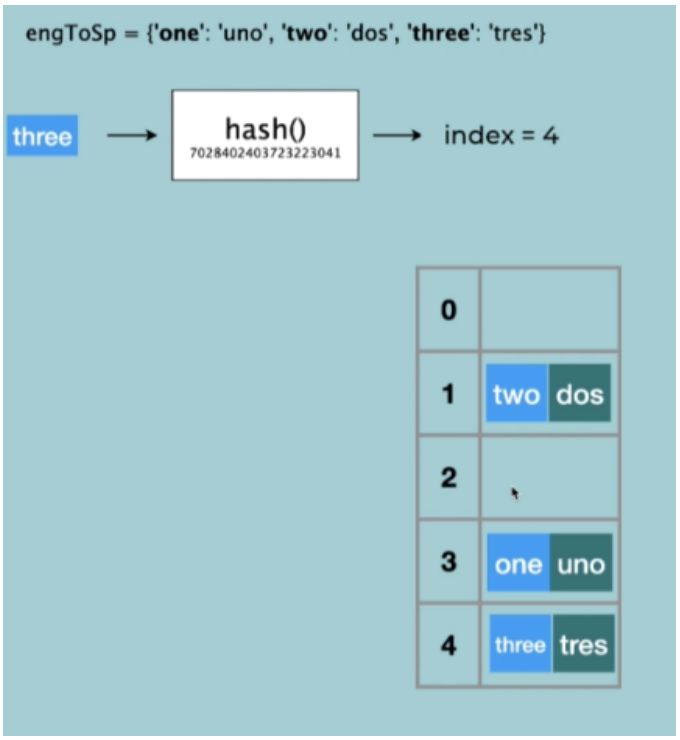
Dictionary

- **What is a Dictionary?**
  Its a collection that is unordered, changeable and indexed,
- **Difference w/ Array?**
  Array value can be reached by index arr[2]= 6, while, Dict value can be reached via its Key + the keys can be of any type(int, string…)
- **Implementation in Memory?**
  - ★ Via a hash table! How?
  - ★ Once we get the pair (key:value)
  - ★ A hash() function will calculate the index of this key
  - ★ After that, it insert this pait to the gotten index and move to the next pair!
  - ★ What if after calculation have the same number? Add at the end as a linked list

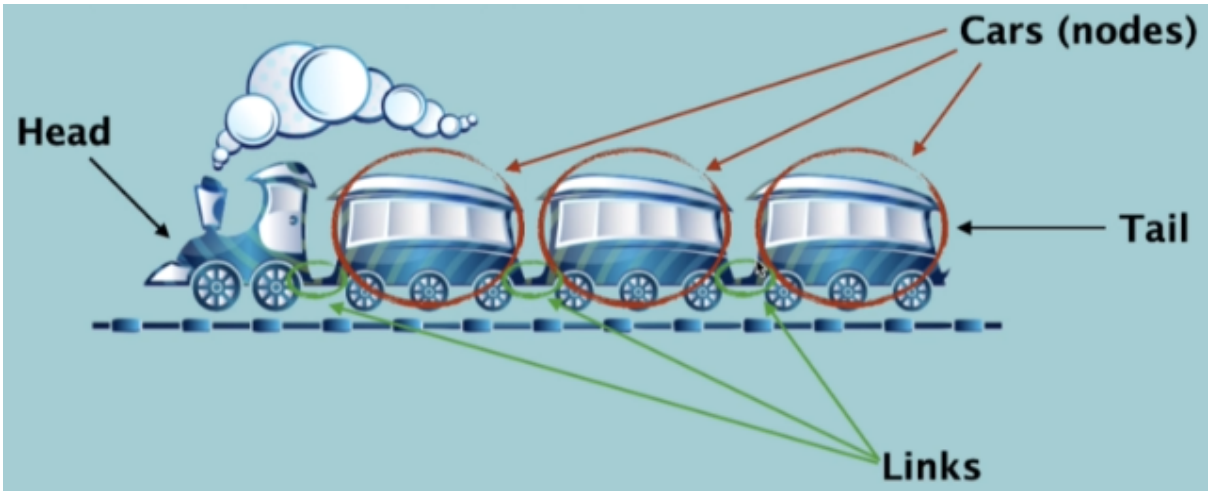


Tuple:

- A collection of values of any type, but Immutable!
- hashable : if it has a value that can have the same value in its lifetime!

LinkedList:

- A form of sequential collection contains nodes in which it has the data + a reference to the next node!



LinkedList vs. Train:

★ Train compose of Cars(nodes in LinkedList) and link to connect the cars(links in LinkedList)

★ Both have head n tail!

★ Cars are Independent! What? It means that in case we don't need a car in the train we can simply remove it while the train yet works! Likewise, if we need an additional car in the train, then we can simply add it to the Train!

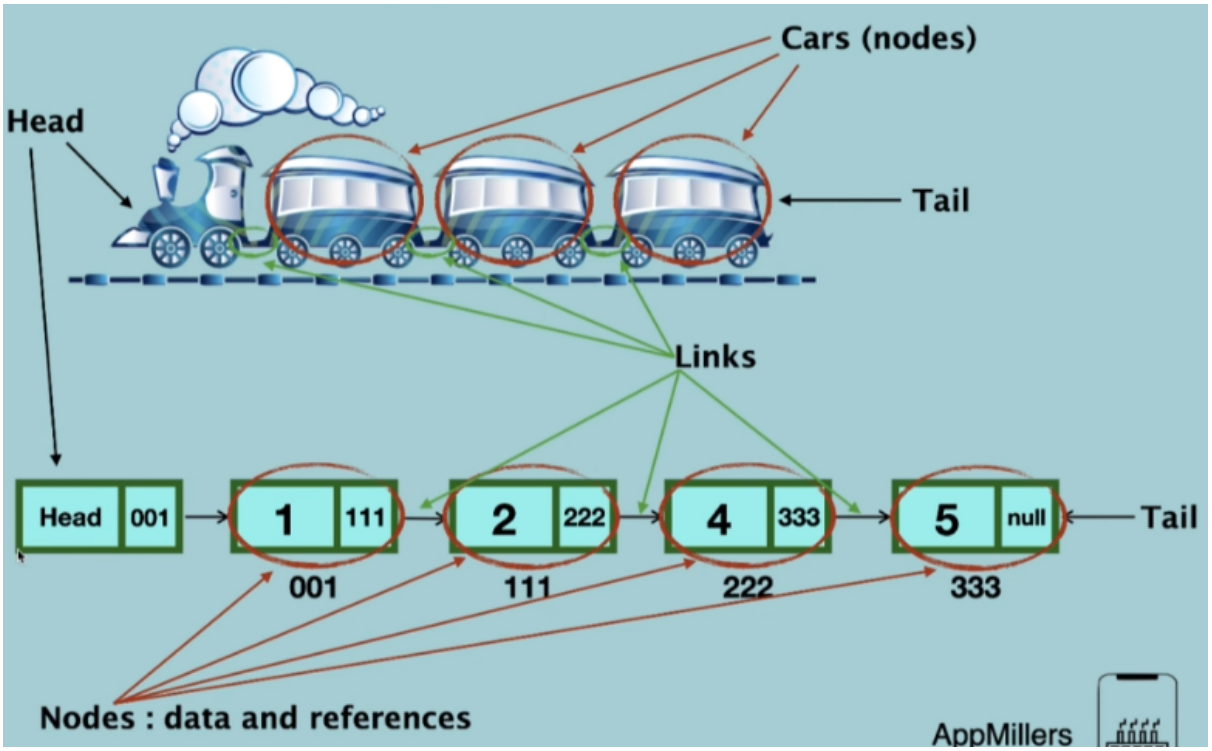
★ While the train is running, we Cannot jump from one Car to another one! We need to traverse all the way to our target Car(~same as LinkedList)

★ Cars consists of Passengers + Links to the next Car (~ Data + links)

★ Difference between Train vs. LinkedList: in Train the Cars state in pretty Contiguous way(they are nxt to

each other!) While in LinkedList the nodes are not necessarily nxt to each other!

★ Why do we need a head? If ya don't know where the List starts in the memory then how would ya access it?



★ Nodes: Values + References

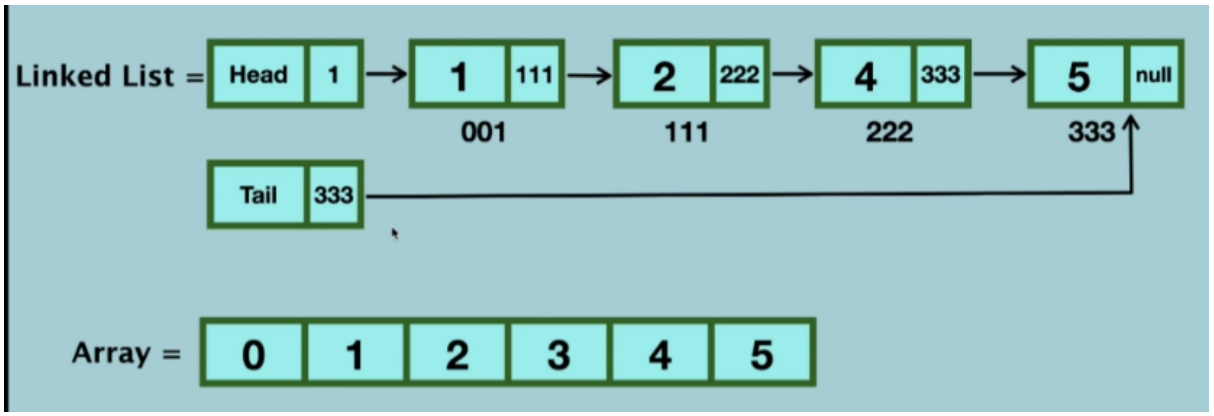
The value and a reference that connects the node to the next node! But what it is really ?

★ We know that when we allocate an element in the memory, it has a physical location in the ram! Basically the physical location address of the next node is stored in the current node to have a reference to it! For exp. Node 2, physical location is 111, so this location is stored in the first node to have a reference to it! So in each node we have the location of the next node! So when ya reach to the second node, we do know where the 3rd node is as we have its address stored in the node!

★ Why do we need a tail? Since without tail, if we wanna insert a node, we need to traverse through ALL nodes to get to the last one, but if we know where the tail is, then we can go straightforward to that point!

LinkedList vs. Array:

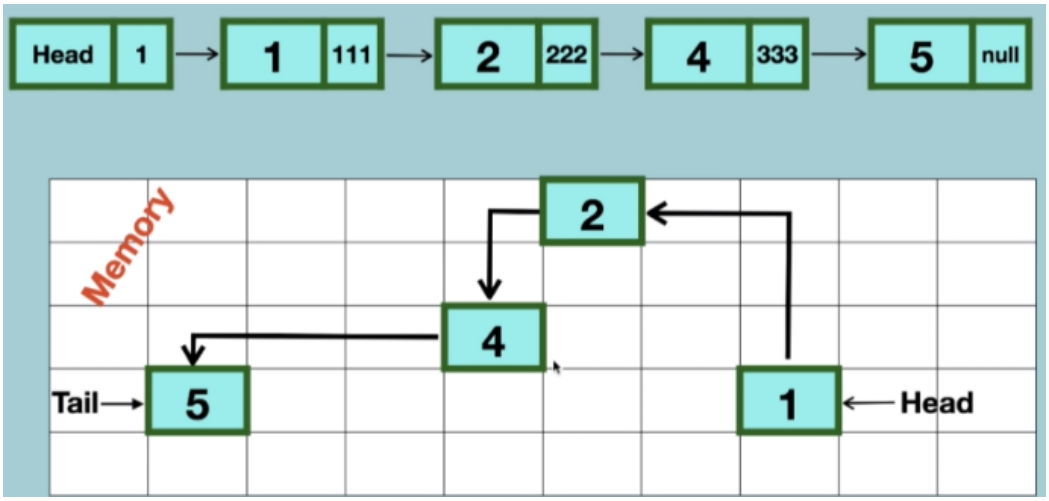
- Each element of the LinkedList is independent object ! Meaning, if we don't need any of the nodes in the LinkedList we can simply delete it and the LinkedList yet works! Whereas, in the Array we cannot do sucha thing, we can ofcourse delete the value of the cell but we CANNOT remove the Cell itself n it will exits in the memory and empty! (move the cells n the last empty)
- Variable Size: in the LL, we don't define the size when we create it! We always add/delete nodes, so there is no restriction! But in Array we should define the size and we cannot add any element to the same array if we reached the maximum size
- **Biggest Difference: insertion and removal;**
  When ya insert a value in the middle of an array, then ya need to update all the following indexes! Since they all need to go one step right! And if ya have a One-million array size then its so inefficient! But in LL if ya insert an element in middle/beginning ya only need to change the links between tem and we are not changing the place of the elements in the memory!



- **Random access:** Array pretting efficient arr[2] is all ya need to do, while in LL ya need to start from first node all the way to the target node!

Linked List in Memory:

- **Located Randomly NOT contiguously!**

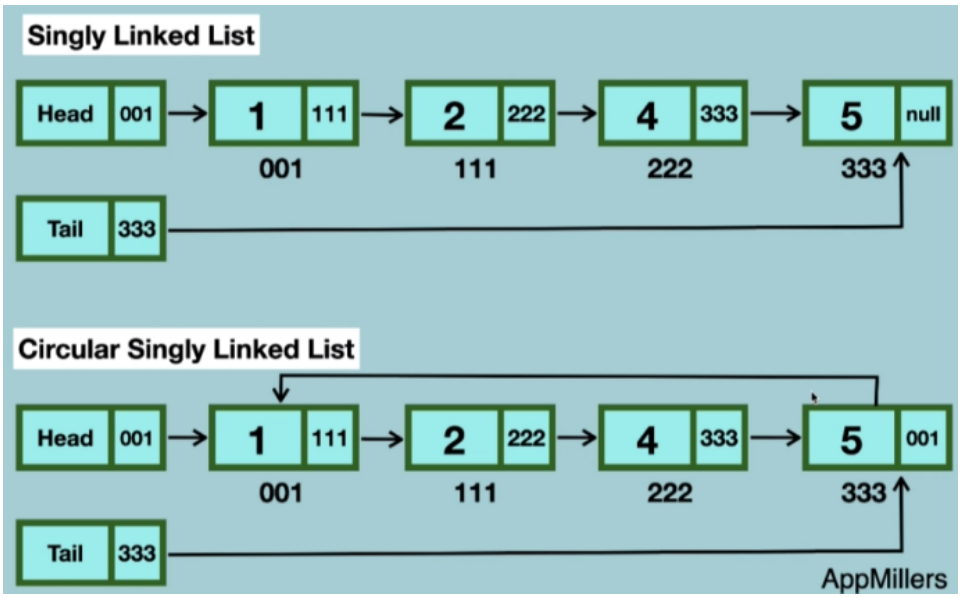


- The elements of a LL are not located contiguous, whereas they are located Randomly!! Allocation is randomly! It means that when we create a new node for LL, it will be created randomly in the memory and linked to available nodes! For exp. If ya consider the second node, ya can see the cell aside of it are empty so it means that the memory allocation is not contiguous! So in LL each node needs an extra space for the address pointer to link nodes! So the random allocation of the nodes allows us to add as many as nodes as we want bc the size of LL do not need to be defined at first! The nodes can be anywhere so we need to traverse through all nodes!

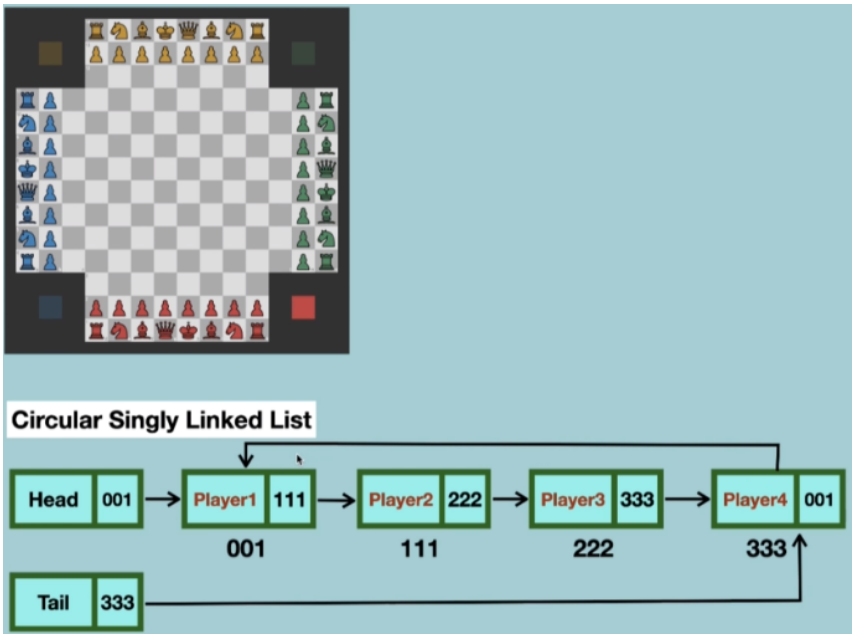
Different Type of LL:

1. **Singly Linked List**
2. **Circular Singly Linked List**
    - ★1. Has only a data and a reference to the next node
    - ★2. In the last node, we have the option of getting back to the first node! It means that the last

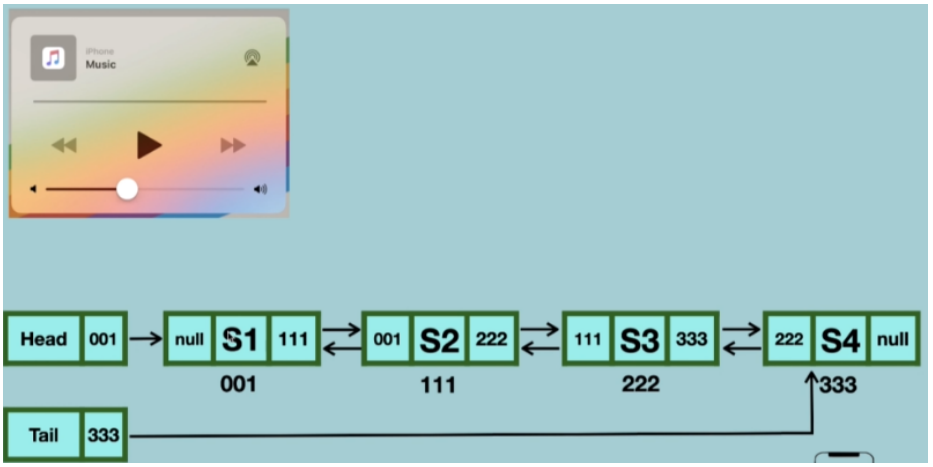
node has a reference to the first node! Any



clearance image?



3. **Doubly Linked List:** We have 2 references! One to the nxt node and one to the previous node! Which means that from each node we have access to both the next and previous nodes! We have the option of traversing back as we have the reference of the prev. Node!



4. Circular Doubly Linked List

The next reference of the last node in CDLL is, first node! So the physical location of the first node is 001 which is stored in the next node of the last node which means it has access to the first node! (mostly used when we look at the LL infinitely! While the list exists we can move forward and backward!)