MAKE SCHOOL

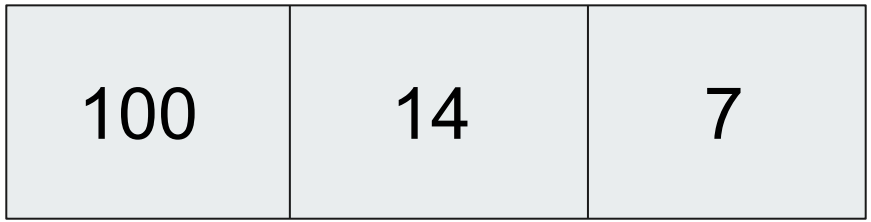
Linked Lists

CS 1.3 - Core Data Structures

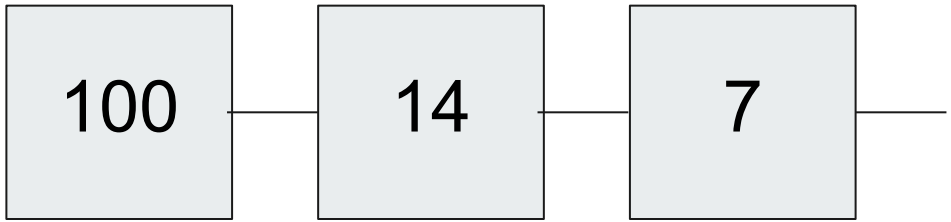
MAKE SCHOOL

Based: Linked Lists

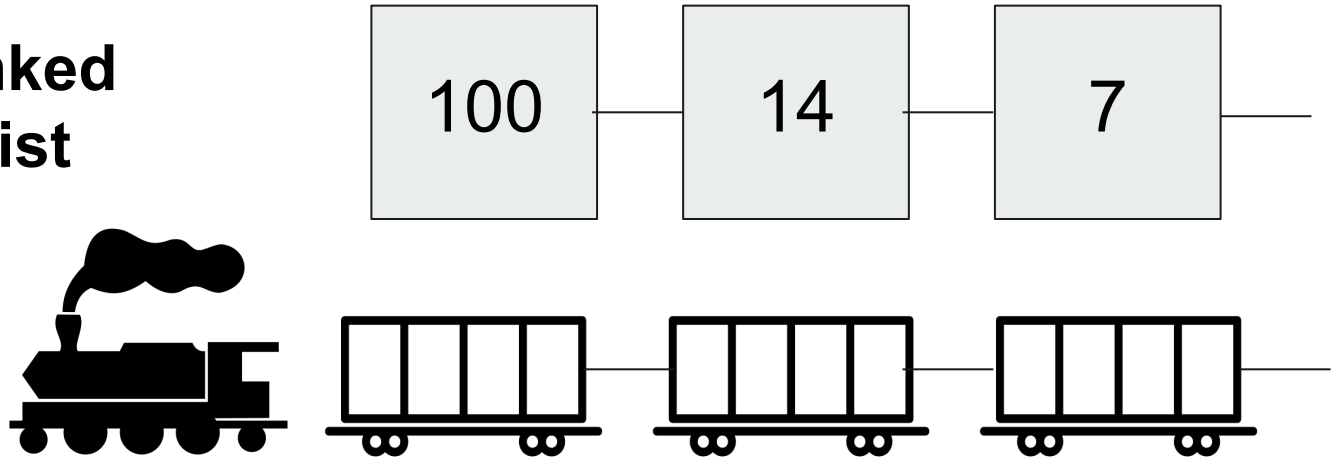
Array



Linked List

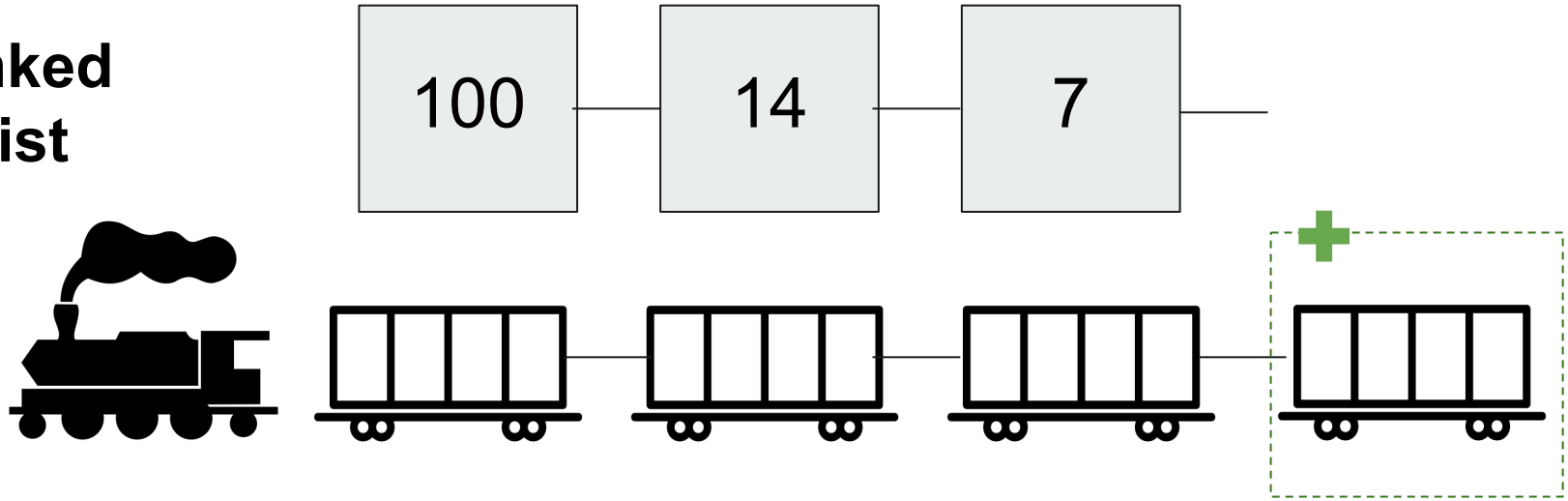


Linked List



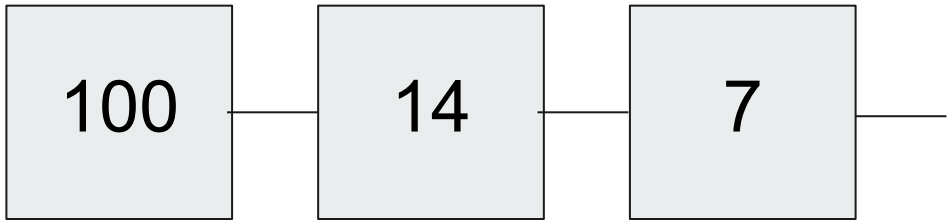
Analogy: A linked list is similar to a train.

Linked List



Analogy: To increase the train load, we do not need to switch bigger train. Just add train cars.

Linked List



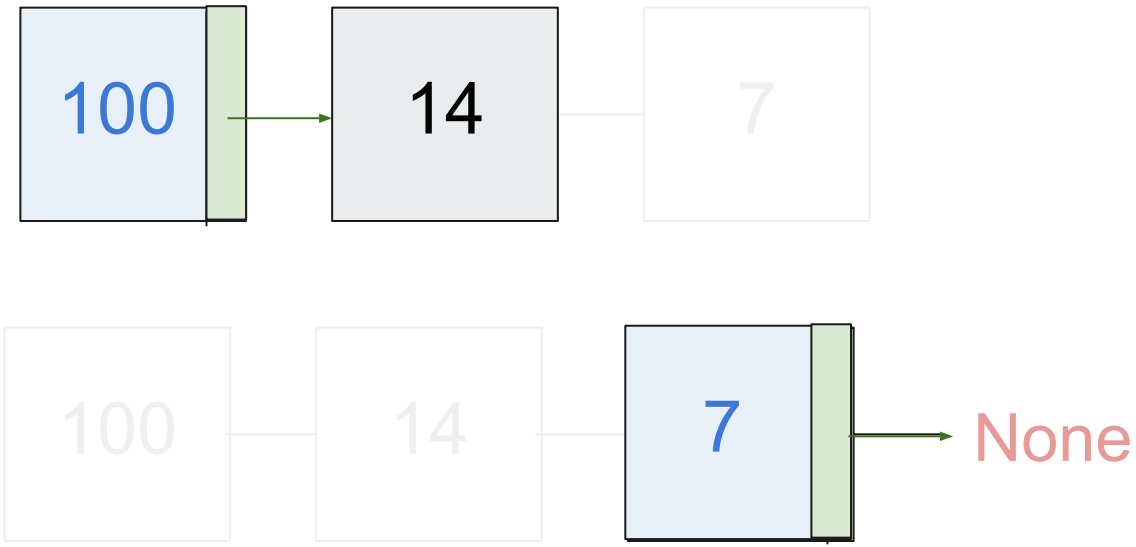
A linked list consists of nodes.

Linked List

Each **node** contains a two pieces of information:

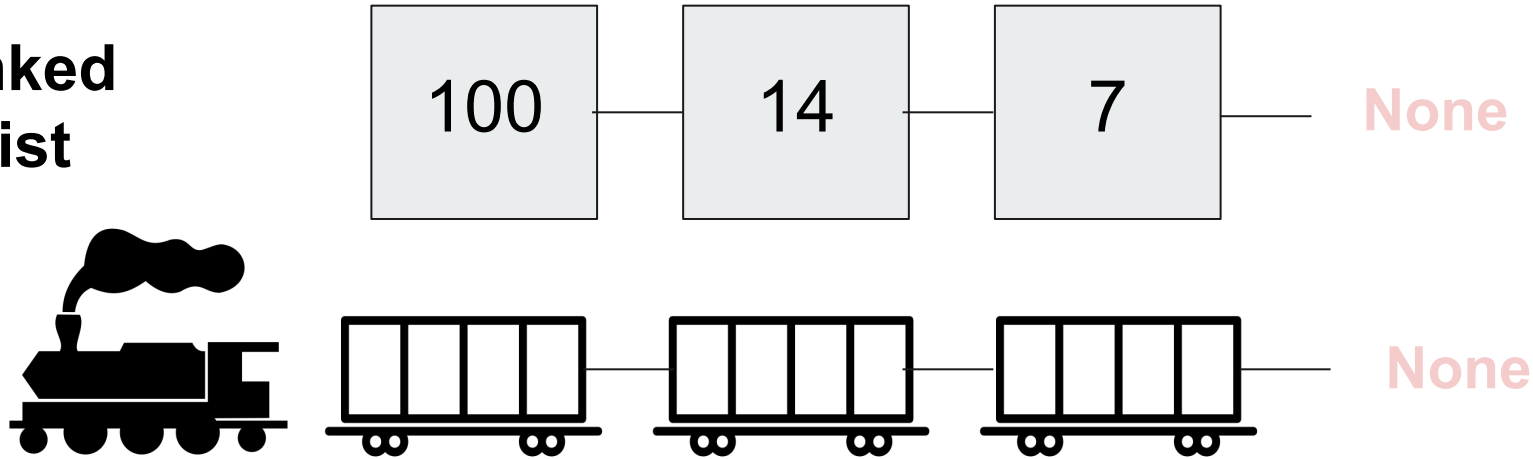
1. **Data**
2. **Next pointer**

Linked List



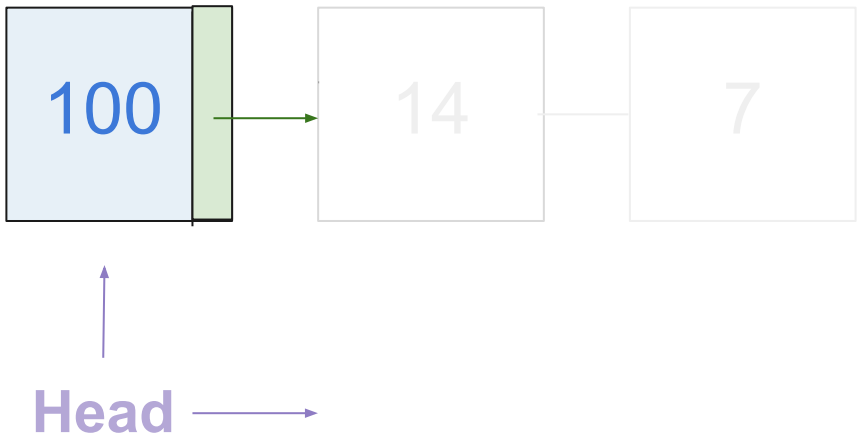
A **next pointer** either points to another node or None, if it is at the end of the linked list.

Linked List



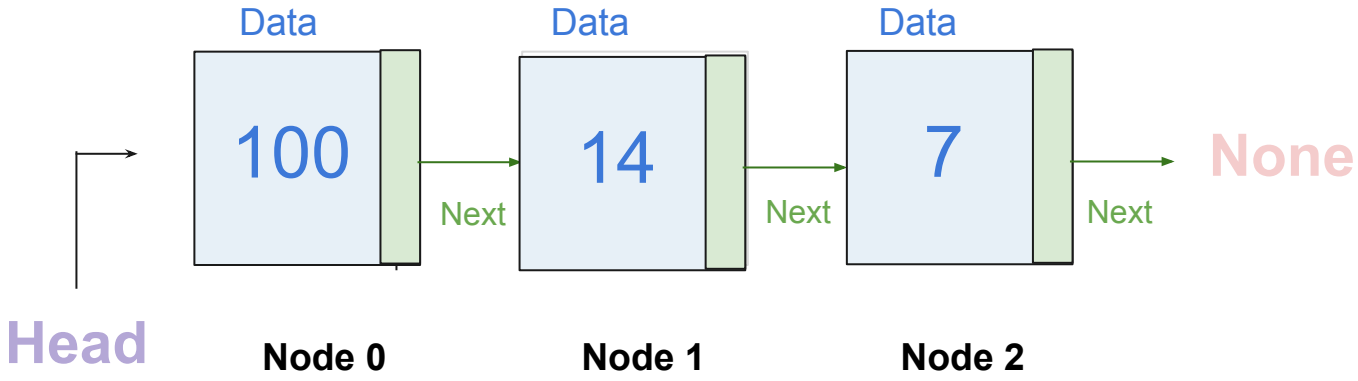
Analogy: After the last train car, there is not another train car. Thus, we are at the end of the train.

Linked List

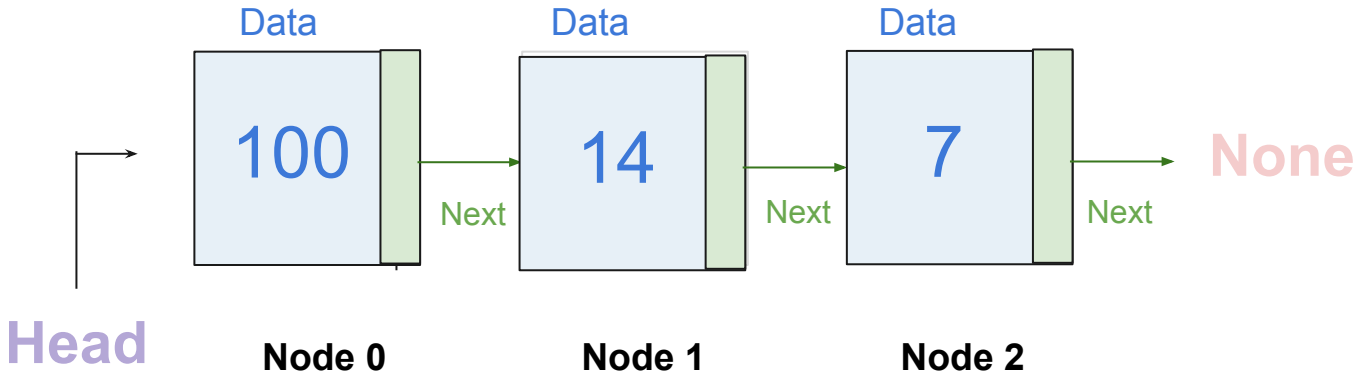


Linked List have a **head pointer** that keeps track of the head (beginning) of the list.

Linked List

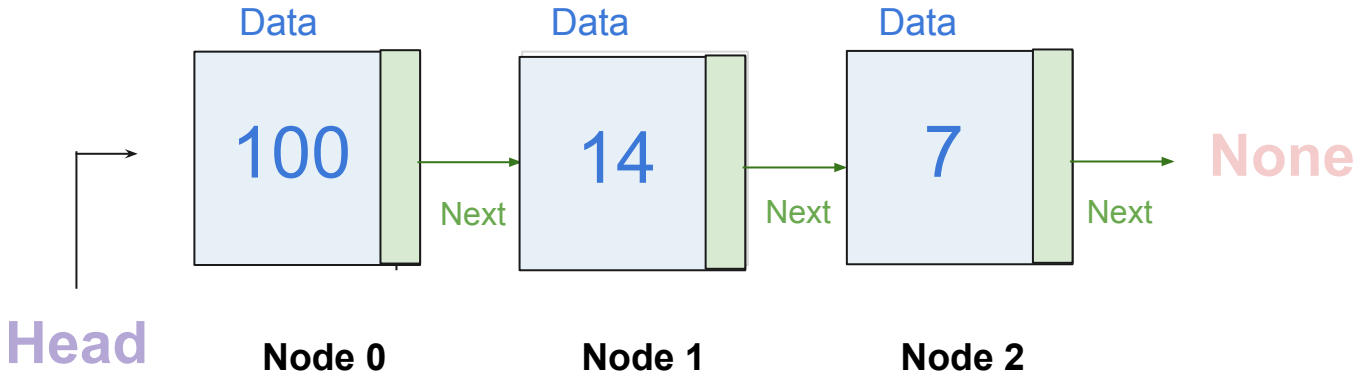

Data
Data
Data
100
14
7
Next
Next
Next
None
Head
Node 0
Node 1
Node 2

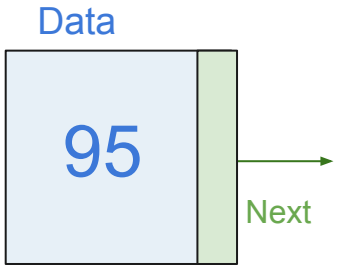
Linked List



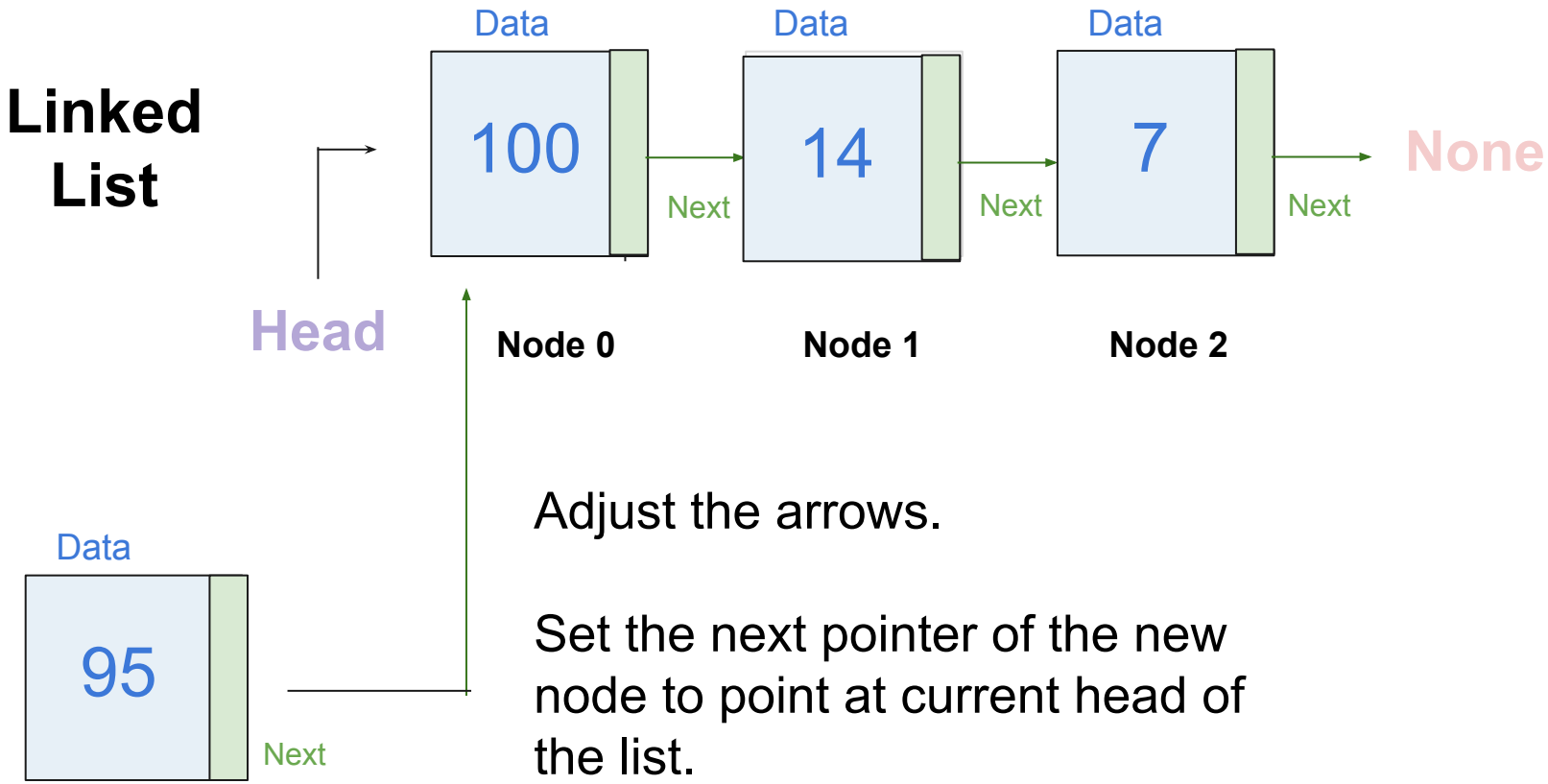
Given the above Linked List, insert **95** at the beginning of the list.

Linked List

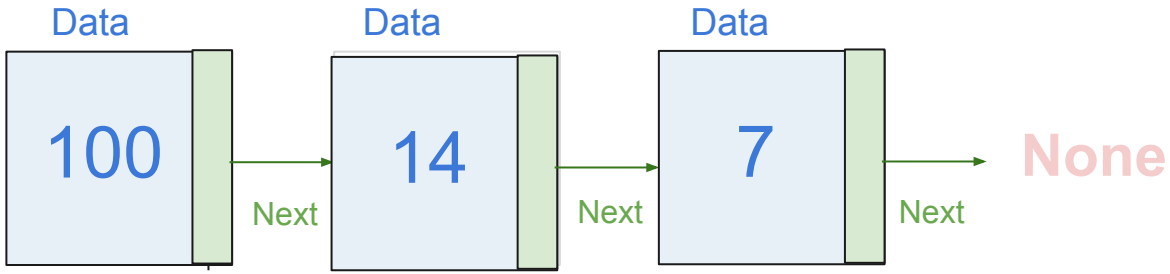


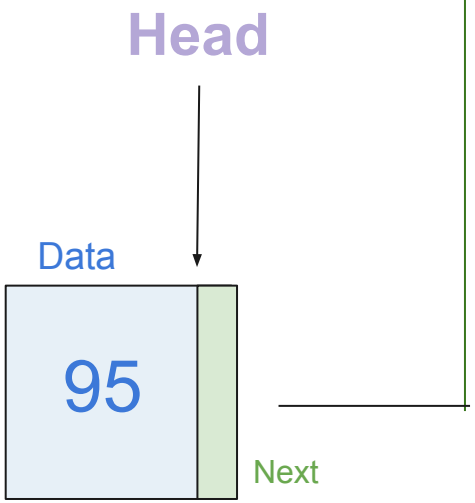


First, create a new node to store 95.


Linked List
Data
100
Next
Data
14
Next
Data
7
Next
None
Head
Node 0
Node 1
Node 2
Data
95
Next
Adjust the arrows.
Set the next pointer of the new node to point at current head of the list.

Linked List





Adjust the arrows.

After, point the head pointer to point at the node storing 95.

The Linked List Abstract Data Type

MAKE SCHOOL

Structure: An empty Linked List. It only should have a head pointer.

Operations:

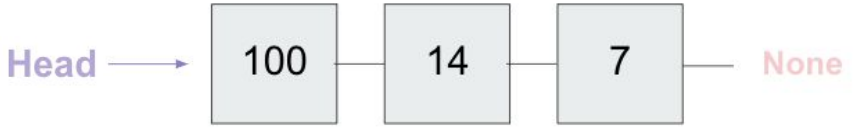
- `add(item)` adds a new item at beginning to the list.
- `size()` returns the number of items in the list.
- `search(item)` searches for the item in the list and returns a boolean value.
- `remove(item)` removes the item from the list.
- `isEmpty()` tests to see whether the list is empty and returns a boolean value.

Other Operations:

- `pop()` removes and returns the last item in the list.
- `pop(index)` removes and returns the item at index.
- `append(item)` adds a new item to the end of the list making.
- `index(item)` returns the index of item in the list.
- `insert(index,item)` adds a new item to the list at index. Assume the list and there are enough existing items to have index.

Check for Understanding

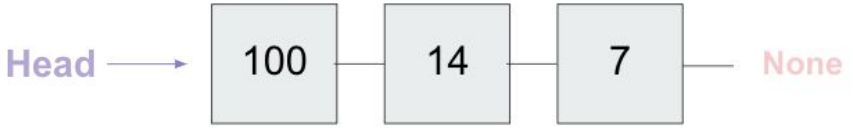
add(item) adds the item to the beginning of the list. Draw how the add(10) would work:



Students, draw anywhere on this slide!

Check for Understanding

remove(item) removes the item from the list. Draw how the remove(14) method would work:



Students, draw anywhere on this slide!

Linked List Class Diagram

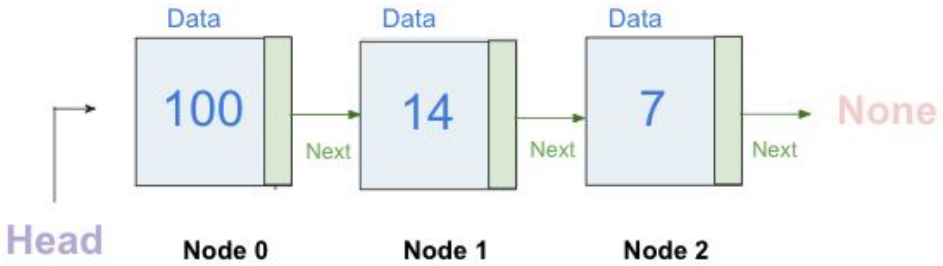
MAKE SCHOOL

Linked List
Attributes: - head: Node()
Methods: + add(item) + size() + search(item) + remove(item) + isEmpty()

Linked List are comprised of nodes.

Composition!

Node
Attributes: - data - next
Methods: + get_data() + get_next() + set_data() + set_next()



MAKE SCHOOL

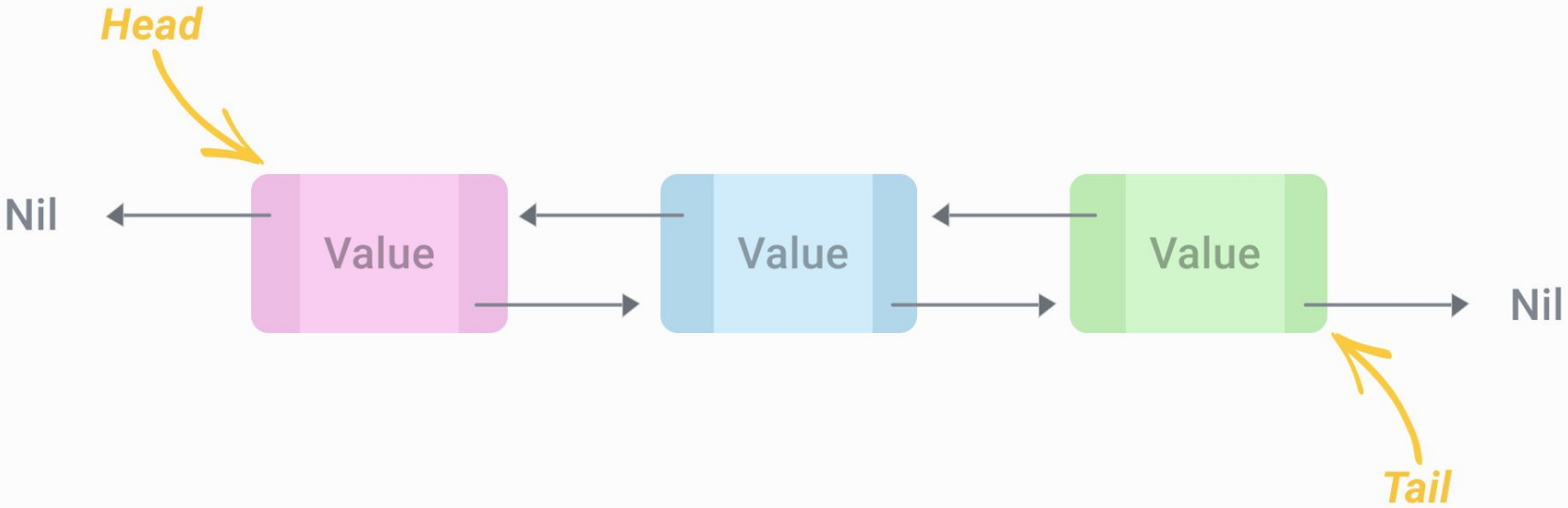
Let’s review how to implement a LinkedList!

Students browse: repl.it/@CoreDS/SinglyLinkedList?lite=true

MAKE SCHOOL

Mastery: Doubly Linked List

What is a Doubly Linked List

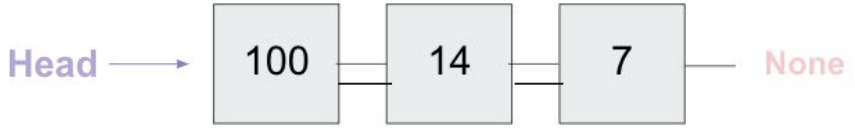


MAKE SCHOOL

Adding a previous pointer changes how some methods are implemented!

Remove for a doubly linked list

remove(item) removes the item from the list. Draw how the remove(14) method would work for a doubly linked list:



Students, draw anywhere on this slide!

Time complexity analysis

What is the time complexity of remove for a singly linked list, what about a doubly linked list?

Students, draw anywhere on this slide!

MAKE SCHOOL

Module 2: Linked Lists