Edinburgh Napier
UNIVERSITY

ALGORITHMS & DATA STRUCTURES (SET08122)

SEQUENTIAL DATA STRUCTURES

Dr Simon Wells
s.wells@napier.ac.uk
http://www.simonwells.org

Edinburgh Napier UNIVERSITY

TL/DR

We can organise data in memory so that the form of the organisation helps us to solve problems. These forms are **data structures** and their possible shapes are partly dictated by computer architecture.

Edinburgh Napier
UNIVERSITY

At the end of this topic you will:

- Understand how memory & data are related & organised
- Recognise the different core data structures: arrays, stacks queues, linked lists, hashes, tree, graphs &c.
- Have insight into the situations in which a given data structure is useful

Edinburgh Napier UNIVERSITY

OVERVIEW

1. Data & Memory
2. Methods for organising data in memory
3. Aggregate, sequential, associative, and non-linear data structures
4. Arrays, Lists, Stacks, Queues, Deques, Linked Lists
5. Hashes
6. Trees & Graphs

Edinburgh Napier
UNIVERSITY

DATA STRUCTURES

- Method for organising, managing, and storing data for efficient access & modification:
  - A collection of data (values, variables, &c.)
  - Relationships between them
  - Operations (Functions) that manipulate them

Edinburgh Napier UNIVERSITY

DATA & MEMORY

- Data stored in memory
- A lot of data often needed to model non-trivial problems
- Want to group things together in useful ways
- Memory organised logically as a huge, sequence of buckets
- Data of various types occupies various numbers of these buckets
- Each bucket is individual, it is not aware of it's surroundings - it doesn't know whether the data in the next bucket is related in any way
- So must manage not only our data, but also how it is organised in the buckets, and how those buckets are related to each other

This is what Data Structures is really all about...

Edinburgh Napier UNIVERSITY

ORGANISING MEMORY & DATA

- How to organise buckets
- Simplest way: Group similar buckets together. Decide how many buckets we need for our data. Allocate those buckets as a group and keep track of the first bucket.
- If we know which is our first bucket & how may we have we should be OK
- This is basically how Arrays work
- Major Drawback: need to group buckets together - in the real world this means allocating a *contiguous** series of buckets (memory locations)
- What if there isn't enough memory to allocate enough contiguous memory?

**meaning: Together in sequence. Having a common border. Touching*

NON-CONTIGUOUS MEMORY

Edinburgh Napier UNIVERSITY

- The only alternative with current hardware architectures is to allocate whatever memory we have non-contiguously, i.e. wherever it can be located (where the gaps are between other memory allocations), then to keep track of that.

- Many data structures are merely patterns for organising and tracking memory allocations.

- i.e. Store data values, perhaps grouped in some way according to needs of problem domain, the also store additional data about where *related* data is located in memory.

- Get's around the major drawback of arrays at the expense of complexity.

Edinburgh Napier UNIVERSITY

TYPES OF DATA STRUCTURE

- Aggregates: **Structs, Unions**
- Linear (Sequential): **Arrays**, Linked Lists, Stacks, Queues, Deques
- Linear (Associative): Dictionaries
- Non-Linear: (Binary) Trees, Graphs

***available in C**

Edinburgh Napier UNIVERSITY

AGGREGATES

- A way to cluster arbitrary collections of data together so that they can be treated as a related group
- In C we use structs a lot to do this
- Is easier to handle a group of things, that might be of different datatypes, but that need to be treated as logically similar
- Contents of a struct *highly* dependent upon specific programming problem, e.g.
  - Modelling a person, might group personal information about them such as Name, Date of Birth, Favourite Star Trek Movie, &c.
- In other languages, these are data that we might group together in a class

Edinburgh Napier UNIVERSITY

SEQUENTIAL STRUCTURES

- A way to organise primitive datatypes in relation to each other - as various kinds of linear sequence - no branches
- Arrays are our core sequential data structure
  - Can use arrays to implement other kinds of data structure (subject to the same limitations of Arrays) using contiguous memory, or,
  - Can replace array with a non-contiguous structure
- Leads to two forms:
  - Contiguous: Each element of the sequence is next to a neighbour until you get to the end
  - Non-Contiguous: Each element of the sequence stores both it own data & metadata about the location of the next & possibly also the previous element in the sequence

Edinburgh Napier UNIVERSITY

SEQUENTIAL APIS

- If we have a sequence of elements then its shapes suggests natural ways to interact with it,
  - e.g. from each end (assume one end is the start/head & the other is the end/tail)
  - If we start at one end, than can move from one element to the next until we reach the other end
- By restricting how we interact with the sequence we get the behaviour of various data structures, e.g. stack, queue, deque (logical variations on the theme of the sequential API)

Edinburgh Napier UNIVERSITY

ASSOCIATIVE STRUCTURES

- Hash tables are an example of an associative structure
- Might be aware of them under different names, e.g. map, dictionary, associative array
- A collection of pairs comprising a key and a value
- Given the key, can retrieve the value
- Think of directly accessing an array by address, e.g. array[5] but instead of an array index you can use some arbitrary value.

Edinburgh Napier
UNIVERSITY

IF A SEQUENTIAL STRUCTURE JUST POINTS TO THE NEXT & PREVIOUS ELEMENTS

COULD WE ORGANISE THESE LINKS TO POINT TO OTHER NODES IN A DIFFERENT PATTERN?

Edinburgh Napier
UNIVERSITY

Edinburgh Napier UNIVERSITY

NON-LINEAR STRUCTURES

- Instead of pointing to next & previous, lets:
  - Point to children - leads to a form of a tree data structure
  - Arbitrarily point to other elements (possibly including itself) - leads to a graph (network) data structure
    - NB. Various forms of trees & graphs depending on restrictions on the links (we'll look at these later)
    - A tree is a sub-type of graph in which there are no-cycles

Edinburgh Napier UNIVERSITY

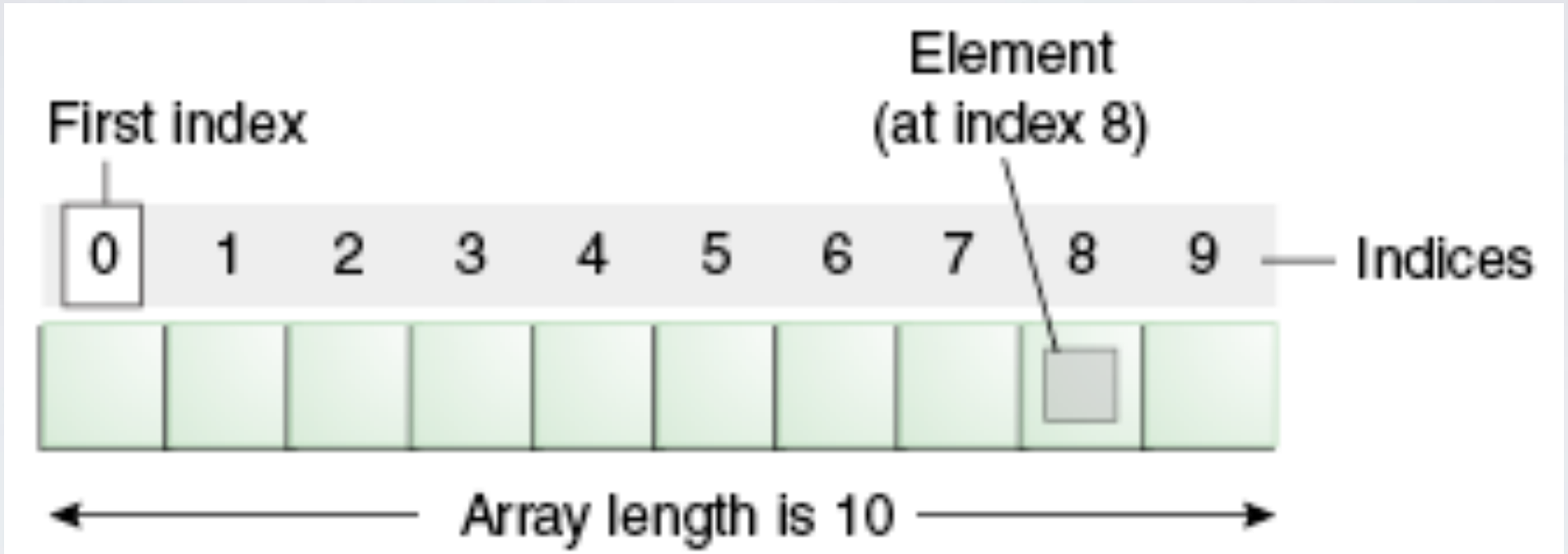
ARRAYS

- A data structure based upon the contiguous allocation of memory

- Remember our buckets?

- An array is a series of buckets that we can store our data in - however each bucket in the array is touching the next. We must keep track of how many buckets we have (or risk disaster)

- Arrays are useful on their own, but are very useful in the implementation of other data structures.


Element
(at index 8)
First index
0
1
2
3
4
5
6
7
8
9
Indices
Array length is 10

Edinburgh Napier UNIVERSITY

ASIDE: ARRAYS & LANGUAGES

- Programming languages, particularly higher level ones, generally implemented using lower level languages
    - Some weirdness when you get to higher-level interpreted languages like Java, C#, Python due to VM. Language itself *may* be self hosted, but the VM is often written in a lower level language. Different VMs can be implemented in different languages yielding different characteristics
- Overcoming the drawbacks of the basic, fixed size array is a spur to development of many other structures:
    - e.g. ArrayList, Vector (Java), Lists (Python), Dynamic arrays in general
        - Desire to store an unknown amount of data without having to deal with the details of increasing the size of the array whenever it runs out of room
        - *Always want more space for new data, but don't want to pre-allocate space unless needed.*

Edinburgh Napier UNIVERSITY

ASIDE: LISTS

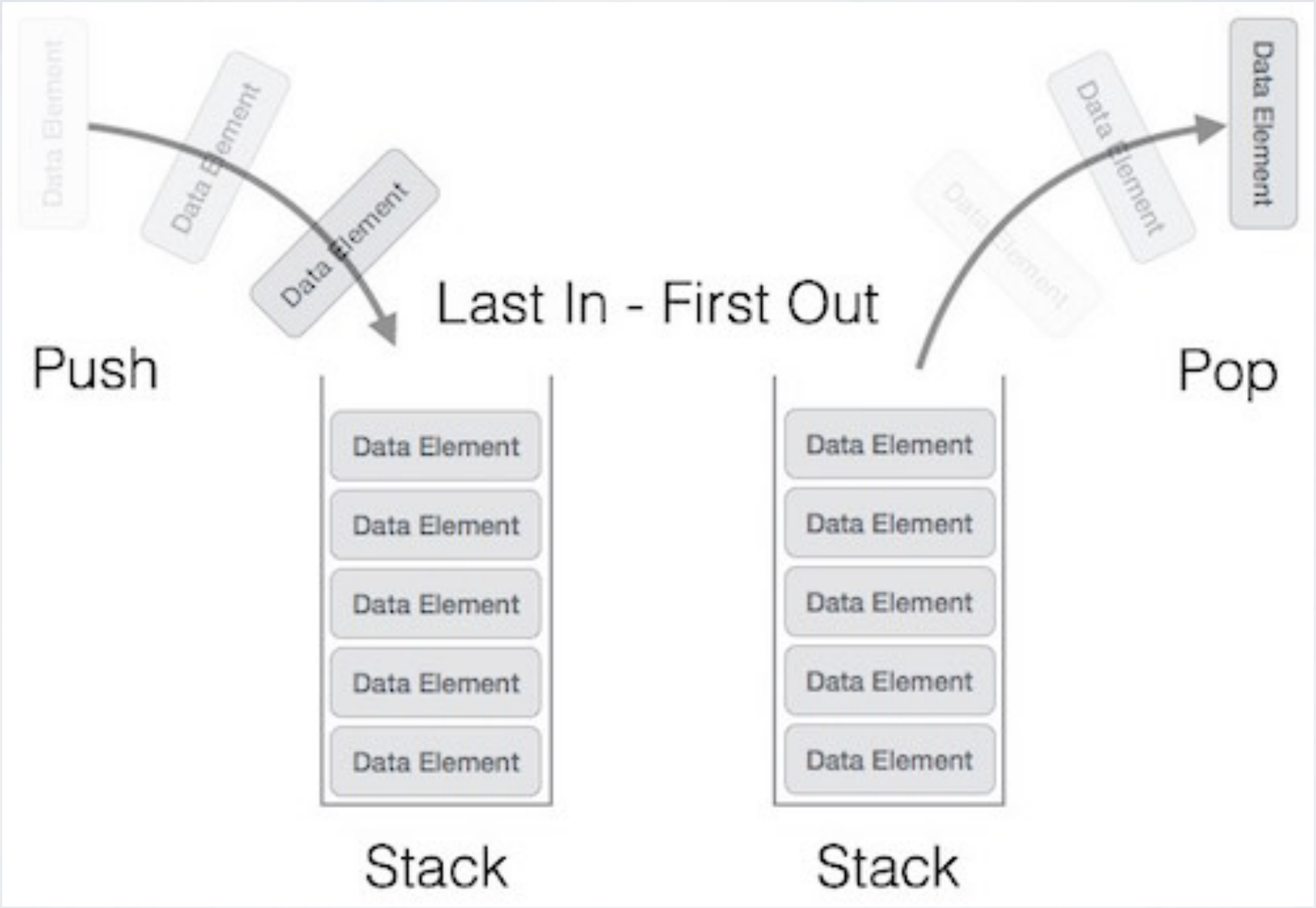
- Not *linked list* but a dynamic, sequential collection
- Found in many languages: Python (lists), Java (ArrayList/Vector), List (list), Prolog (list)
- Used like an array (but without some of the drawbacks)
- Treat as a sequential structure (despite implementation differences)
- Dynamic (extended and compacted)
- Managed (don't have to manage allocation & deallocation)
- *They grow as required? Is this magic?*

Edinburgh Napier UNIVERSITY

STACKS

- Think of a stack of trays in a canteen - this is an almost archetypal physical real-world implementation of a stack
- Stacks are collections that are based on sequential structures so member elements are organised as a sequence
- They have a well defined interface: The stack has a top & a bottom. Items can be **pushed** on to the top of the stack and **popped** off of the top also.
- Often have non-essential operations implemented, e.g. "peek" - observe but don't remove
- Items within the stack are not accessible within the definition of the stack API until they are at the top - then they can be popped (NB. pragmatism)
- The **last item in is the first item out** (**LIFO**) - means that if we have a finite sequence of elements then it is trivial to reverse that sequence.
- Stacks don't specify how the underlying collection should be implemented:
    - An array is an easy approach - limited size, need reorganising
    - Linked Lists are an alternative - increased flexibility & complexity
- However underlying collection is implemented, still need to track the top and bottom of the stack (depending on how you're filling the array)

STACK GRAPHIC



Edinburgh Napier UNIVERSITY

USES OF STACKS

- Reverse a string (or any other sequential collection)
- Evaluating expressions (reverse polish notation for arithmetic and for some programming languages)
- Keeping track of user actions (useful for implementing undo features)
- Backtracking - looking for a path through a maze
- Part of implementation of many languages, e.g. the stack machine model
- Anything else?

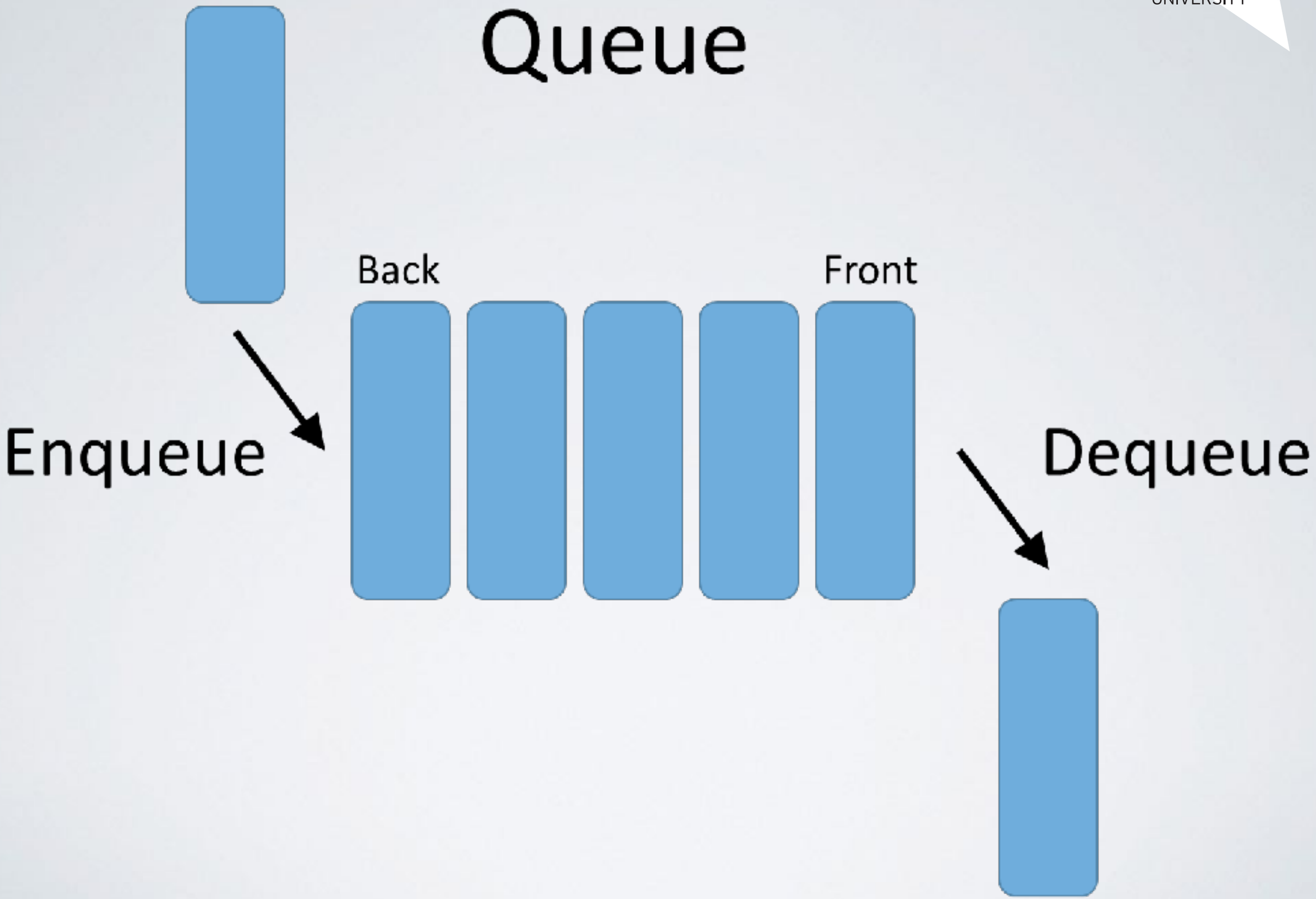
Edinburgh Napier UNIVERSITY

QUEUES

- Familiar to everyone. Who's been in a queue today?
- People waiting in line is an archetypal real-world example of queuing
- Just like stacks, based on a sequential collection of data but the interface, how we can use that data is different.
    - Queue has a front and a back
    - Elements are **enqueued** (added to the back) and **dequeued** (removed or "serviced" from the front)
    - This leads to a natural order in how elements are dealt with. In a queue the oldest element is dealt with first and the newest must wait it's turn (depends on when it joined the queue)
        - Leads to a **first in first out** (**FIFO**) data structure - elements are dealt with in the order in which they arrive
- Easily implemented by defining an API for how we interact with an Array. Can be implemented using other sequential structures, like linked lists (with same caveats as for Stacks).
- NB. We have the start and the end of the underlying collection, e.g. Array, but also need to keep track of the front and rear of the queue

Edinburgh Napier
UNIVERSITY

Queue



Edinburgh Napier
UNIVERSITY

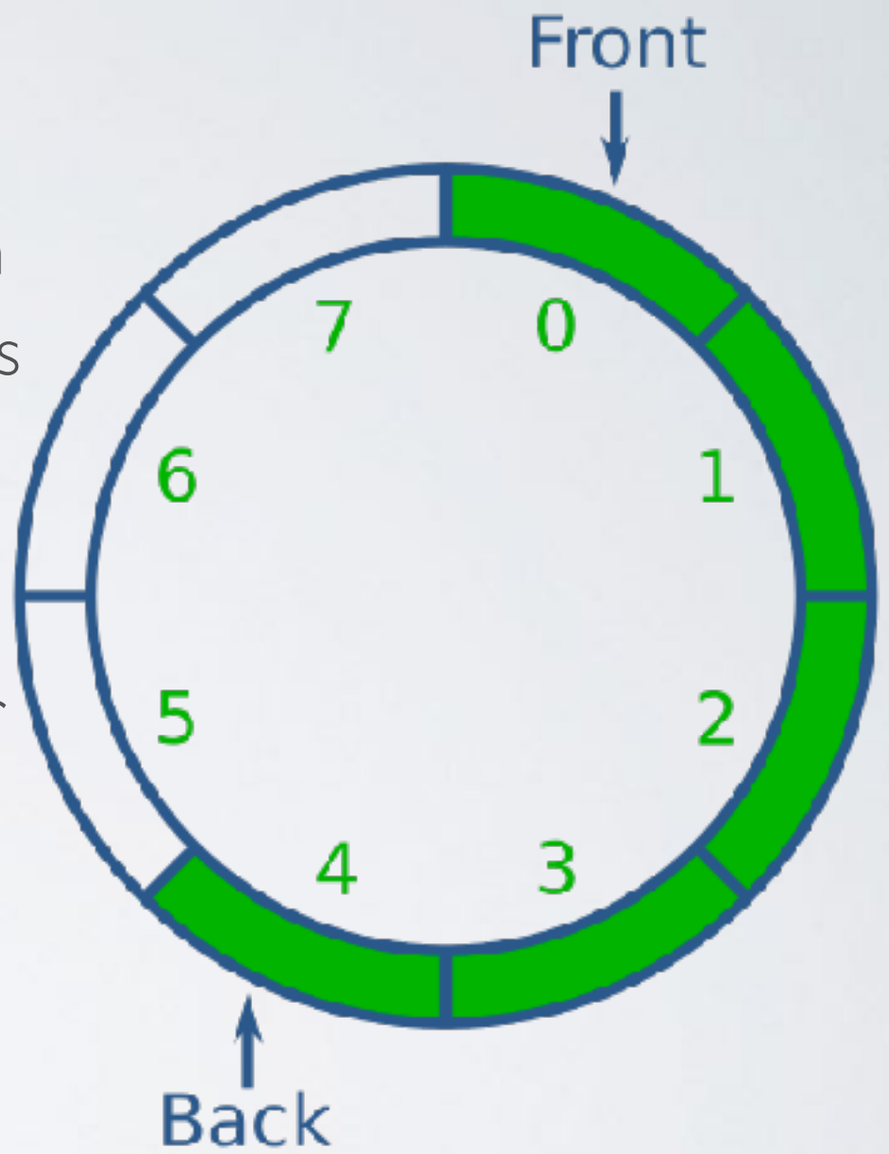
USES OF QUEUES

- Scheduling, e.g. CPU, HDD
- Buffers, e.g. communication buffers, particularly when data is transmitted asynchronously, i.e. received at different rate to which it is sent.
- Messaging, e.g. message queues
- Job queues
- Store & replay order of moves in a game

VARIATION: CIRCULAR QUEUE

- One of the complexities of an array implementation of a queue is that elements need to be moved up each time one is dequed
  - The queue is fixed in size so;
    - Each item enqueued gets us closer to the end of the array
    - Each item dequeued leaves a gap between the start of the array and the first element in it
    - So must shift our elements along each time we dequeue so that the gap at the front is closed and there is more space at the end
    - Means dequeuing involves shifting a potentially large number of array contents

- Can remove need for moving queue contents within the array by making the array circular
- When we get to the end of the array we loop around to the beginning again.
- The back of the queue is always chasing the front in one direction (as elements enqueued) whilst front is chasing rear in other direction (as elements dequeued)
- This "circular queue" is also known as a (ring) buffer
- Useful if your problem requires LIFO storage of a fixed size (some communication protocols)
- Also useful in "producer-consumer problem" scenarios - If consumer is unable to keep up then new data will overwrite older data which is discarded



Edinburgh Napier UNIVERSITY

DEQUES

- A version of a queue in which elements can be added to or removed from either end
- Can be further specialised by restricting this behaviour, e.g.
    - **Input restricted**: Remove from either end but insert at one end only
    - **Output Restricted**: Insert at either end but delete from one end only
    - Also easily end up with:
        - **Stack** (add/remove from one end only)
        - **Queue** (add at one end remove from other)
- So a Deque is a kind of archetype structure for the queue and stack.
- A List can usually support all the behaviours of a Deque (but generally has many more features than a deque should officially support)
    - This kind of brings up full circle

DEQUE GRAPHIC

Edinburgh Napier UNIVERSITY

USE OF DEQUES

- This is more specialist
- Often the structures we've already seen are adequate, however…
- Ageing items in a fixed size stack (e.g. undo feature or "back" button).
  - Fixed size because don't want to store everything
  - Contents are pushed in from top, to give age ordering over contents (oldest @ bottom, youngest @ top)
  - When stack is full, remove oldest item from bottom to make a new space at the top.
  - Strictly can't do this with a pure stack as that only supports adding and removing at the top (& expected behaviour is for stacks to overflow when full rather than discarding their lowest items)

Edinburgh Napier
UNIVERSITY

USE OF DEQUES

- A-STEAL — An adaptive, work-stealing algorithm for distributing jobs (job scheduling) across processors [He *et al.* (2012) "Provably Efficient Two-Level Adaptive Scheduling"] - used to underpin distributed processing libraries, e.g. in C++

- More generally:
  - When you have a chain of items to model in your program where these can be processed at each end but not in the middle, e.g. modelling a train on a railway track: a sequence of carriages where the front can be detached, or the last, but generally the middle carriages are stuck in the middle until the ends have been dealt with.

  - When you have sequence of elements where you want to compare each end of the sequence

Edinburgh Napier UNIVERSITY

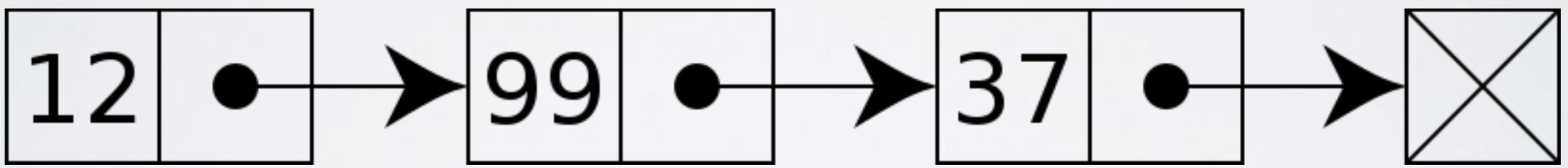
LINKED LISTS

- An alternative to the dynamic array that overcomes some of the limitations of the basic array
    - but introduces further complications
- A list created by linking discrete items of data together so that they are spread across memory
    - Start with one item, the head of the list
    - Each item "points" to the next item
    - Move through the sequence by accessing the current item & following the link to the next item
- Many structures can be implemented quite efficiently using linked lists, e.g. there are linked list based implementations of Deques (which can in turn be specialised to provide Stack & Queue structures)

Edinburgh Napier UNIVERSITY

SINGLY LINKED LISTS

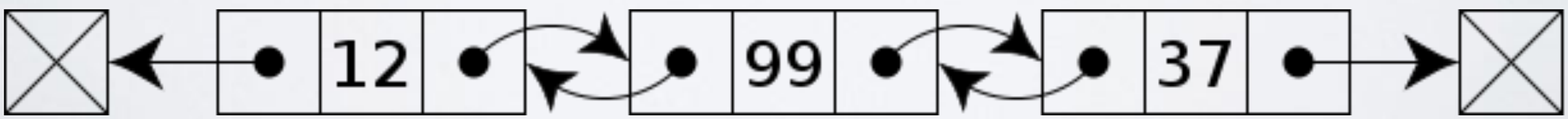
- Each element stores it's own data and a pointer to the next elementSingly Linked List Graphic



Edinburgh Napier UNIVERSITY

DOUBLY LINKED LIST

- Each element stores it's own data and a pointer to both the next element and the previous element
- Can think of this as two singly-linked lists, each containing the same data but each sequence being the opposite order of the other
- Can traverse the list in either direction
- Head and tail immediately accessible (no traversal necessary) all internal elements require traversal



Edinburgh Napier UNIVERSITY

LINKED LIST CHARACTERISTICS

- Constant time insertion and deletion - no reallocation or reorganisation needed - because we've removed the contiguous constraint on memory layout
- Dynamic can grow or shrink as necessary
- Use more memory than arrays
- No random access to or efficient indexing of data in the list
- Many common operations: getting last node, finding a particular node containing specific data, locating insertion point for new data - require iterating through the list

Edinburgh Napier UNIVERSITY

LINKED LIST USES

- Lists (dynamic arrays), Stacks, Queues, Deques can all be implemented using a linked list instead of a dynamic array
- Understanding the basic linked list structure is useful because it gives a foundation for understanding other structures, e.g
    - Trees:
        - Head of the list is the root node of the tree
        - Each node points to further nodes
        - Depending upon the type of tree, each node points to 2 or more *child* nodes
- Graphs:
    - Trees are a sub-type of graph
    - A graph is made of nodes but has no root node
    - Each node can point to any other node
    - In some graphs nodes might point to each other
    - In some graphs nodes might even point to themselves
- We will see both of these structures in subsequent lectures (Powers will talk about Trees & Hart will talk about Graphs)

Edinburgh Napier UNIVERSITY

ACCESSING DATA

- Until now, once data has been added to a data structure we have had two basic ways to retrieve it:

1. By index - when a structure supports direct indexing
   - e.g. Arrays - really only useful if something has stored the index or if iterating over the structure or if the index represents something about the world
2. By value - search through the structure until we find our target value. then retrieve that value & any associated data (assume a struct)

Sometimes it would be nice to directly access our data by some value that is relevant to our problem domain...

Edinburgh Napier UNIVERSITY

KEYS & VALUES

- Think of keys as really flexible array indices…
- Instead of just a number it can be useful to use real data as an index and associate that index with a further set of data (the values associated with the key)
  - e.g. Store user records by some useful identifier such as an email address - if an email address is supplied the retrieve the user data stored with that address
  - Assume the key is unique and maps to one set of data.

Edinburgh Napier UNIVERSITY

KEY:VALUE STORES

- Key:value stores (databases) are very popular ways to store data as an alternative to relational, graph, document, or column stores (NB. If you weren't yet aware, there are *lots* of different types of database out there [many with quite specialist applications])

- Need to have a quick way to look up which users are currently logged in?
  - Key value store contains email-address:boolean data, e.g.

    s.wells@napier.ac.uk:true, s.powers@napier.ac.uk:false

- When a person logs in, set to true, when they log out remove the entry (or set to false) - Logged in status is now just a lookup away (can be quicker than a traditional database as many key:value stores are in memory (store everything in RAM)

- Examples include: Redis, Riak, Project Voldemort, Berkeley DB, Memcached, Dynamo, &c.

Edinburgh Napier UNIVERSITY

KEY:VALUE DATA STRUCTURES

- Not just databases
- Basic data structures provide key:value structure as well
  - e.g. HashTable (map, dictionary, symbol table, associative array)
  - Appears in many languages (Python, Java, JavaScript, &c.)
- The collection comprises a set of key:value pairs such that each possible key appears at most once in the collection

Edinburgh Napier UNIVERSITY

ASSOCIATIVE ARRAYS

- Typically support the following API:
  - Add a key:value pair to the collection
  - Remove a key:value pair from the collection
  - Modify an existing key:value pair within the collection
  - Lookup the value associated with a particular key
- N.B. The value doesn't have to be a basic data type, can be a more complex structure.

Edinburgh Napier UNIVERSITY

IMPLEMENTATION

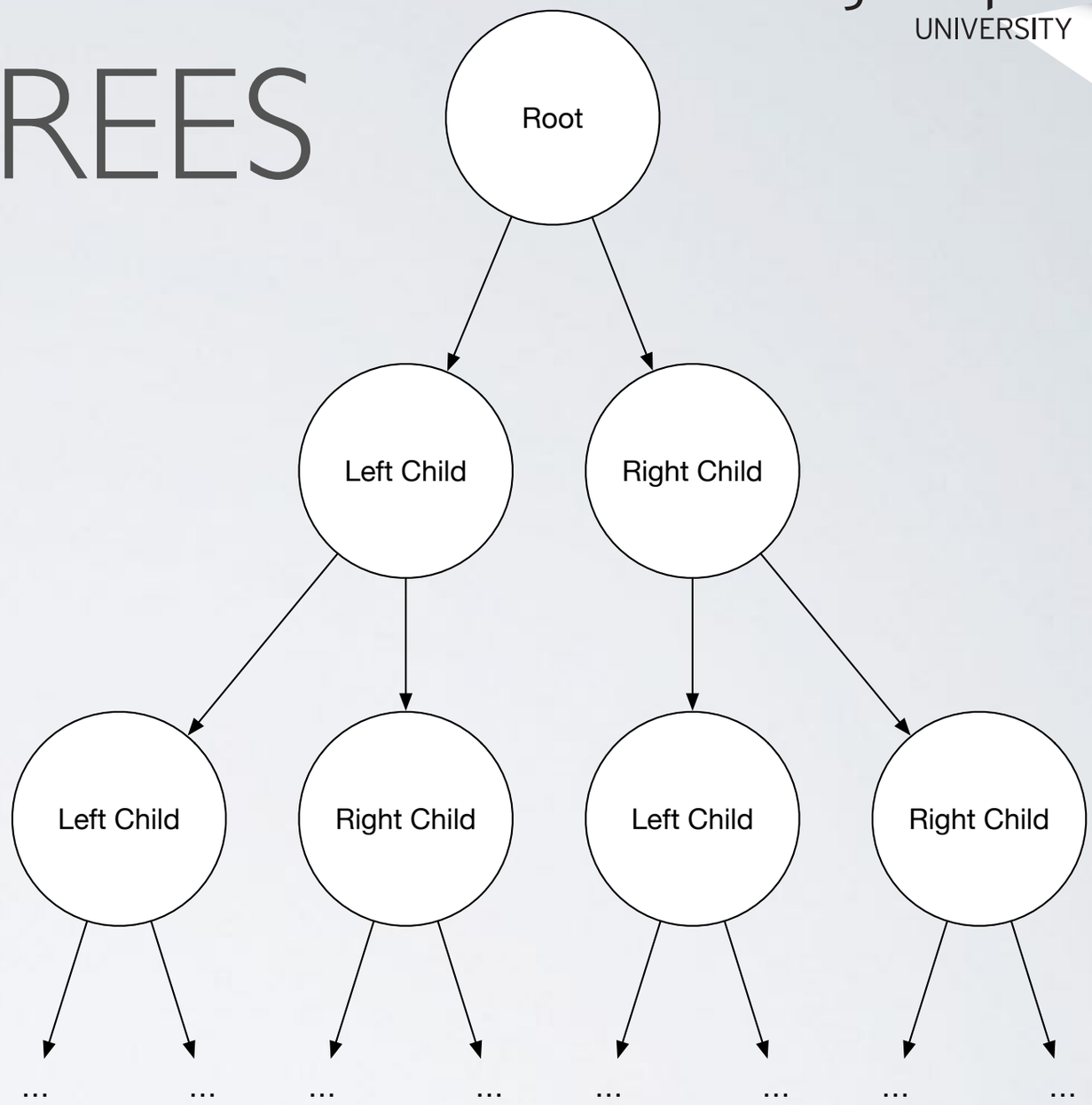
- Various ways to implement associative arrays
- Key problem is mapping a key to a unique location (array index, memory address, list position, etc.) - needs to be a way to take a key, such as an email address and turn that into a particular location where the value is stored
- Want to do this is rapidly as possible, i.e. in (near) constant time
- In our lab example we used the hashCode() function:
  - Array has a certain length so turn key into a number modulo array size
  - If location is already filled, check next location. Repeat
  - Can lead to bad performance, e.g. checking every array location looking for the key
- Mapping keys to locations in storage is a problem known as hashing - there are *many* hashing algorithms of varying performance
- Ideal solution is a perfect hash that maps every key to a single unique storage location without **clashing**
- We'll return to hashing algorithms later in the module.

Edinburgh Napier UNIVERSITY

NON-LINEAR STRUCTURES

- So far our data structures have been very linear, e.g. arrays, lists, even hash tables (follows for those based on linear storage, e.g. stacks, queues, deques, …)
- What about non-linear structures?
  - Non-linear & hierarchical: trees
  - Non-linear & non-hierarchical: graphs
- Consider these as like linked lists, they consist of a collection of nodes that point to each other but the pattern of how they point to each other is different

TREES



- Hierarchical
- Start with root node
- Each node can have children
  - e.g. up to two children (left child & right child) gives a binary tree
- Various types of tree available allowing various numbers of children, being constructed in various ways, being balanced, ordered

Edinburgh Napier
UNIVERSITY

USES OF TREES

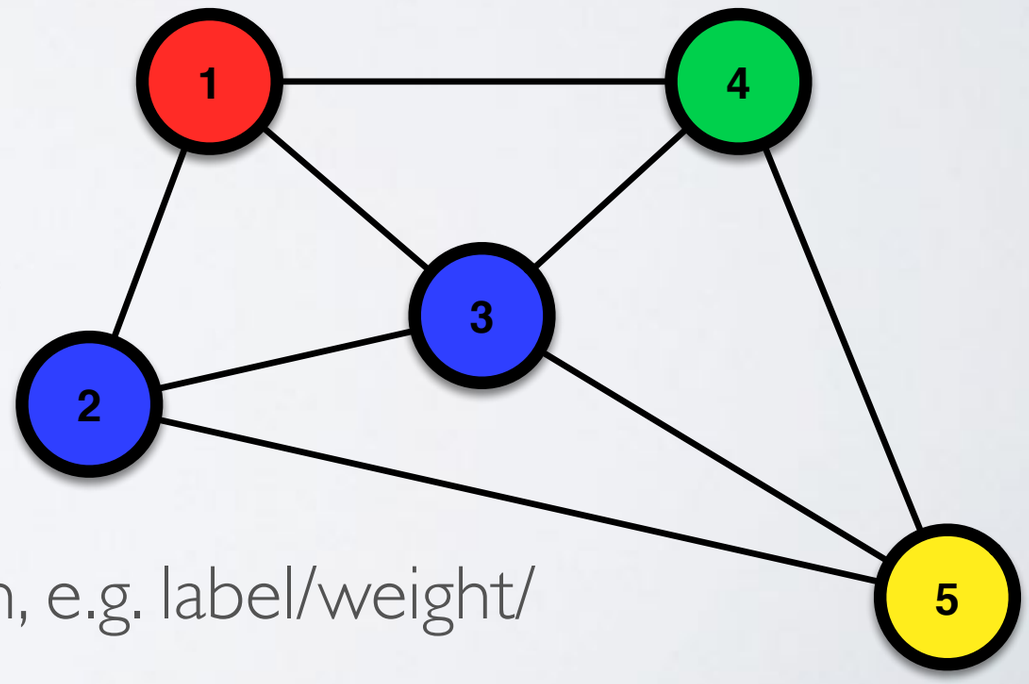
- Represent hierarchy, e.g. structure of natural (& un-natural) languages - used to process computer languages & represent structure of sentences in linguistics, file-systems, &c.

- Index & search data, e.g. within databases, to provide reasonable access times, e.g. slower than arrays but quicker than linked lists. Some types of tree provide upper bounds of $O(\log n)$ on search

Edinburgh Napier UNIVERSITY

GRAPHS

- Another non-linear data structure used a lot on mathematics
- Also non-hierarchical
  - Whilst tree has a root node, there is no root node in a graph
  - Any node in a graph can point to any other node
- Formally:
  - A set of nodes (vertices) & a set of edges
  - The edges connect the nodes
  - Nodes & Edges can also store information, e.g. label/weight/ length/distance on edges

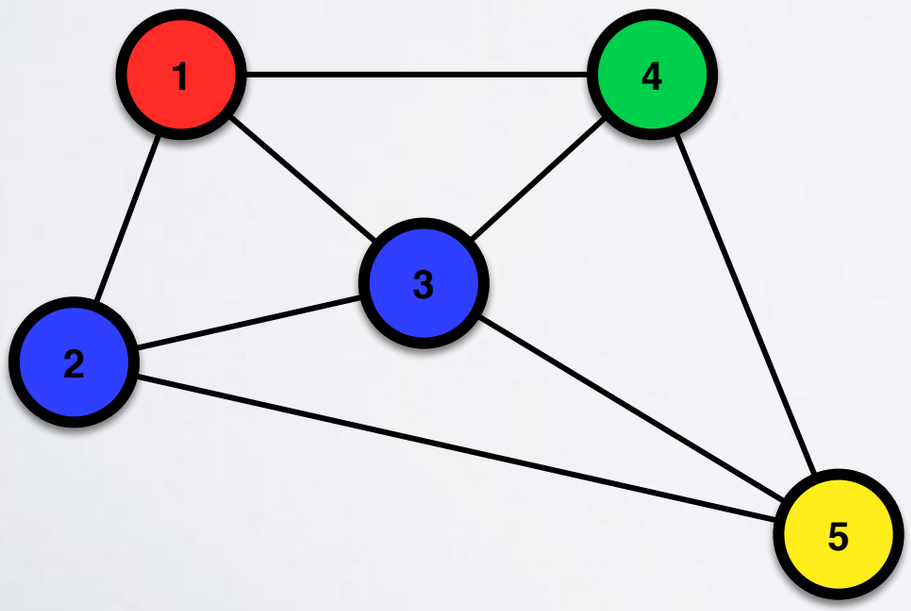


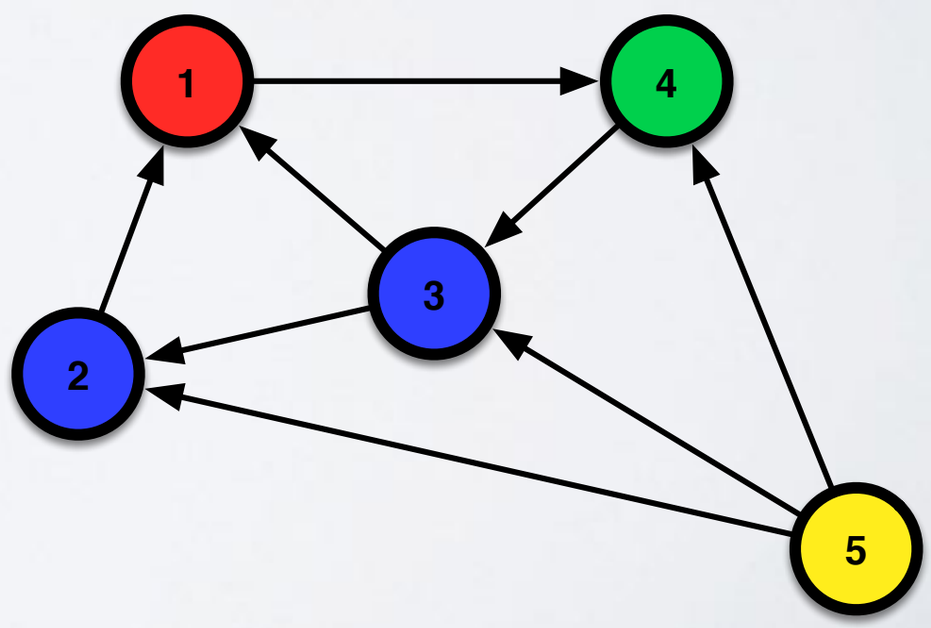
Edinburgh Napier
UNIVERSITY

DIRECTED & UNDIRECTED

- Edges can have a direction, e.g. pointing from one node to another - **directed**

- or else they don't - **undirected** graphs have edges that merely link nodes together





Edinburgh Napier UNIVERSITY

GRAPH IMPLEMENTATION

- We looked at an **adjacency list** implementation:
    - Maintain a list of vertices. For each vertex, maintain a list of links to other vertices that it is connected to.
- Other approaches are the **adjacency matrix** (2D matrix where row shows source and column shows target. Value shows edge existence or weight. Data in each node is stored separately). & the **incidence matrix** (2D matrix of booleans - rows are vertices & columns are edges. Entries indicate incidence of specific vertex & edge).
    - Choice of implementation based upon nature of problem (affects storage, time to add/remove edges/vertices, time to test/retrieve substructures, e.g. adjacency):

Edinburgh Napier UNIVERSITY

GRAPH API

- API is specific to graph structure
- Complexity & structure of graphs leads to a set of operations different to those of other structures we've seen so far:
- **Adjacent** - test whether any pair of nodes is connected by a single edge
- **Neighbours** - list all nodes connected to this one
- **edge/vertex addition/removal**
- **get/set vertex value(s)**
- **get/set edge value(s)** - *where appropriate*

Edinburgh Napier UNIVERSITY

USES OF GRAPHS

- Graphs often match the structure of real world problems very closely.
- The world comprises many inter-related elements
  - Selecting those elements to model
    - & the relationships between those elements to capture
      - is a standard aspect of programming (solving real world problems)
        - Graphs provide a flexible basic structure that naturally fits with the complexity of the world
- Social networks - connections between people: Facebook social graph

Edinburgh Napier UNIVERSITY

TREES & GRAPHS

- Trees are a sub-type of graph
- If a directed graph has no cycles and each node has only one incoming node then the graph forms a type of tree
- Many graph algorithms involve identifying a tree that has particular properties, e.g a spanning tree:
    - A **spanning tree** is an undirected tree comprising only those edges necessary to connect all the vertices in a graph
    - For any pair of vertices there is only one path between them
    - Adding an edge to a spanning tree will create a unique cycle
    - The particular spanning tree that you get depends upon the criteria for generating it, e.g. minimum spanning tree - the sum of edge weights is as small as possible (Prim's algorithm, Kruskal's algorithm)
    - Used in route finding, electrical circuit analysis,…
        - e.g. Find the minimum length of cable required to wire a neighbourhood for fibre (connect each house once)

Often: Graph represents the data but answering specific questions about that data can involve identifying a tree within that graph

Will return to this relationship again later in the module.

Edinburgh Napier UNIVERSITY

PRAGMATIC CONCLUSIONS

- Queues are FIFO | Stacks are LIFO
- Queue? - get things back in the order in which they were added
- Stack? - get things back in reverse order of addition
- List? - get things back regardless of addition order & don't want automatic removal
- Associative Stores? Great flexibility for retrieving based on domain feature (key) rather than location
- Trees? Great when you have hierarchy
- Graphs? Great when you have complex inter-related data
- **Pragmatically:** Stacks & Queues are tools for completing specific tasks rather than long term storage of core data (which lists, trees, graphs & associative stores are great for)

Edinburgh Napier UNIVERSITY

IN SUMMARY...

- We've come full circle. Starting with a basic data structure (array), we added dynamism (Lists), restricted behaviours for modelling clarity (Stacks, Queues), and discovered an archetypal structure (Deque) which is essentially a (sub-) type of List.

- An important point here is that pragmatically a list and a Deque are the same basic collection but theoretically a Deque only has the operations of a deck (insert [front|back], remove [front|back], peek [fron|back]) rather than the full complement of List functions.

- Many structures follow from the basic layout of memory & lead to various linear (contiguous & non-contiguous) structures. Non-linear & Associative structures identify other ways to organise memory that lead to flexible, but complex structures.

NEXT…

- Foundation for exploring:
  - Performance characteristics, computability, &c.
- Algorithm selection & design (algorithms generally operate on data. The way the data is structured can have an effect on performance, complexity, ease of use)
- Concentrate on searching & sorting for a few weeks before returning to hashing, trees, & graphs

Edinburgh Napier
UNIVERSITY

QUESTIONS ???

Edinburgh Napier UNIVERSITY

WHAT DID WE LEARN?

- *We can now…*
- Understand how memory & data are related & organised
- Recognise the different core data structures: arrays, stacks queues, linked lists, hashes, trees, queues &c.
- Have insight into the situations in which a given data structure is useful