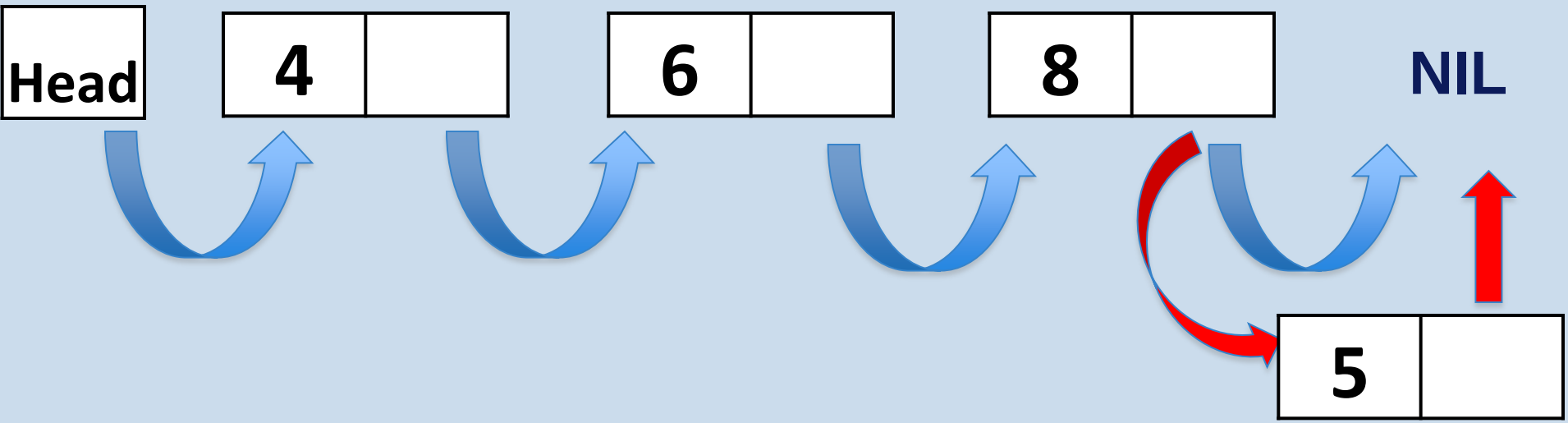
1. **Push:** put new element in to the queue

1. **Pop:** remove an element from the queue

Should this be implemented using an array?
No!

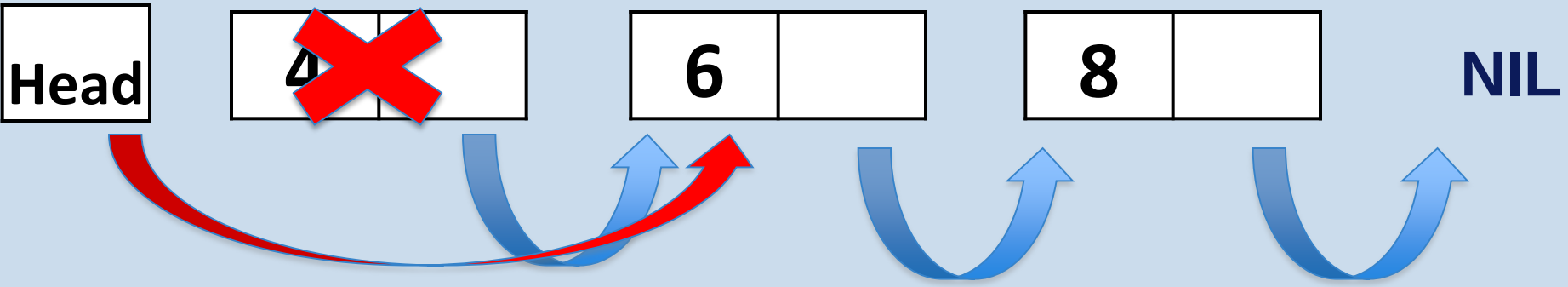
Abstract Data Type 3: Queues

1. Push



Abstract Data Type 3: Queues

2. Pop



Thank You !


OLLSCOIL TEICNEOLAÍOCHTA
BHAILE ÁTHA CLIATH
TU
DUBLIN
TECHNOLOGICAL
UNIVERSITY DUBLIN