COMP26120 Algorithms and Data Structures
Topic 2: Data Structures

Data Structures vs Data Types

Dr. Thomas Carroll
thomas.carroll@manchester.ac.uk
All information on Blackboard

Learning Outcomes

- Recall what an Abstract Data Type (ADT) is
- Understand what a Data Structure is
- Explain the difference between ADT and Data Structure
- Understand how common ADTs can be implemented with Linked Lists and Arrays

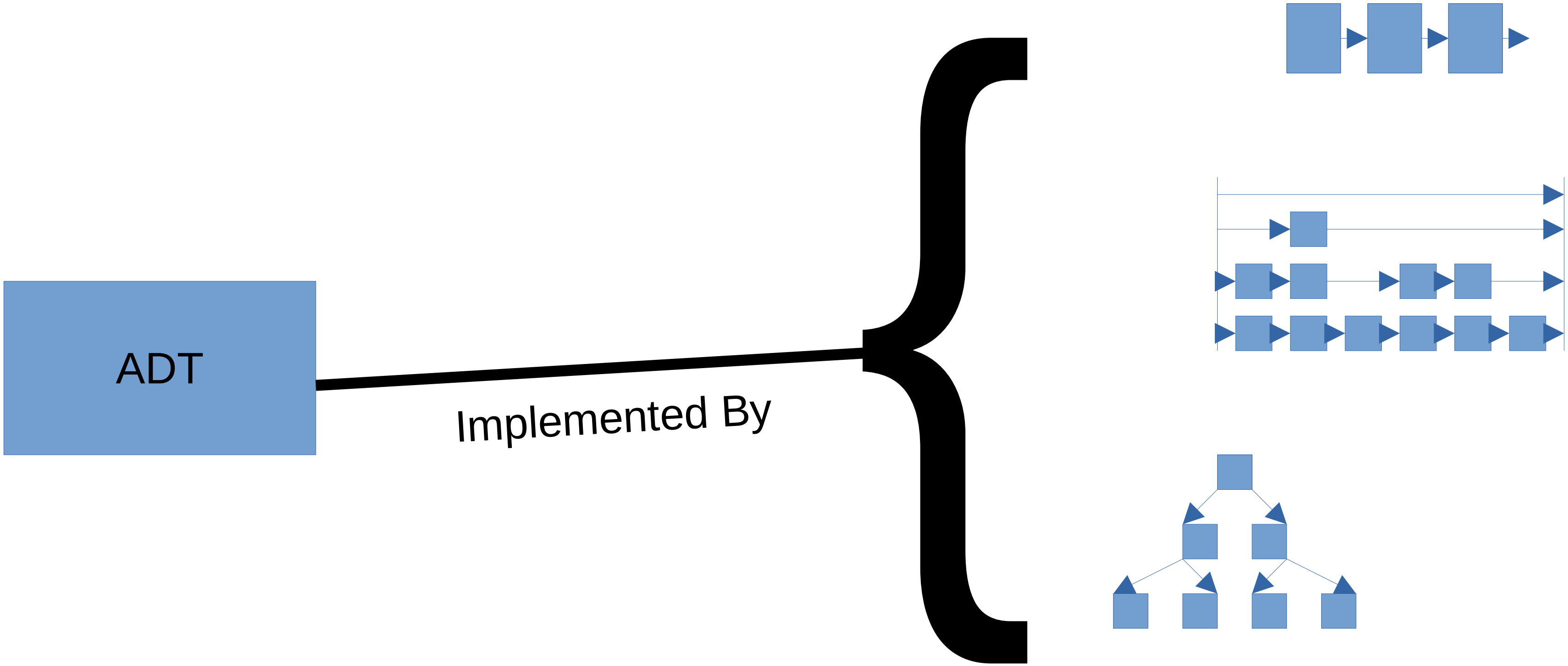
Abstract Data Type

- A theoretical concept which, for some given *data type* defines:
  - The **possible values** it can take
  - The **operations** that can be performed
  - The **behaviour** of these operations
- A **list** might have the following operations:

add	Adds an element to the end of the list
head	Returns the first element of the list
tail	Returns the sublist *after* the head

Data Structure

- A **data structure** is a way to **store and organise** data to facilitate access and modifications
- Data structures **implement** ADTs
- Eg:
  - Linked Lists and Dynamic Arrays can implement the List ADT



ADT

Implemented By

Now...

- See how…
  - Queue
  - Stack
  - Set
  - Dictionary

- Can be implemented by…
  - Dynamic Array
  - Linked List

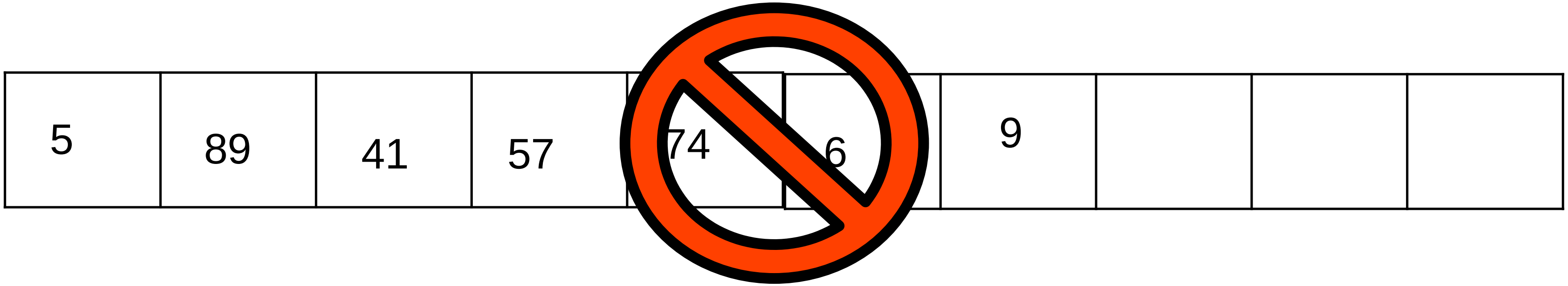
But First...

- What is a Linked List?
- What is a Dynamic Array?

Dynamic Array

Let’s insert the following *k* numbers to the back of the array:
5,89,41,57,74,6,9

```
for i = 0 to k-1
   A[i] = next number
```



5	89	41	57	74	6	9			

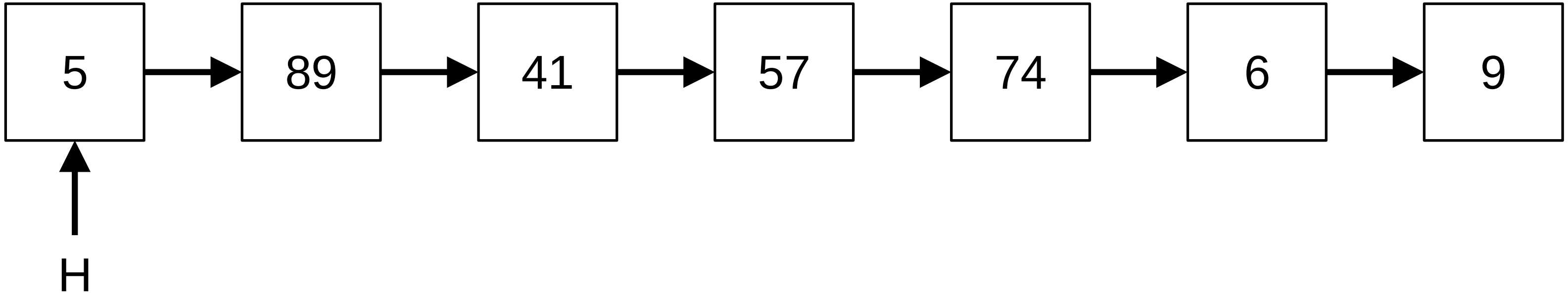
Dynamic Array

```
init:
    allocate N cells as A
    len = N;
    size = 0;


add(v):
    if size = len-1 then
        allocate 2N cells as B
        for i = 0 to N: B[i] = A[i]
        A=B
    A[size] = v
    size = size +1
```

```
get(i):
    if i > size then
        raise Error
    else
        return A[i]


set(i,v):
    if i > size then
        raise Error
    else
        A[i] = v
```

Linked List

Let’s insert the following *k* numbers to the linked list:
5,89,41,57,74,6,9



Linked List

```
init:
    head = new node;


addFront(v):
    n = new node; n.value = v;
    n.next = head; head = n;
```

```
addBack(v):
    n = new node; n.value = v;
    n.next = NULL;
    if head == NULL: head = n;
    else:
        curr = head;
        while curr.next != NULL:
            curr = curr.next

        curr.next = n;
```

Dynamic Arrays vs Linked Lists

- Memory footprint:
  - Dynamic arrays have O(n) wasted space (plus the expensive copy!)
  - Linked Lists have space overhead for pointer
- Access:
  - Dynamic arrays have O(1) access
  - Linked Lists have O(n) access

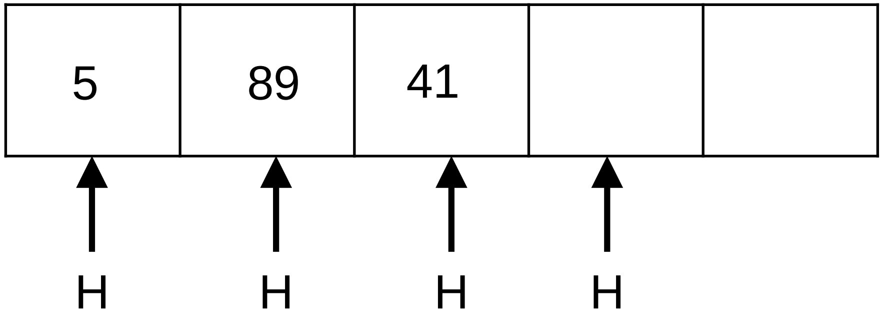
Queue

- First In, First Out (FIFO)

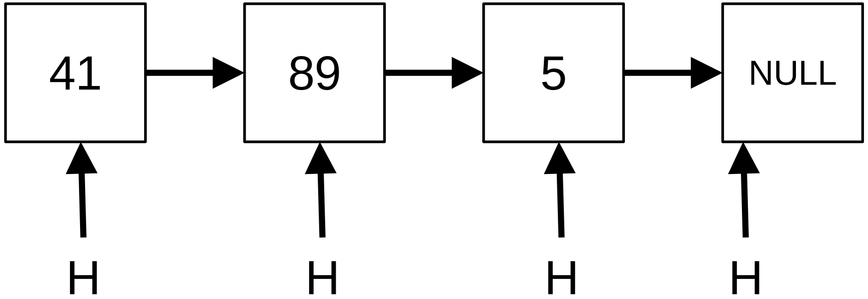
Stack

- First In, Last Out (FILO)

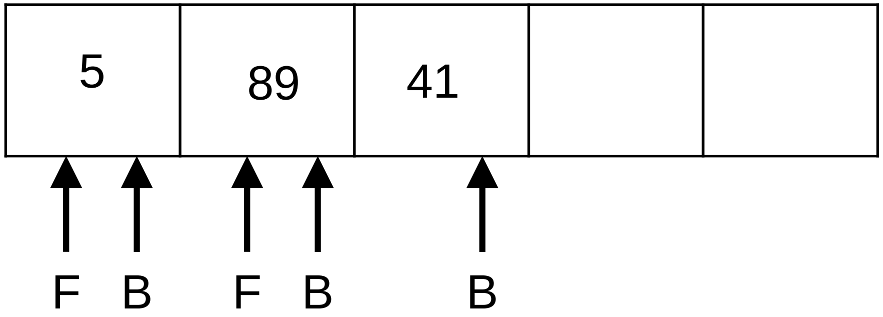
Stack with (Dynamic) Array



Stack with Linked List



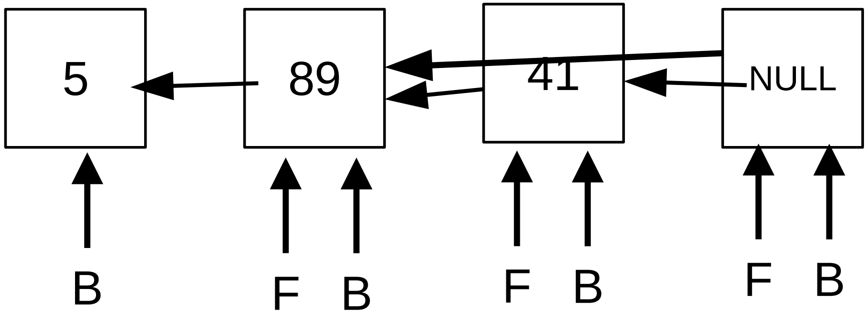
Queue with (Dynamic) Array



5	89	41		

F B F B B

Queue with Linked List



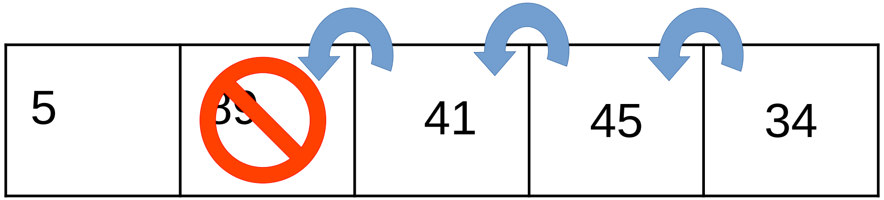
Sets and Dictionaries

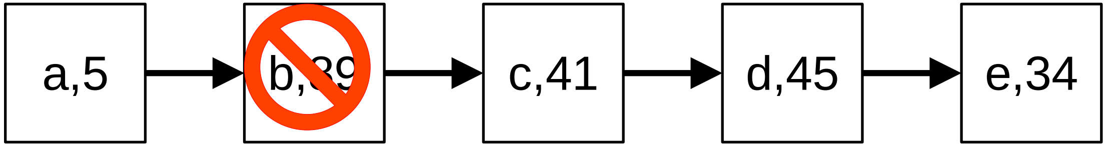
Set – *an unordered collection*

add(o) : Adds o to the set
remove(o) : Removes o from the set
find(o) : Returns TRUE if o is in the set

Dictionary – generalisation of arrays to non-int indices

insert(k,v): Inserts value v at k
remove(k): removes entry at k
find(k): returns value of that at k





remove(89) / remove(b)

Priority Queues and Disjoint Sets

Priority Queue – queue that is ordered by “priority”

insert(k,e): Insert element e with key k
RemoveMin(): Remove and return element with smallest key (maybe not unique)

Disjoint Set – Collection of disjoint sets that can be merged

add(x) : Create new set containing x
find(x) : find the set containing x
union(x,y): merge sets of x and y