Linked Lists

- They are a type of **data structure** like arrays, hashes, objects, etc.
- They are **linear**, which means, we traverse through elements or nodes sequencially

Difference between Arrays and Linked Lists

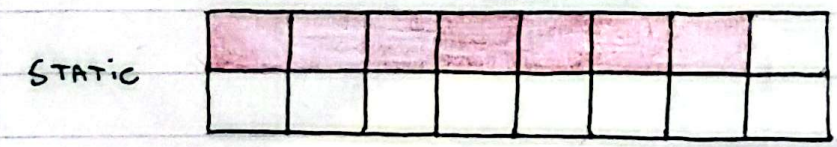
- Memory management

Note If we work with dynamically typed languages like Python, Js or Ruby, we don't have to worry about how much memory an array uses because there are several layers of abstraction that end up with us not having to worry about memory allocation at all. But that doesn't mean memory allocation doesn't happen.

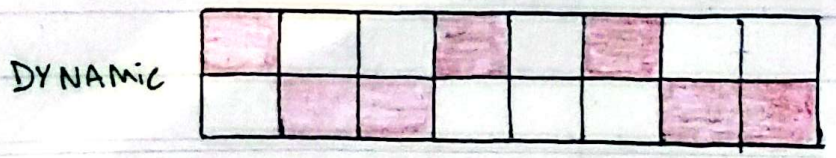
Array
- Needs memory all together in one big block.
- If you want to allocate 7 letters, the computer must find 7 bytes next to each other

Linked Lists → Each piece can be stored anywhere in memory.

Memory Allocation

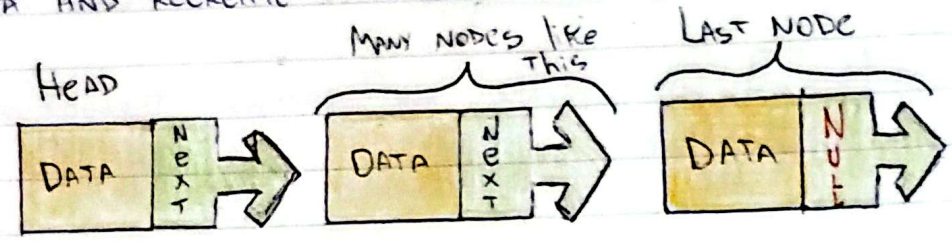


Arrays need a contigous block of memory



Linked Lists don't need to be contigous in memory. They can grow dynamically.

NOTE If we add more to an array, eventually it won't have enough space. At that point we need to copy the data and recreate it somewhere else with more memory.



- Adding or removing a node with a linked list becomes as simple as rearranging some pointers, rather than copying the elements of an array.

Types of linked lists

1 Singly Linked List

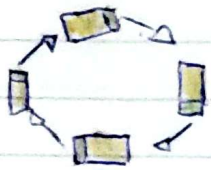
- Each node has:
  - Data
  - Pointer to the next node
- Only goes forward
- Simpler and uses less memory.

2 Doubly Linked List (DLL)

- Each node has:
  - Data
  - Pointer to the next node.
  - Pointer to the previous node
- Goes forward and backwards
- More flexible but uses more memory.

3 Circular Linked List

- Last node points back to the first node, forming a circle
- Can be singly or doubly circular.
- Example: Carousel of images



Drawbacks

1. More memory per element
   - Each node stores not only the data but also a pointer (or two, in Doubly Linked Lists)
   - The extra memory usage can be significant.

2. Poor cache locality
   - Nodes are scattered across memory
   - Arrays are stored next to each other, so they are much faster to access because of CPU caching

3. Slower Access
   - To get to the N-th element, you must start at the head and follow the links.
   - Arrays can access the N-th element instantly with indexing.

4. Extra complexity.
   - Insertion and deletion logic is trickier (need to manage pointers carefully)
   - In low level languages it's easier to make mistakes with pointers.

5. Not ideal for small/medium datasets.
   - The extra memory cost usually isn't worth it, unless you do a lot of insertions/deletions in the middle.