APS106

UNIVERSITY OF TORONTO

Advanced Data Structures: Linked Lists

Week 11 | Lecture 2 (11.2)

if nothing else, write #cleancode

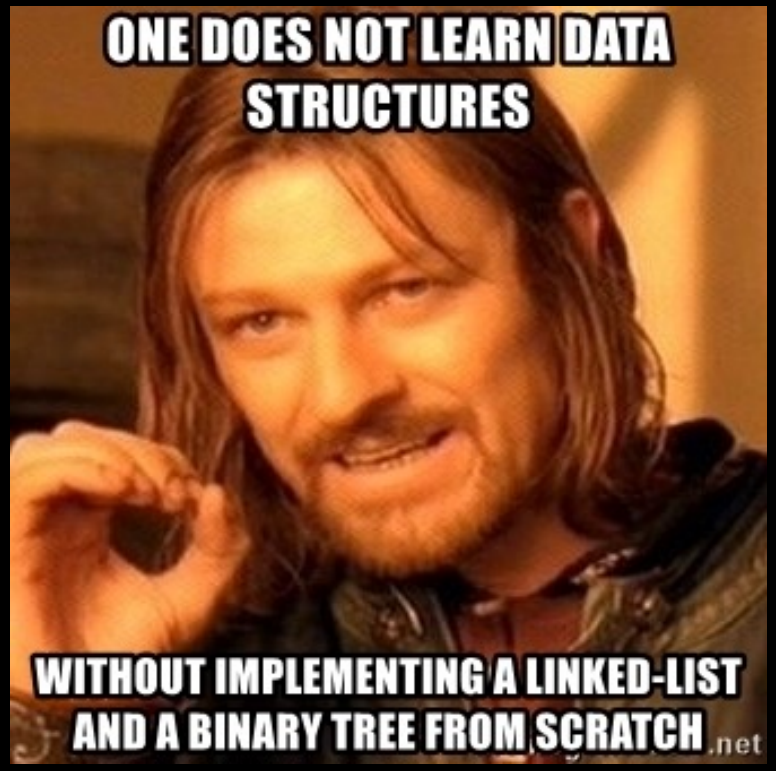
This Week's Content

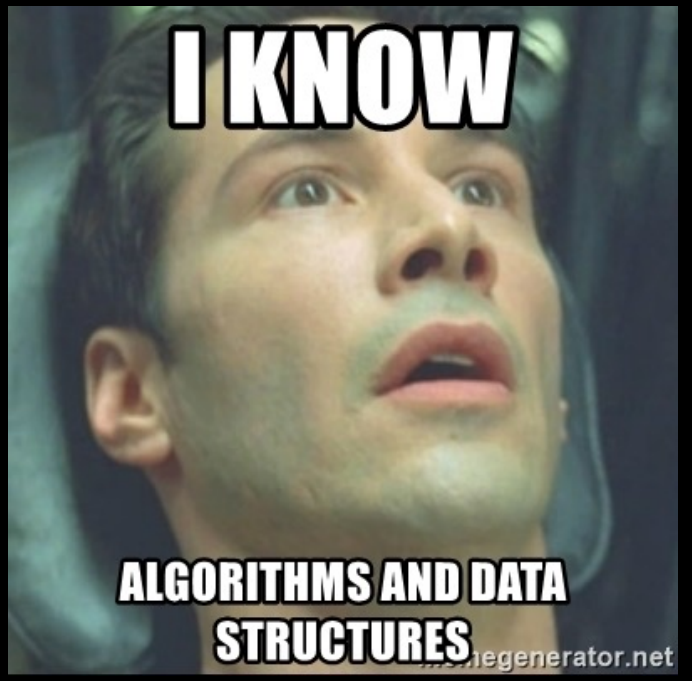
- **Lecture 11.1**
  - More OOP! Encapsulation and Examples
- **Lecture 11.2**
  - **Advanced Data Structures: Linked Lists**
- **Lecture 11.3**
  - Design Problem! Gaussian Elimination

Data Structures!

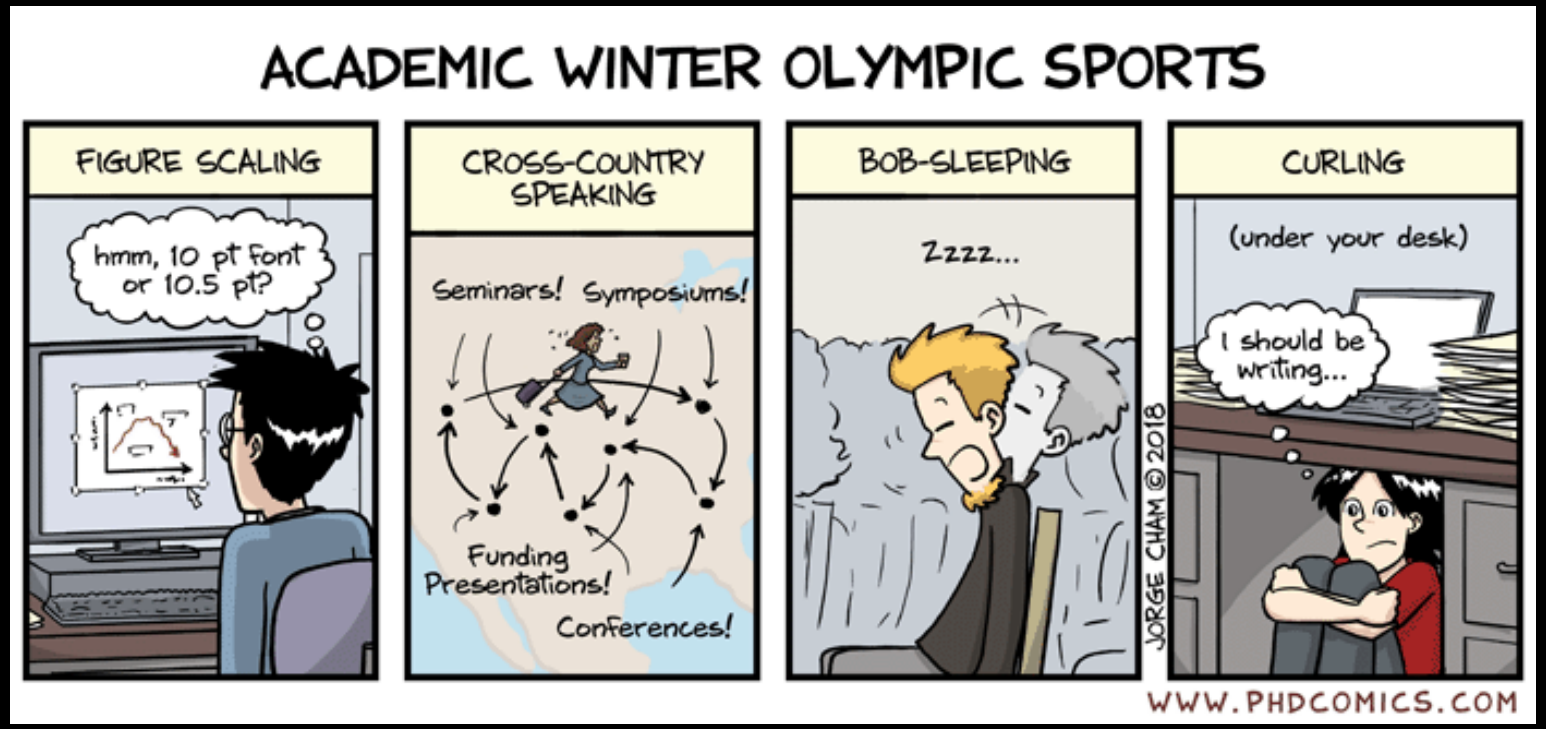
Data structures are "containers" that organize and group data

- Lists
- Sets
- Tuples
- Dictionaries
- Linked lists
- Binary trees

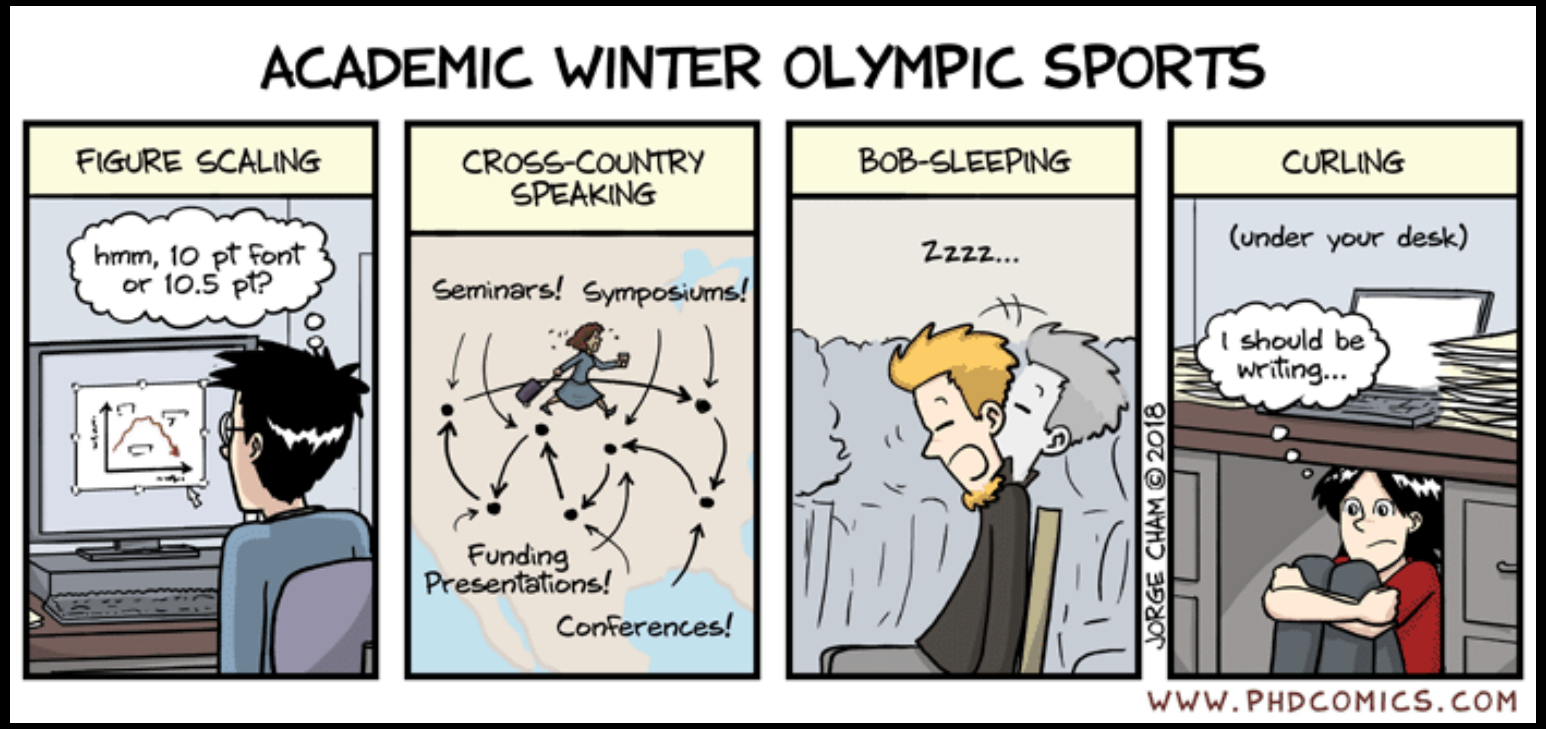




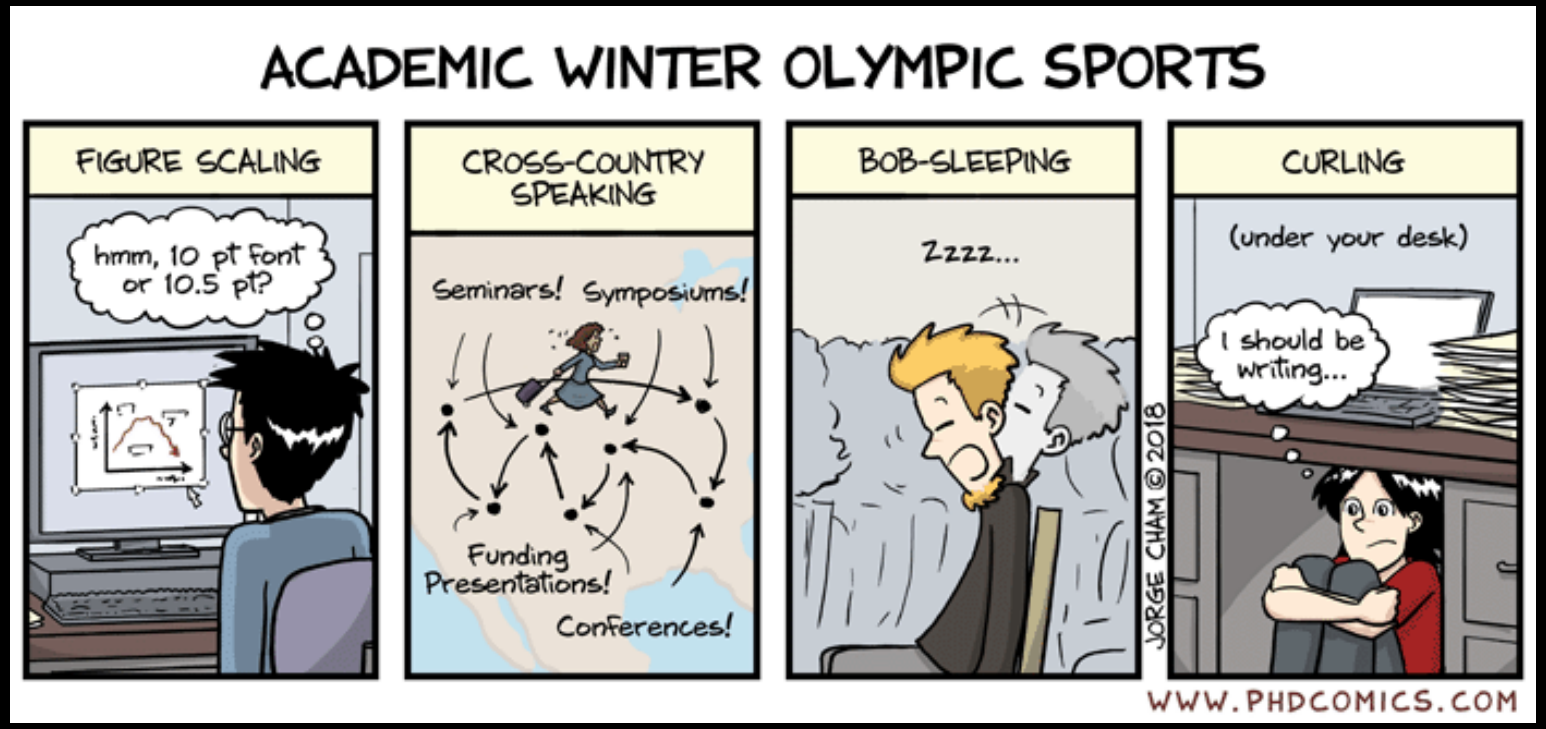
Linked Lists – Sequences of Things to Do



Linked Lists – How do we read comics?



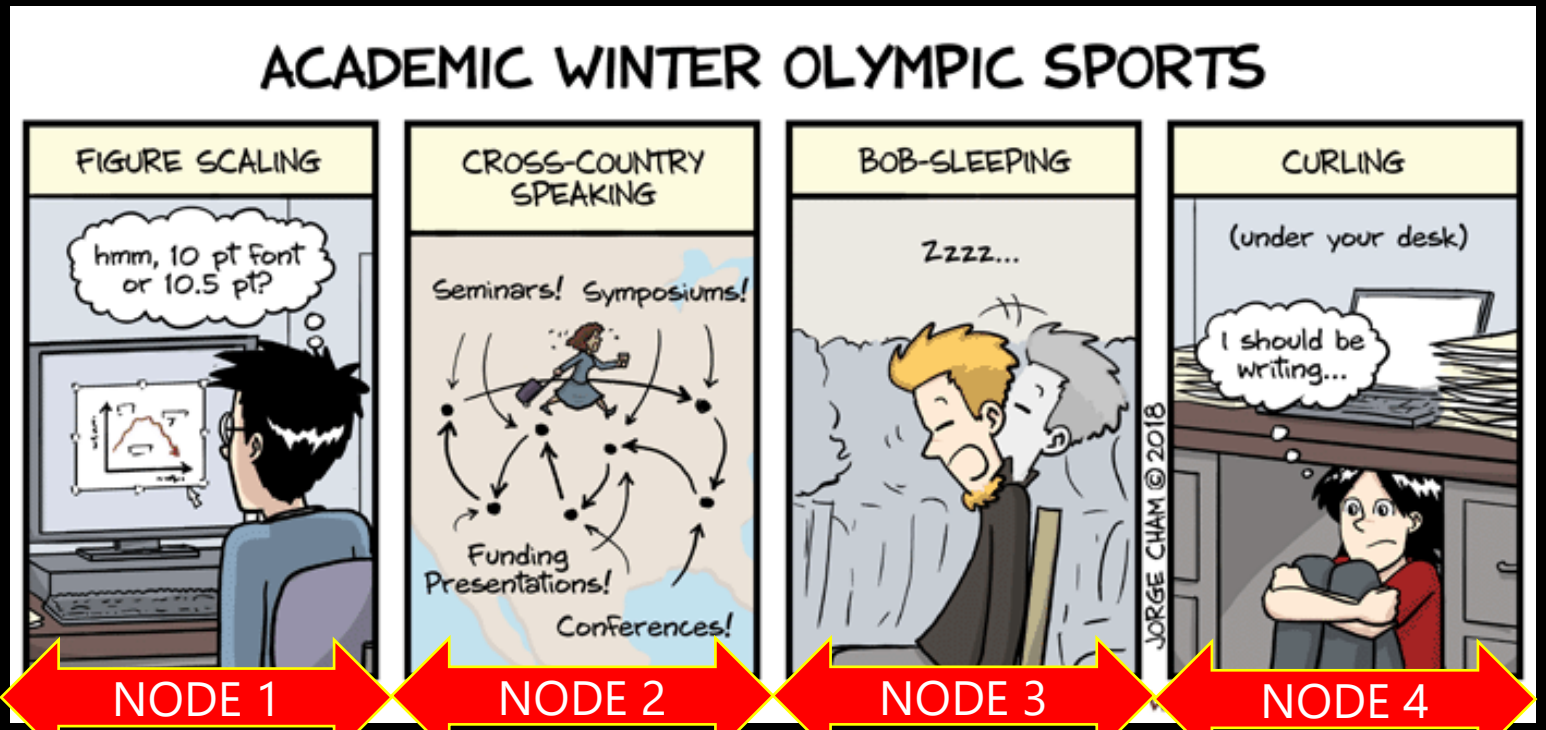
Linked Lists – How do we read comics?



1	2	3	4

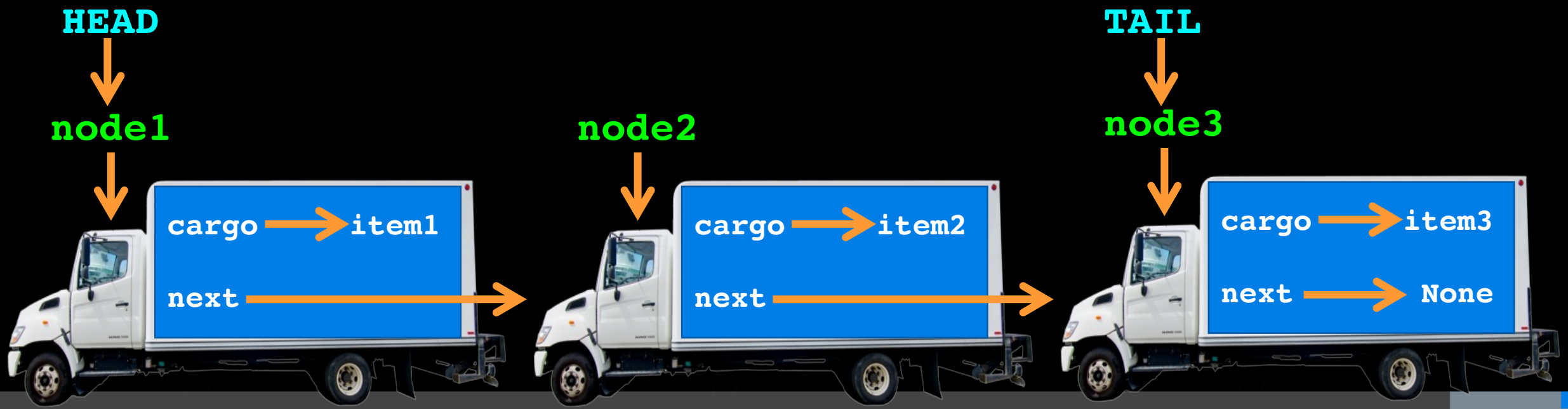
Linear	Collection	Data	Elements
Linked		List	

Linked Lists – How do we read comics?

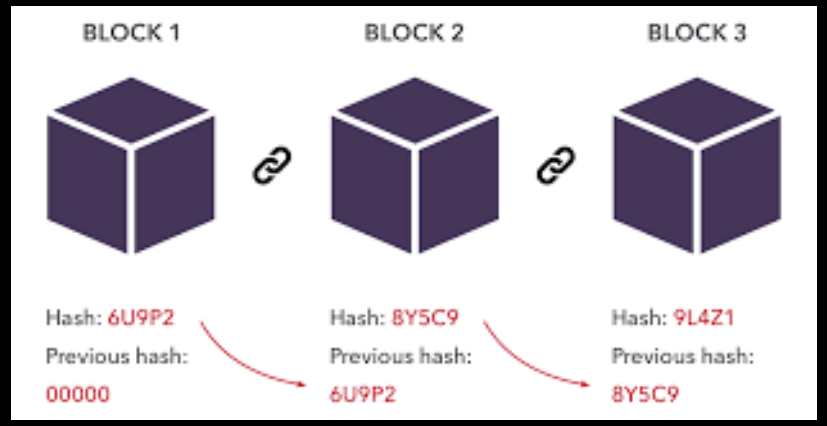


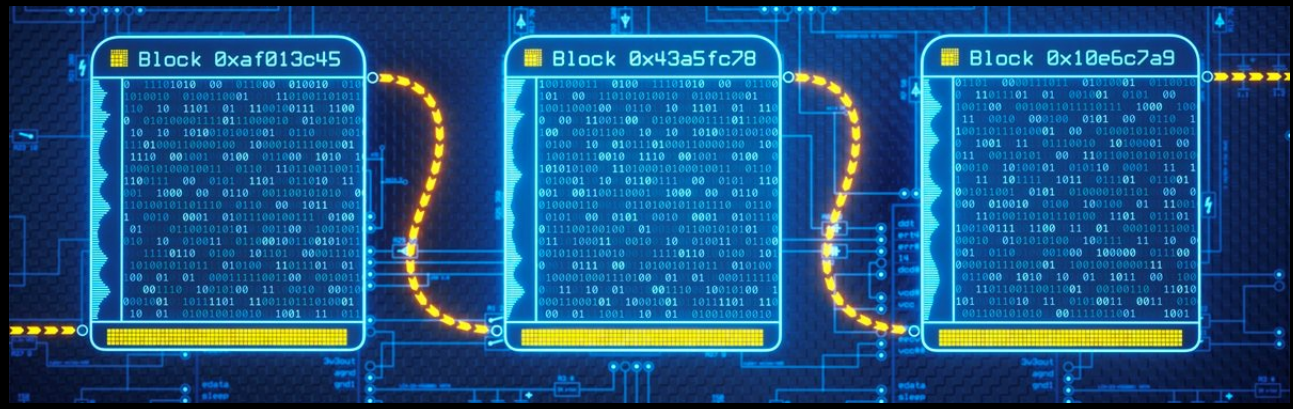
Linked Lists!

- Linked lists are a linear collection of data elements made up of nodes
- A node contains a link to the next node in the list, and some unit of data (i.e. `str`, `int`, `list`, `set`, etc.) that we will call the cargo
- The last node in a linked list is `None` and does not provide a link to any other nodes
- The beginning of the LL is called the "head" and the end is the "tail"



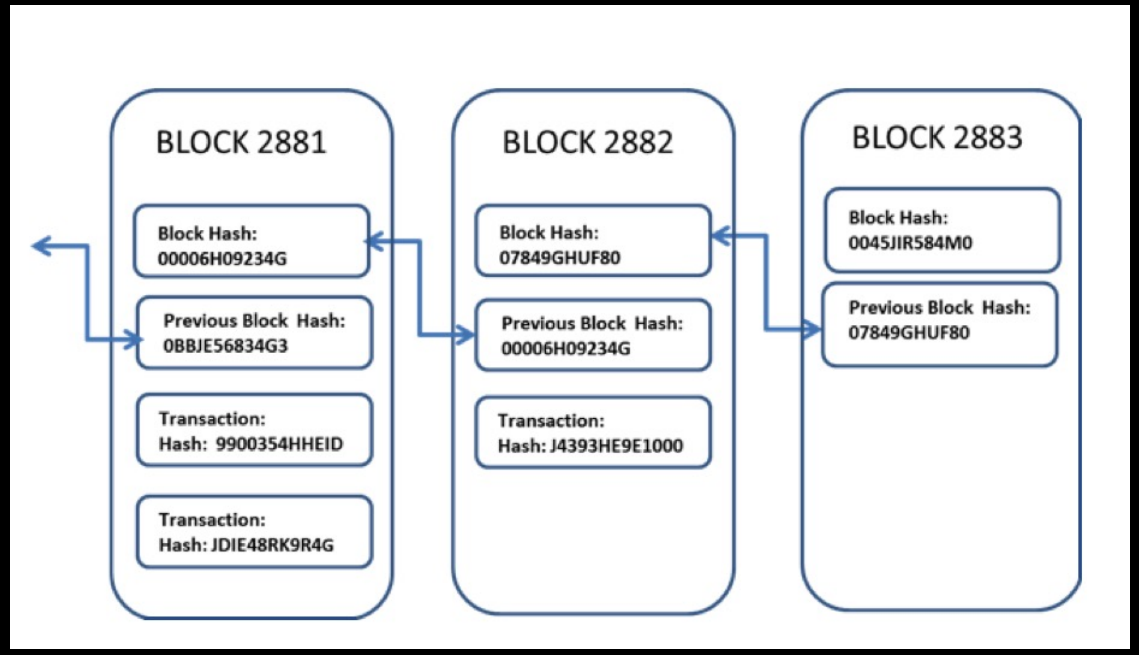
Sound familiar to anyone?





What if...

- Cargo = transaction data?
- Node = block?
- Link = chain?



Linked List Advantages

- Advantages
  - Can dynamically shrink or grow at run-time
    - Other programming languages require to define the length of array upon creation, leading to wasted space unless it's full
  - Faster insertion and deletion
    - No need to shift every element afterwards
  - Efficient memory management
    - Does not need to store elements sequentially (or contiguously) in memory
  - Implementation of data structures
    - Helpful for representing queues and stacks

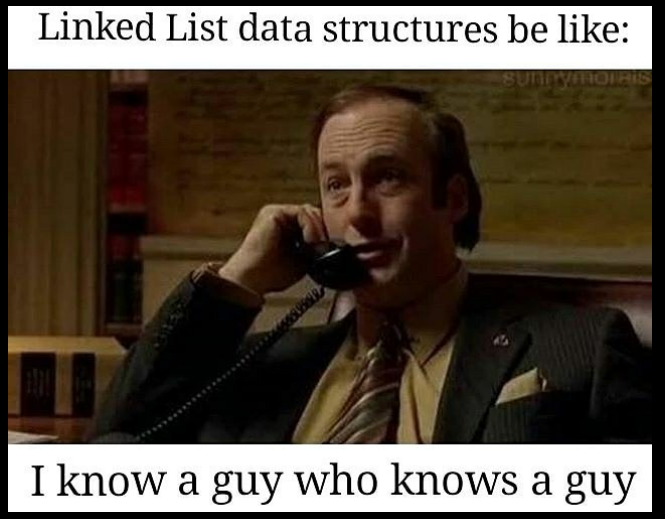
Traffic jam entering Toronto

Civil engineers model this!

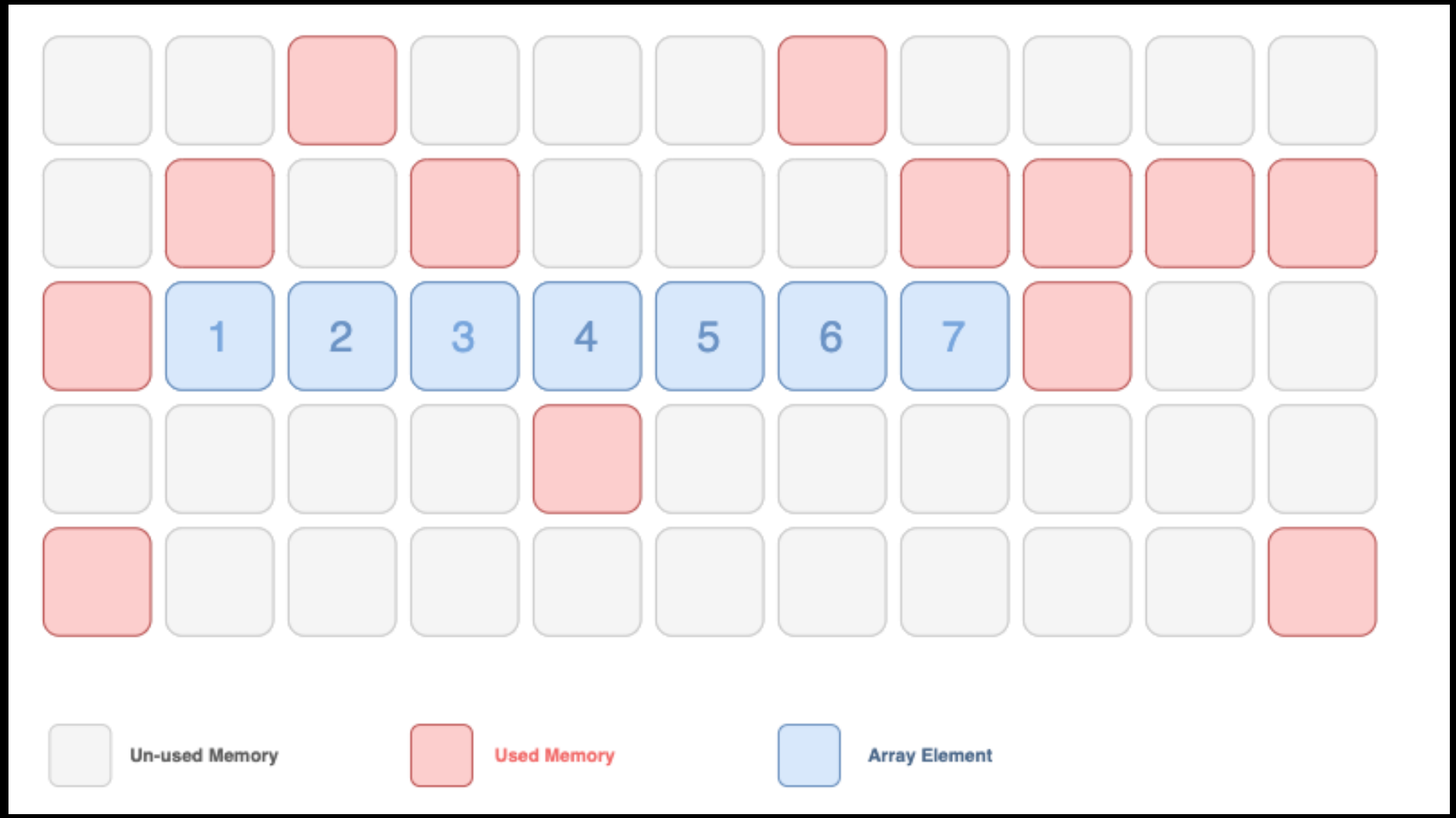
A line (or queue) waiting for COVID rapid tests in Union Station

Linked List Disadvantages

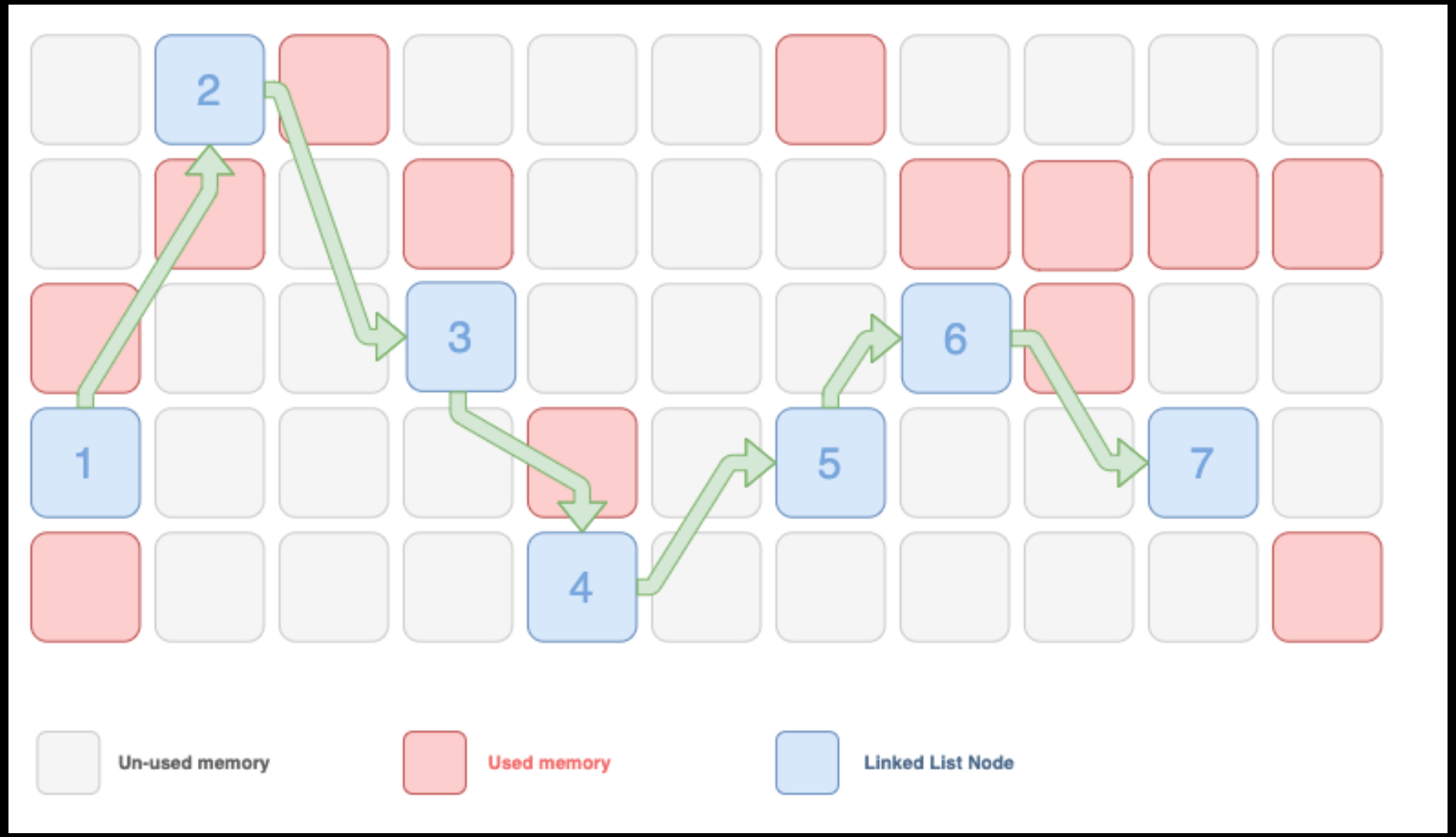
- Disadvantages
  - More memory used for each element
    - Must store its own cargo AND the pointer to next node
  - Random access
    - Because storage is not contiguous, you must traverse through all nodes to access content at node X, whereas you can directly index a list/array with list[X]
  - No easy way for reverse traversal
    - A "doubly-linked list" solves this by having a pointer to both previous and next nodes, but consumes more memory



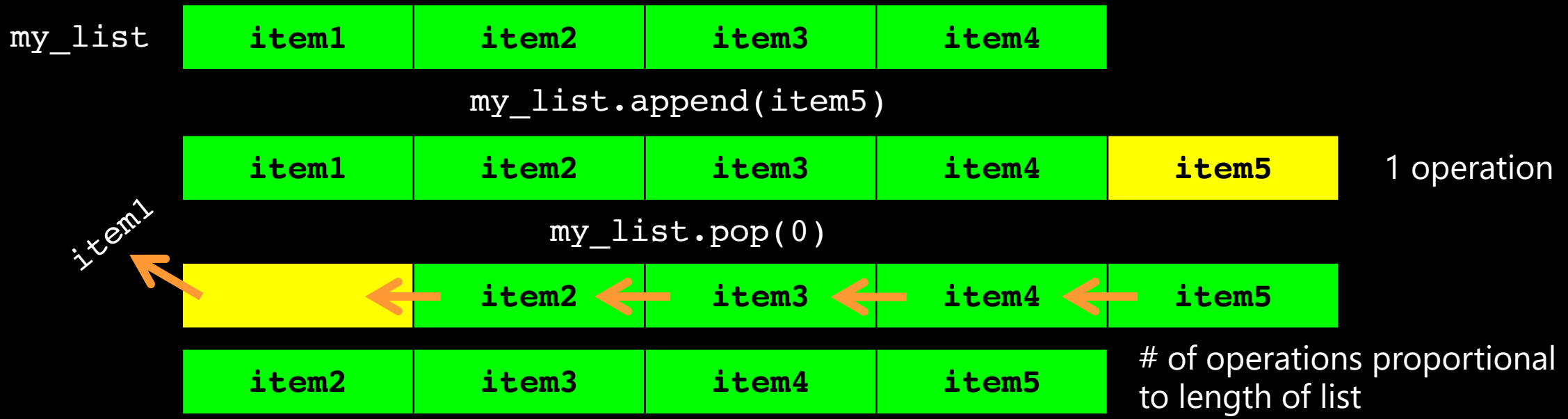
Contiguous (beside each other) Storage



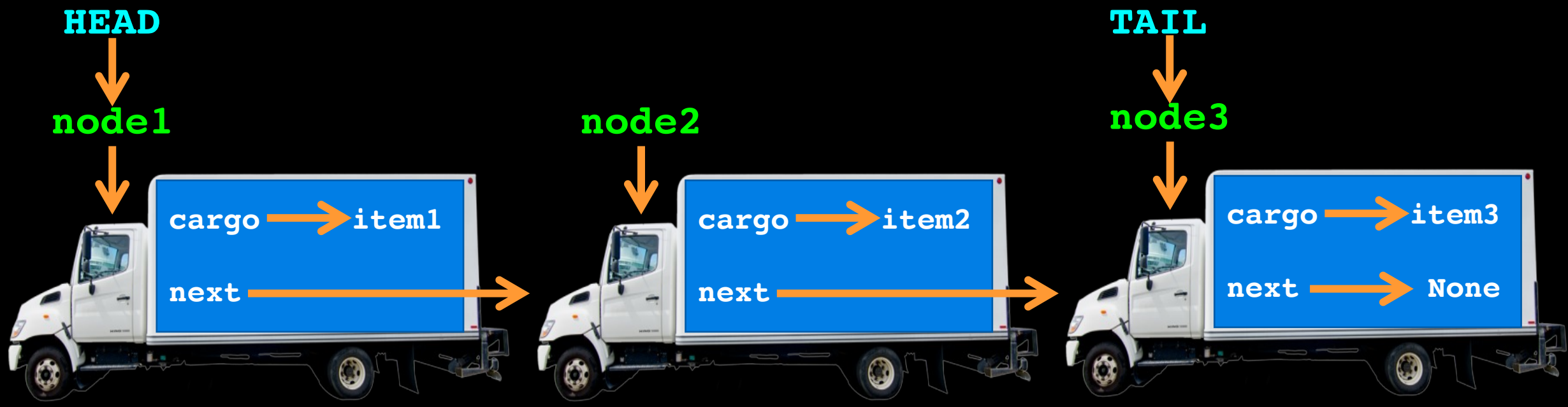
Non-contiguous Storage



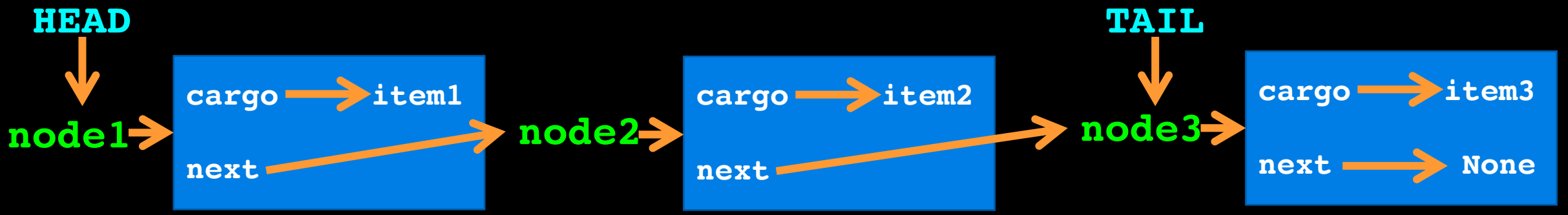
Why Linked Lists? Modeling a Queue



Why Linked Lists? Modeling a Queue

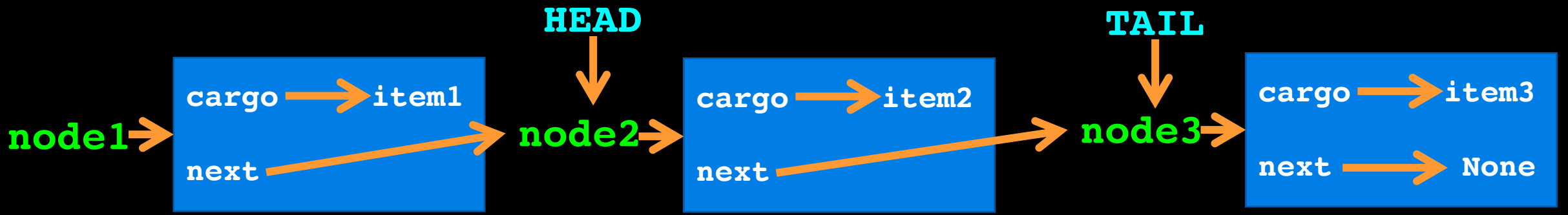


Why Linked Lists? Modeling a Queue

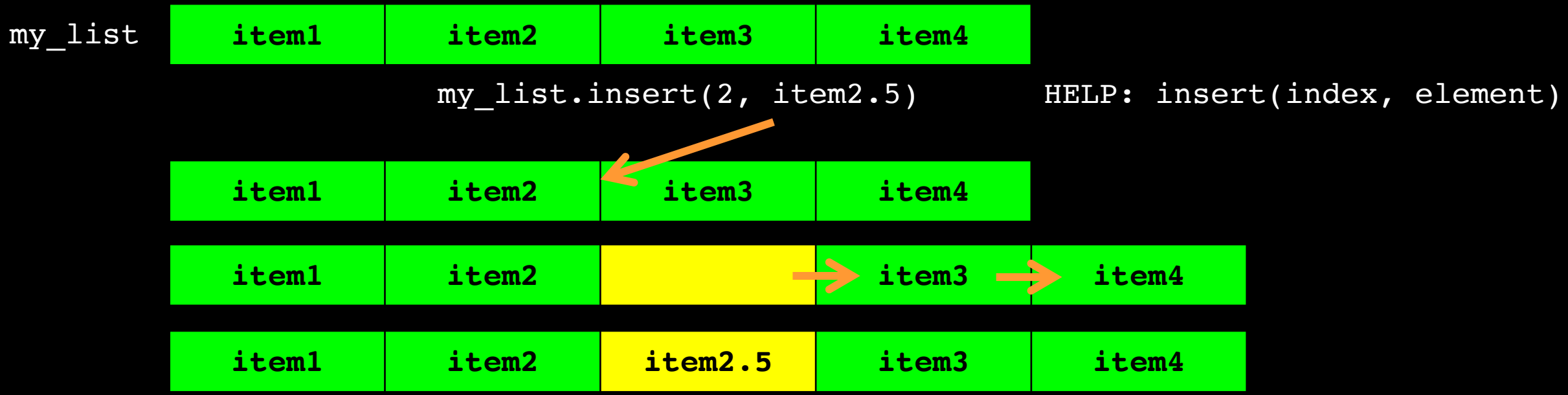


Remove a node from the beginning

1 operation

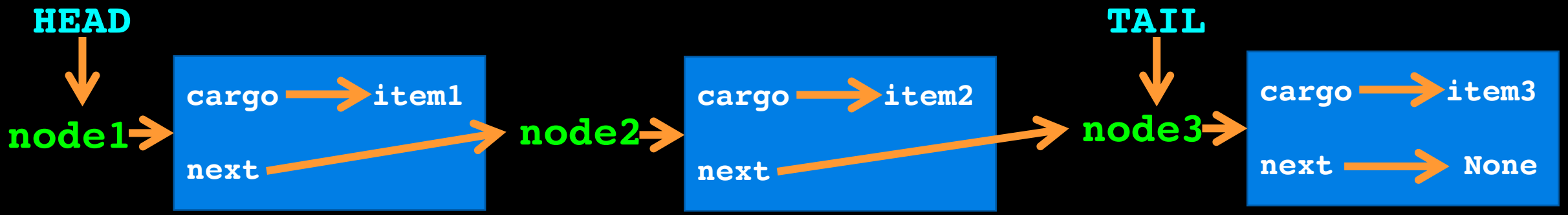


Why Linked Lists? Efficient Insertion

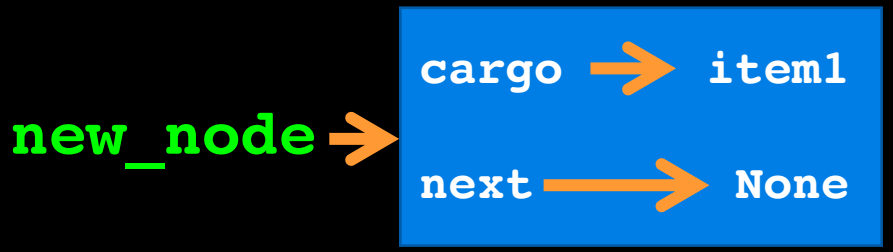


# of operations proportional to length of list

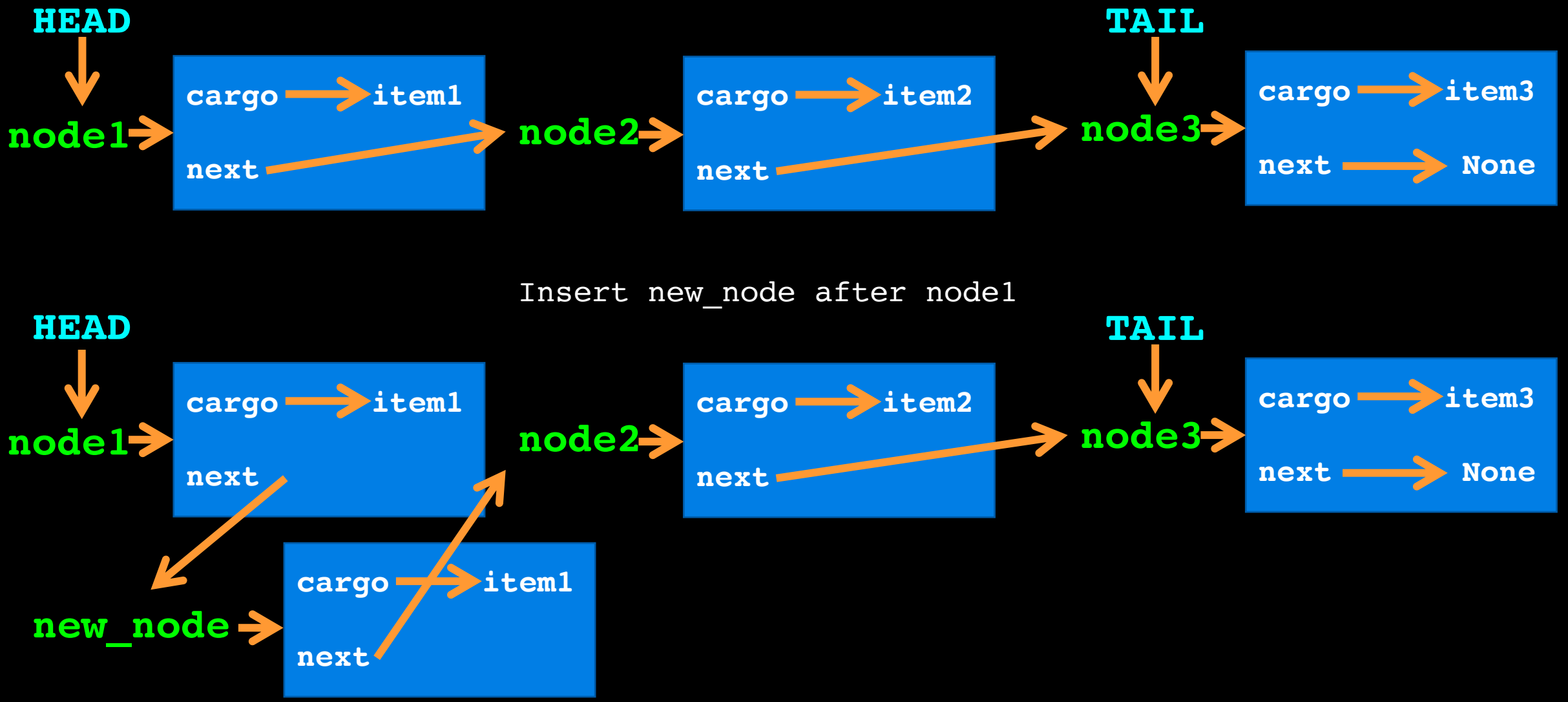
Why Linked Lists? Efficient Insertion



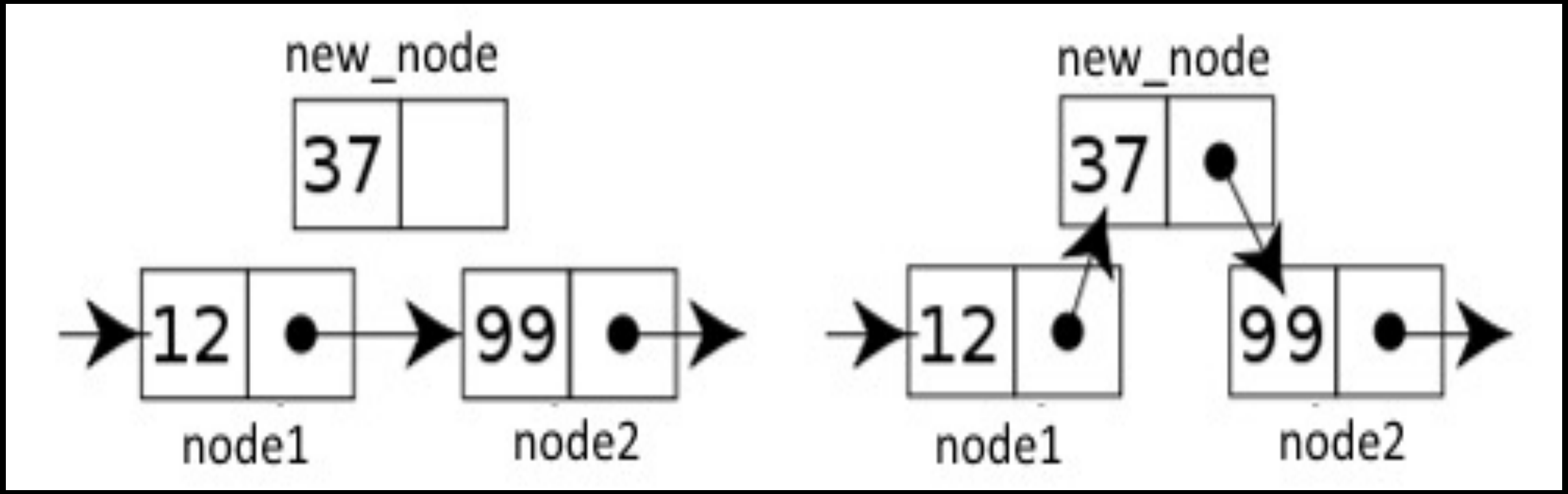
Insert new_node after node1



Why Linked Lists? Efficient Insertion

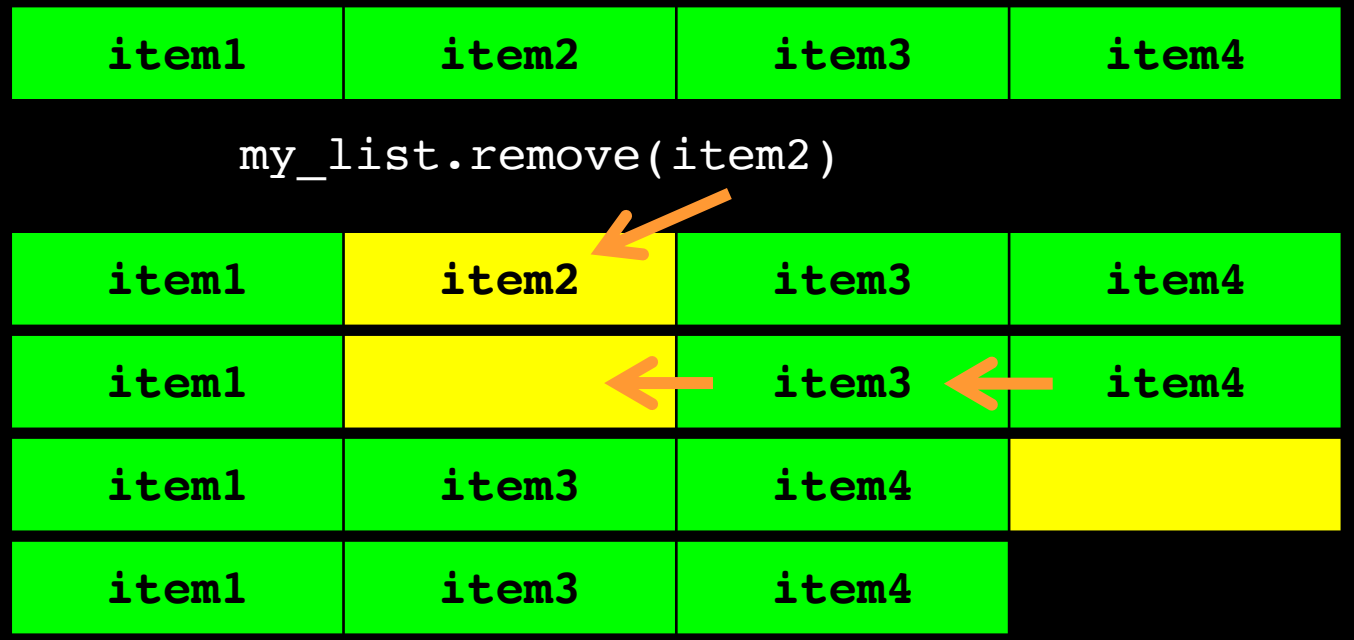


Efficient Insertion



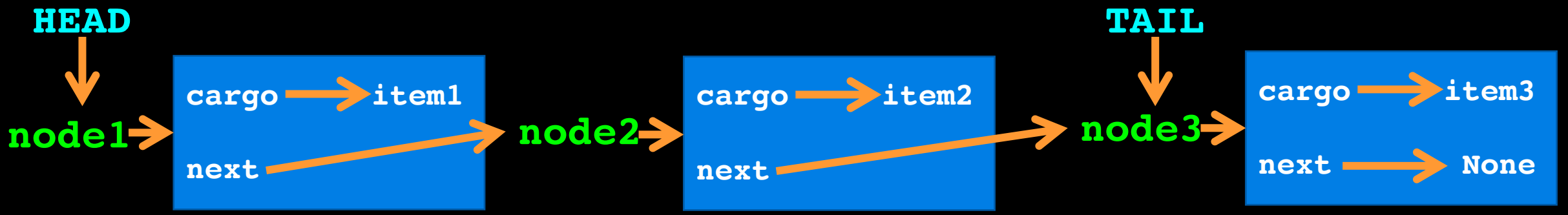
Why Linked Lists? Efficient Deletion

my_list



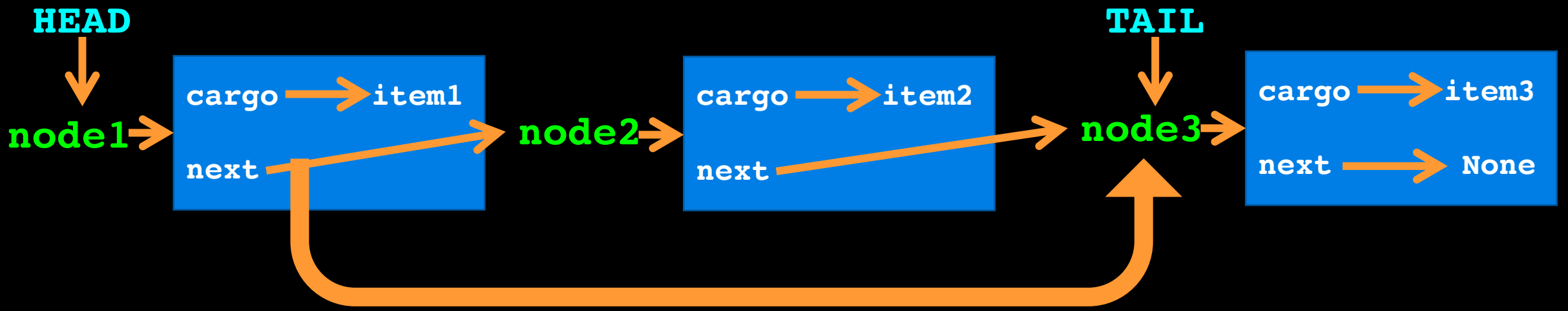
# of operations proportional to length of list

Why Linked Lists? Efficient Deletion



Delete node2

Why Linked Lists? Efficient Deletion



Delete node2

The Node Class

- Let's use our knowledge of classes to prepare a linked list data structure in Python.
- First, we need to make a node class.
  - We need __init__ and __str__ methods so we can create and display our new type

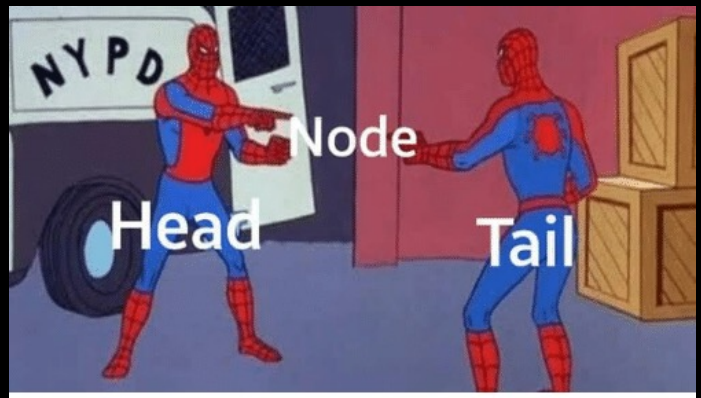
```
class Node:
    def __init__(self, cargo=None, next=None):
        self.cargo = cargo
        self.next  = next

    def __str__(self):
        return str(self.cargo)
```

The Node Class

- The string representation of a node is just the string representation of the cargo
- To test the implementation so far, we can create a node a print it:

```
>>> node = Node("test")
>>> print(node)
test
```

- BUT, a linked list needs more than one node!



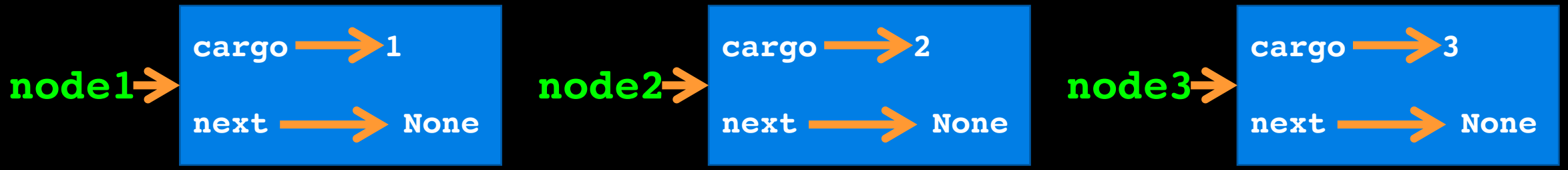
When you have a Linked List with just one element...

The Node Class

- To make a linked list, we need more than one node:

```
>>> node1 = Node(1)
>>> node2 = Node(2)
>>> node3 = Node(3)
```

- What's wrong here?



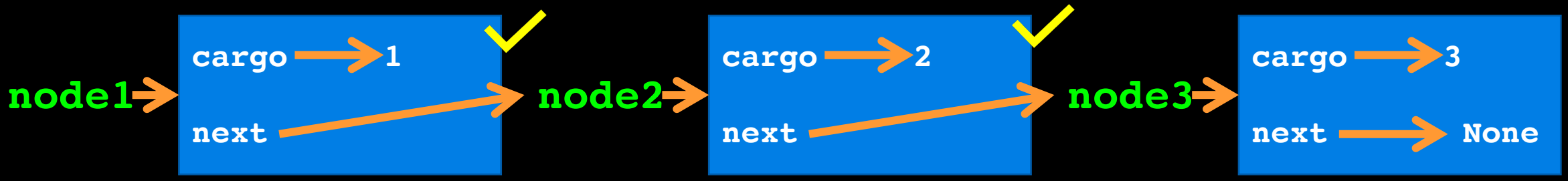
These nodes aren't linked!

The Node Class

- To link the nodes, we have to make the first node refer to the second and the second node refer to the third

```
>>> node1.next = node2
>>> node2.next = node3
```

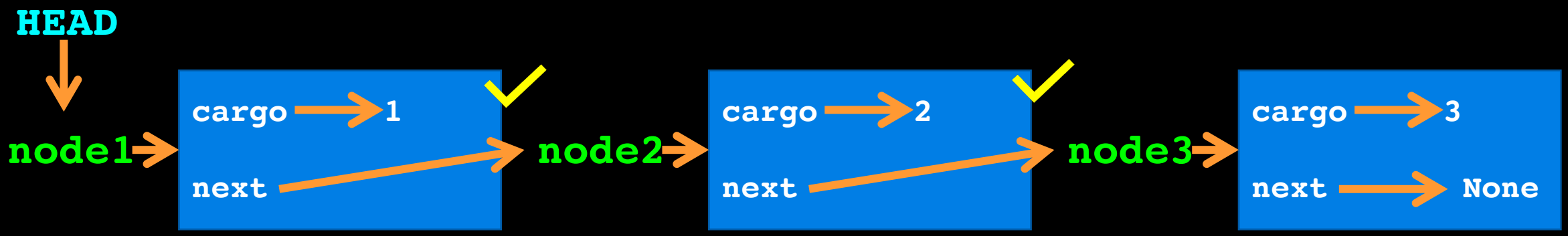
- The reference of the third node is None, which indicates the end of our linked list



These nodes are now linked!

The Node Class

- Now we've assembled multiple objects into a single entity: a collection
- Recall the beginning node we refer to as the head. This head serves as a reference to the entire collection
- To pass our linked list as a parameter, we only have to pass the reference to the first node, or our head node



These nodes are now linked!

Breakout Session!

- Let's look at how this works in Python!
  - Node class
    - Creating nodes
    - Setting cargo and next pointers

Open your notebook

Click Link:
1. The Node Class

Traversing a Linked List

- Let's write a function `print_list` that takes a single node as an argument (usually the head), and prints each node until it gets to the end (tail) of that linked list

```
class Node:
    def __init__(self, cargo=None, next=None):
        self.cargo = cargo
        self.next  = next

    def __str__(self):
        return str(self.cargo)

def print_list(node):

    while node != None:
        print(node, end=" ")
        node = node.next
```

OR `while node:`

```
node1 = Node(1)
node2 = Node(2)
node3 = Node(3)

node1.next = node2
node2.next = node3
```

```
>>> print_list(node1)
1 2 3
```

Traversing a Linked List

- To call this method, we pass a reference to the first node

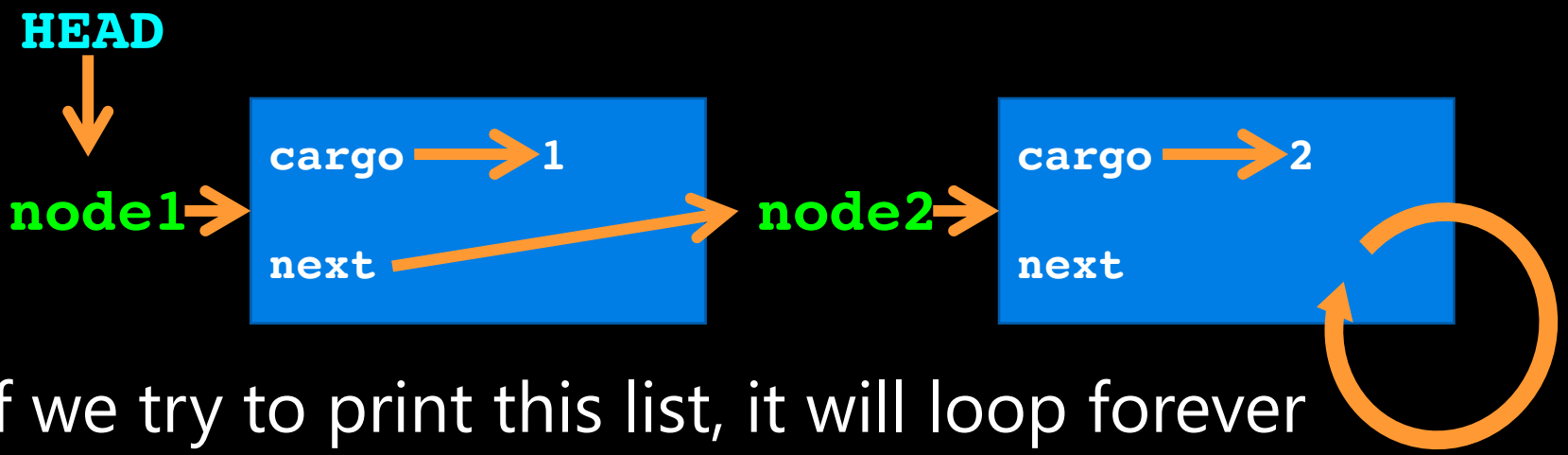
```
>>> print_list(node1)
1 2 3
```

- Inside print_list we have a reference to the first node of the list, but there is no variable that refers to the other nodes. We have to use the next value from each node to get to the next node
- What would happen if we put node2 instead of node1?

```
>>> print_list(node2)
2 3
```

Infinite Lists

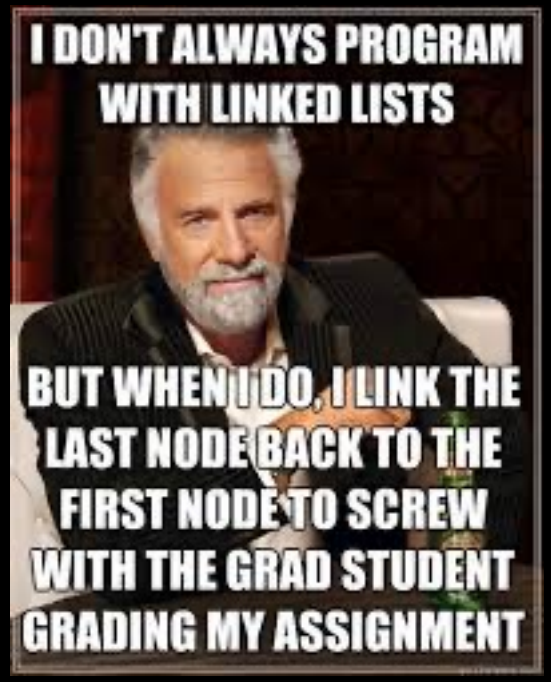
- There is nothing stopping a node from referring back to an earlier node, including itself



- If we try to print this list, it will loop forever
- print_list should come with a pre-condition that no node references a node earlier in the sequence

Let's Code!

- Let's look at how this works in Python!
  - Traversing a Linked List
    - Printing a list
  - Modifying a Linked List
  - BREAKOUT SESSION!



Open your notebook

Click Link:
2. Traversing Linked Lists

APS106

UNIVERSITY OF TORONTO

Advanced Data Structures: Linked Lists

Week 11 | Lecture 2 (11.2)

if nothing else, write #cleancode