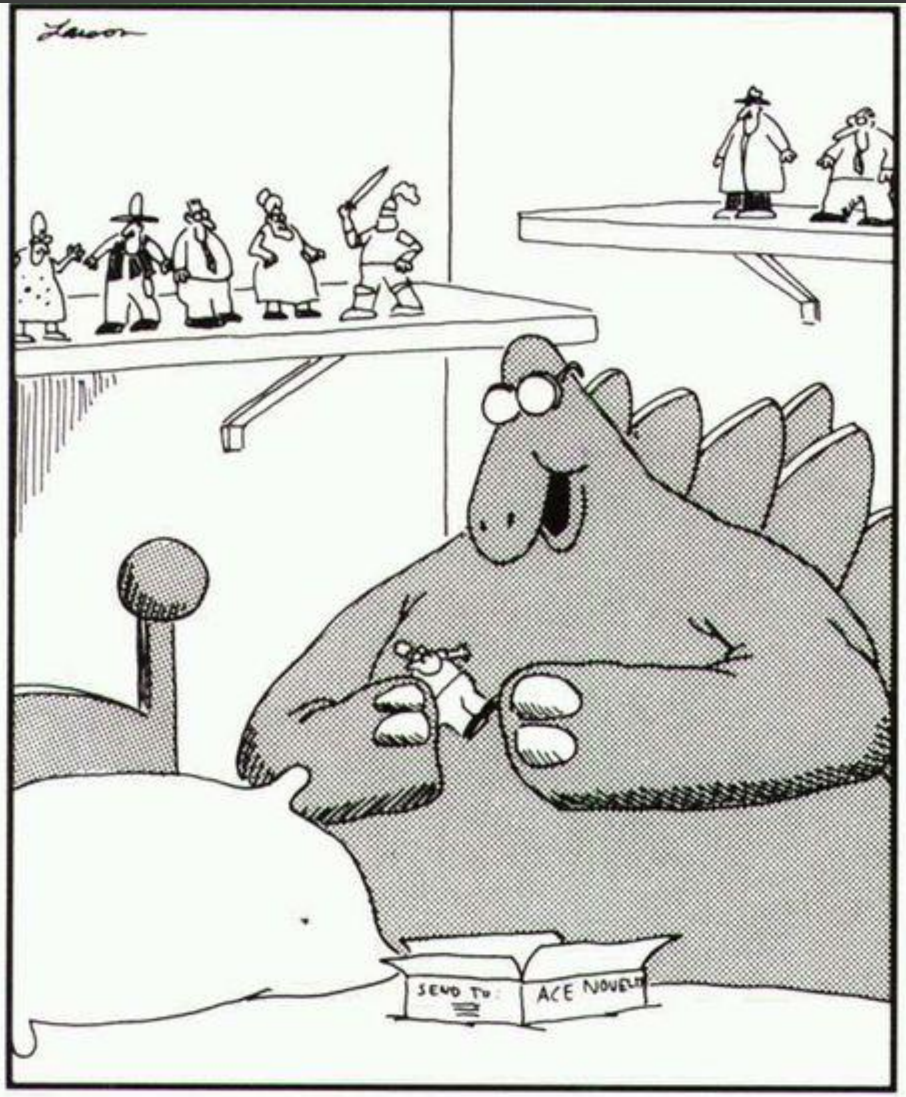
Queues

And a brief introduction to Linked Lists

Spouting off before listening to the facts
is both shameful and foolish.

Proverbs 18:13



"Oh, boy! The 'Nerd'! ... Now my collection's complete!"

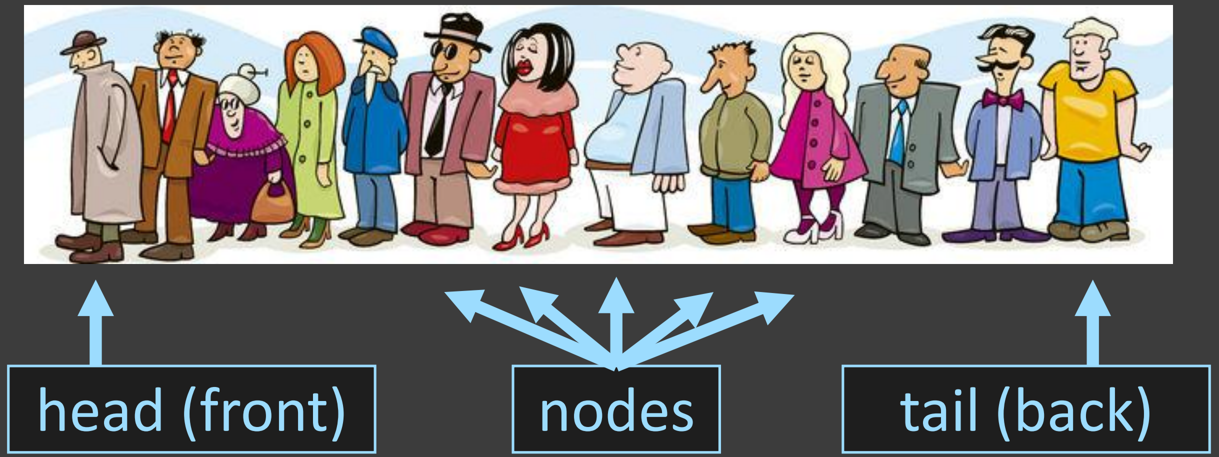
The Far Side by Gary Larsen

Introducing Queues

Simple ADT whose purpose is to keep data in a certain order

- Queues always give back the data item that has been waiting the longest
- First In First Out – FIFO
- Queues already exist for .NET in System.Collections.Generic Queue<T>

stockfreeimages.com



- Used to model grocery lines, implement music playlists, etc.

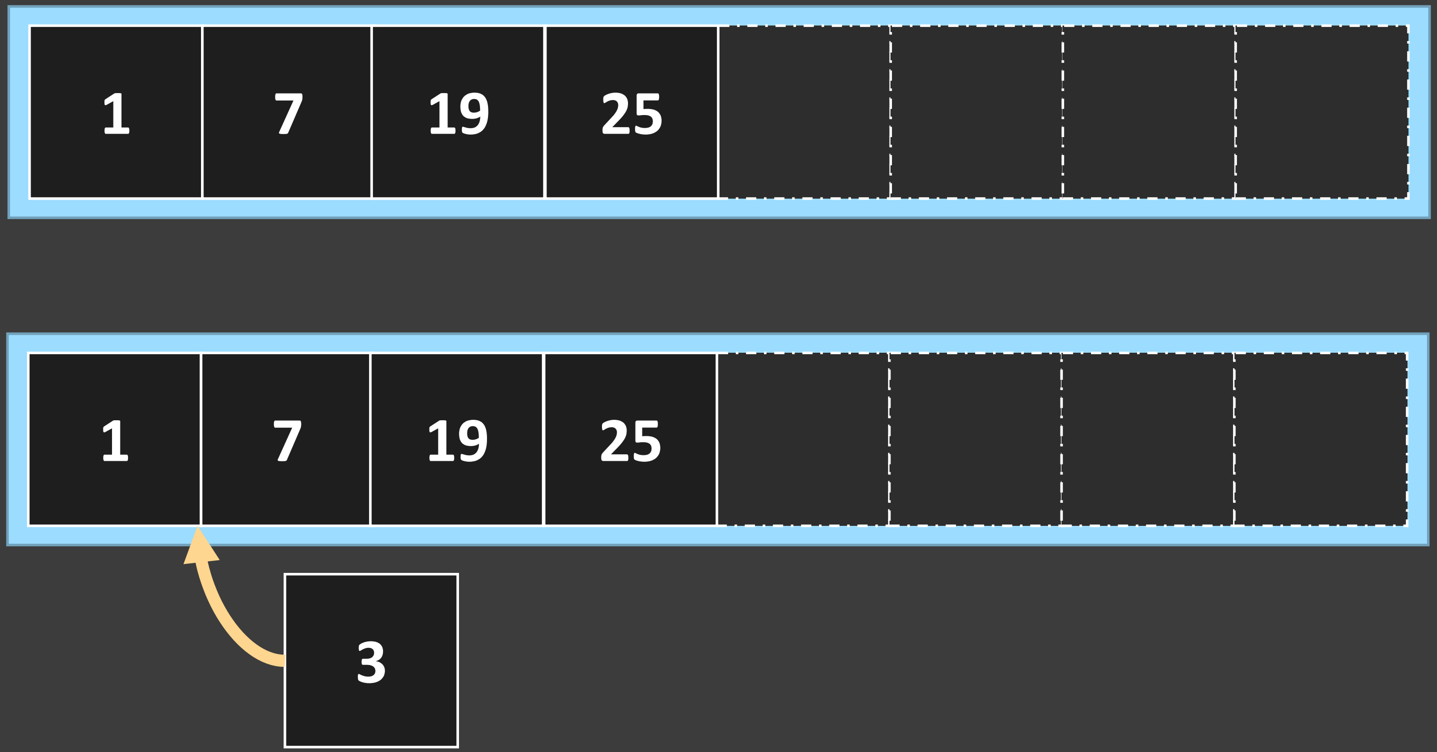
Back to Arrays

Arrays are a sequence of values in memory, all of the same type. All memory is allocated at creation time.

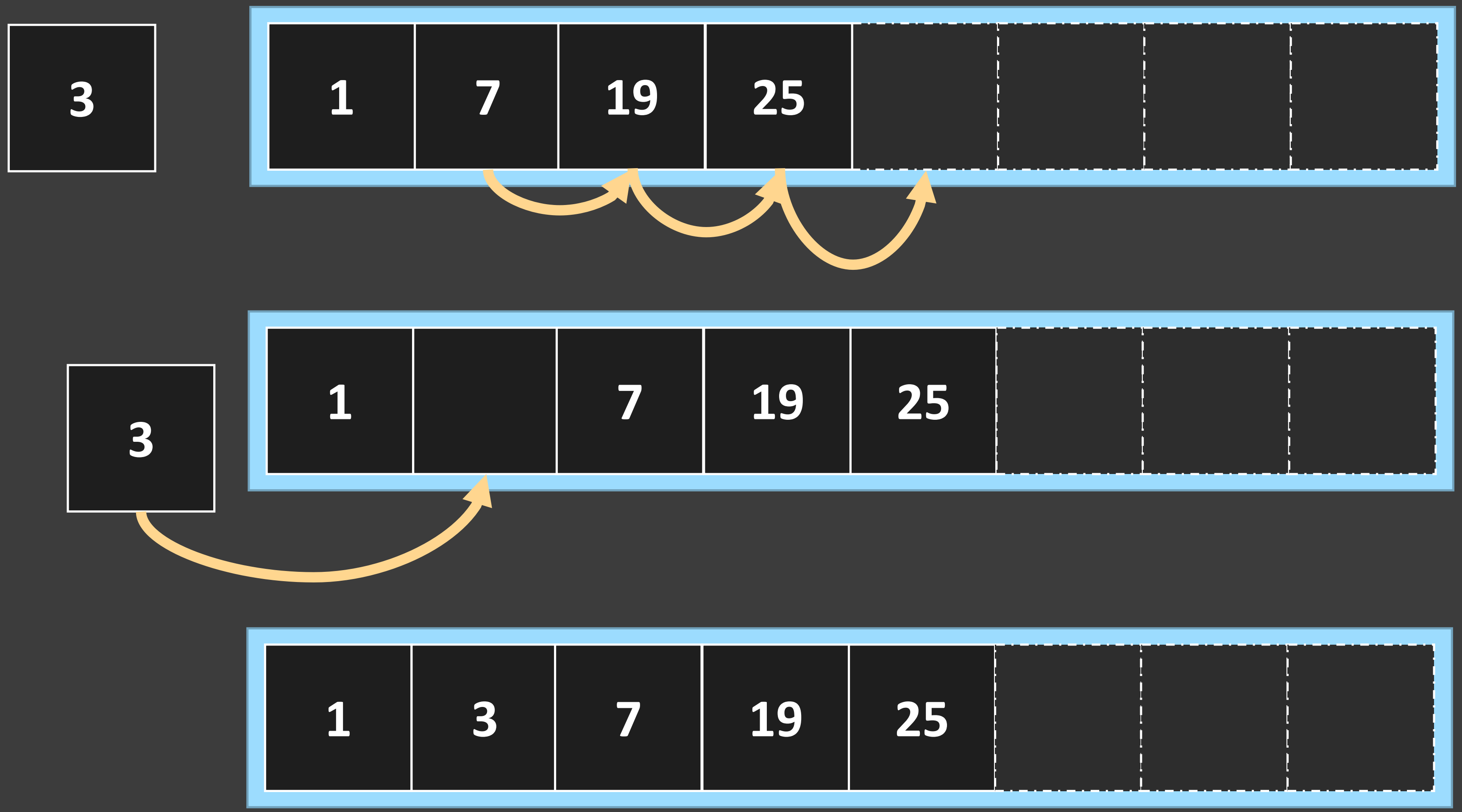
- Benefits of arrays:
  - Easy to use
  - Efficient access if you know the index of an item
  - Good for modeling fixed-sized problems (like a chess board)
- Problems with arrays:
  - Hard to insert new element into the middle of an array
  - Fixed size; cannot grow or shrink without considerable work
  - *The problems get worse as the array gets larger*

Array Shortcoming #1

Hard to insert new element into the middle of an array

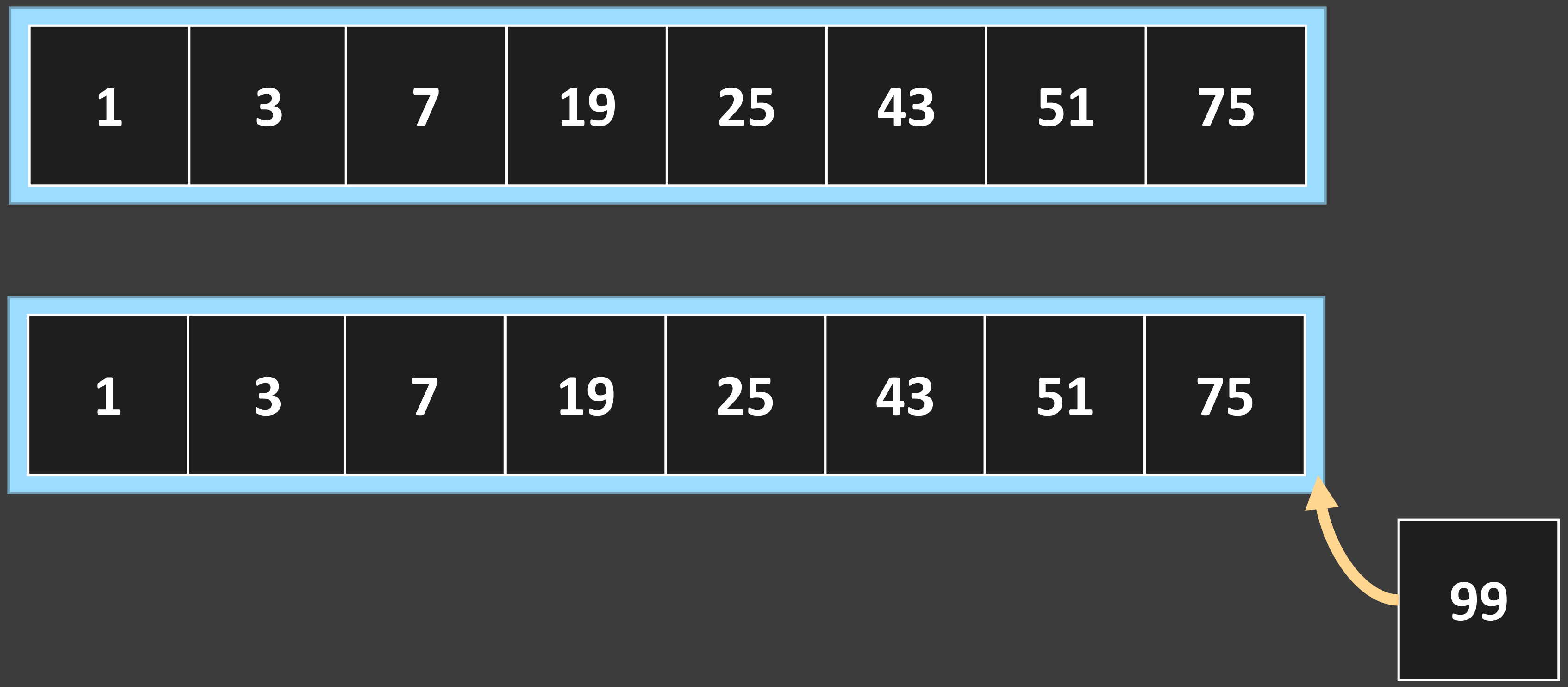


Array Shortcoming #1

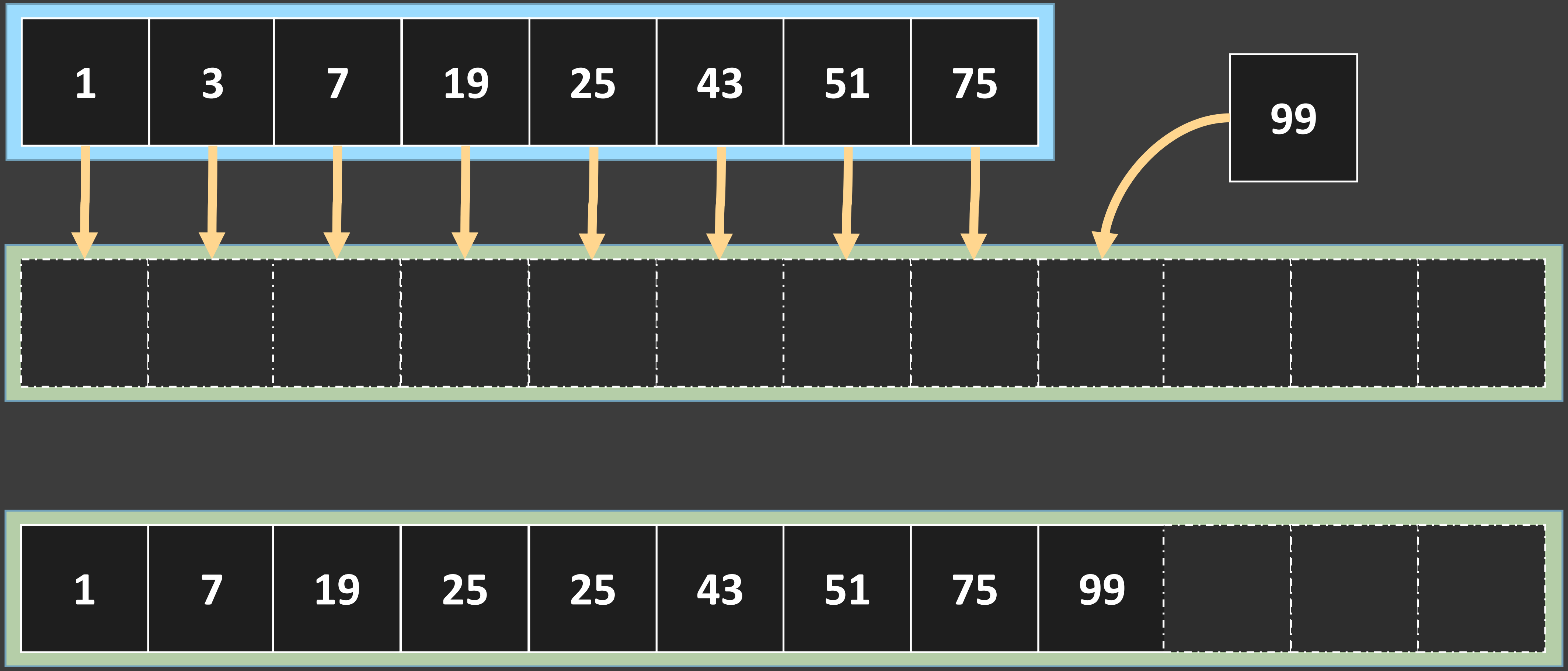


Array Shortcoming #2

Fixed size; cannot grow or shrink without considerable work



Array Shortcoming #2

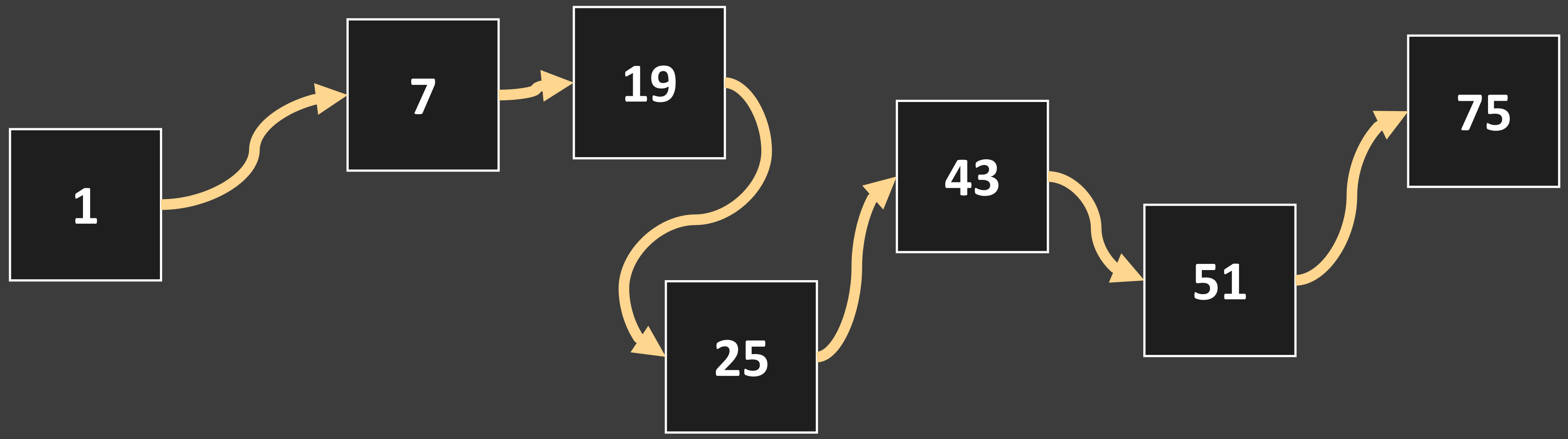


Solutions to Arrays

What if arrays did not require a sequence of contiguous memory?

What if they were spread throughout memory and each element in the array knew the location of the next element?

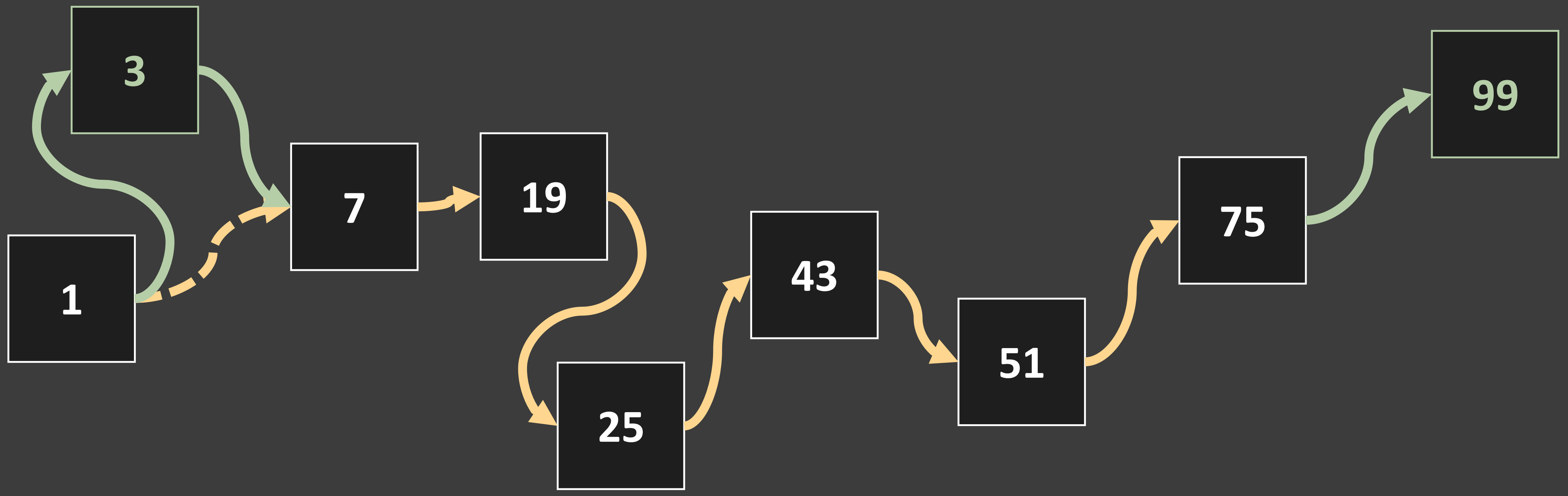
Then it would be more efficient to insert new items and grow the array



Solutions to Arrays

There is a data structure like this, it’s called a linked list

It solves the problems of arrays using linked nodes



Linked List Nodes

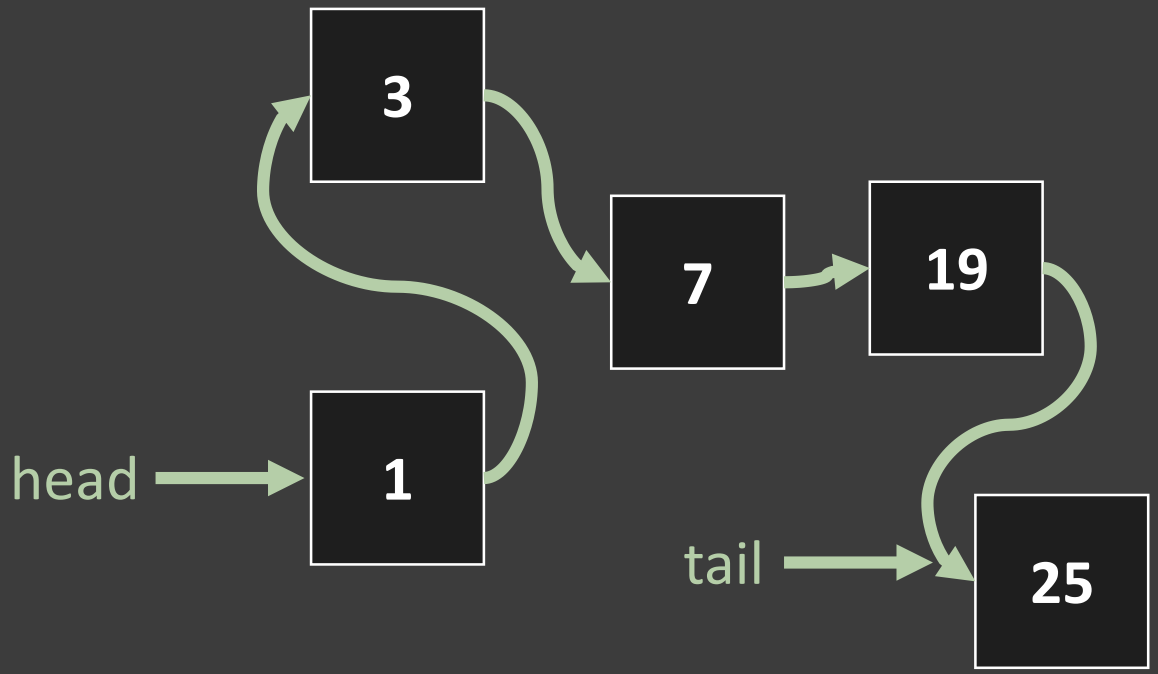
Nodes hold a pointer to the next item and the actual data

```
class Node<T>
{
  public T data;
  public Node<T> next;

  public Node() {
    data = default;
    next = null;
  }
}
```

Linked Lists

Linked lists objects encapsulate nodes by containing a pointer to the first node (usually called head) and the last node (usually called tail)

```
class LinkedList<T>
{
  private Node<T> head;
  private Node<T> tail;

  public LinkedList() {
    head = null;
    tail = null;
  }
}
```



Adding and Removing Nodes

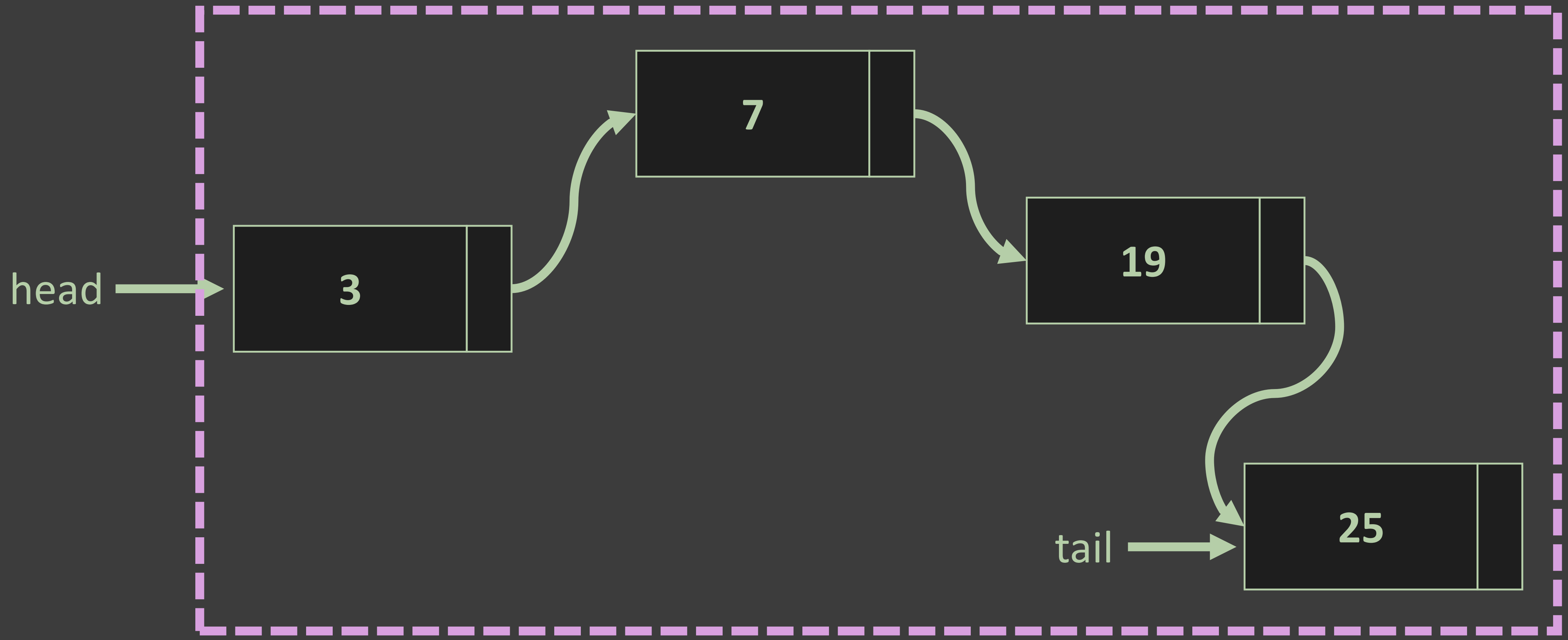
Add nodes to the tail of the linked list

1. Create a new node and set the data value
2. Take the tail pointer (which has been null) and point it at the new node

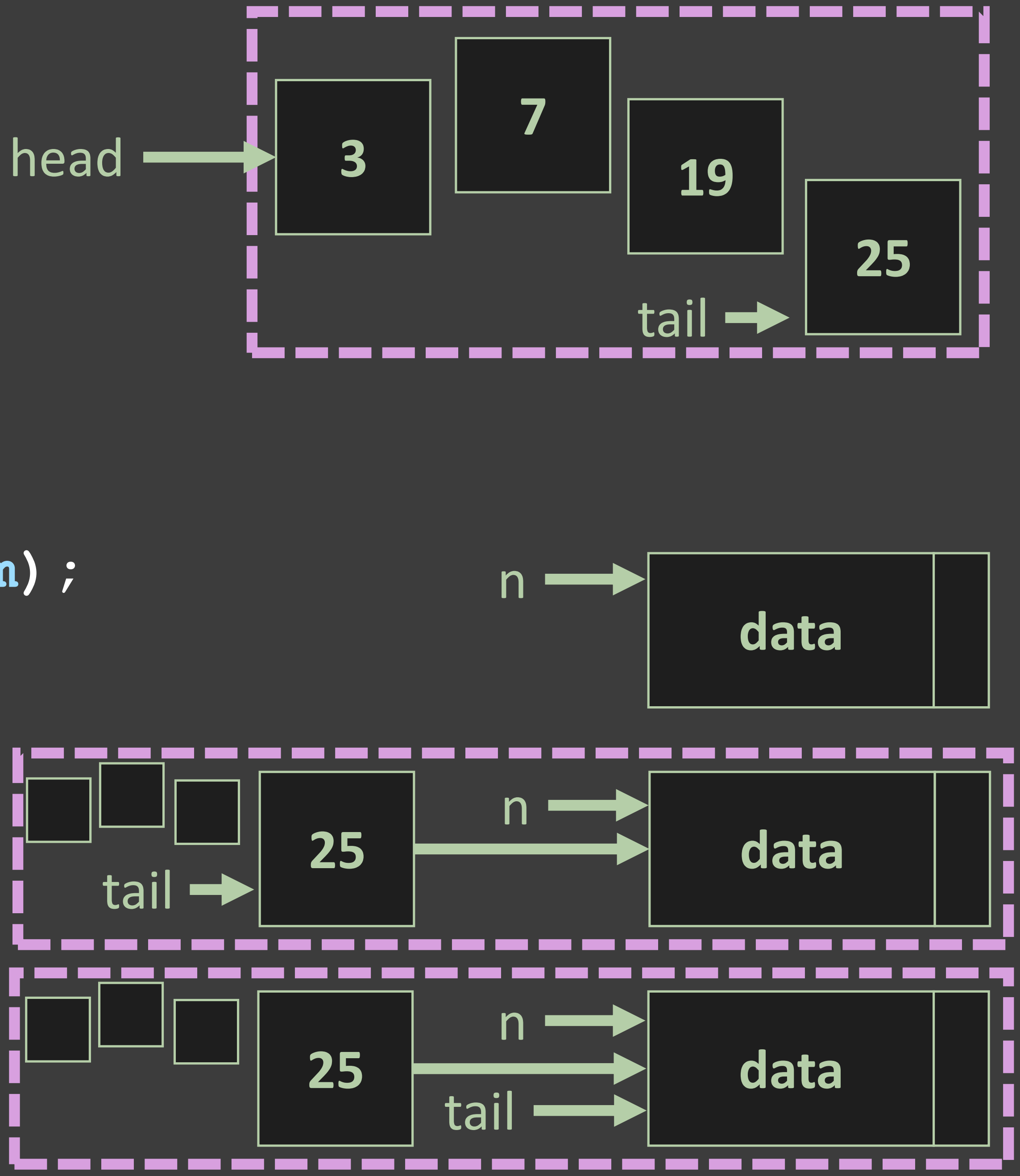
Remove nodes from the head of the linked list

1. Create a 2^{nd} pointer to the head (this will be temporary)
2. Take the head pointer itself and re-point it at the second node in the list
3. return the temporary pointer to the node that has been removed

Adding and Removing Nodes



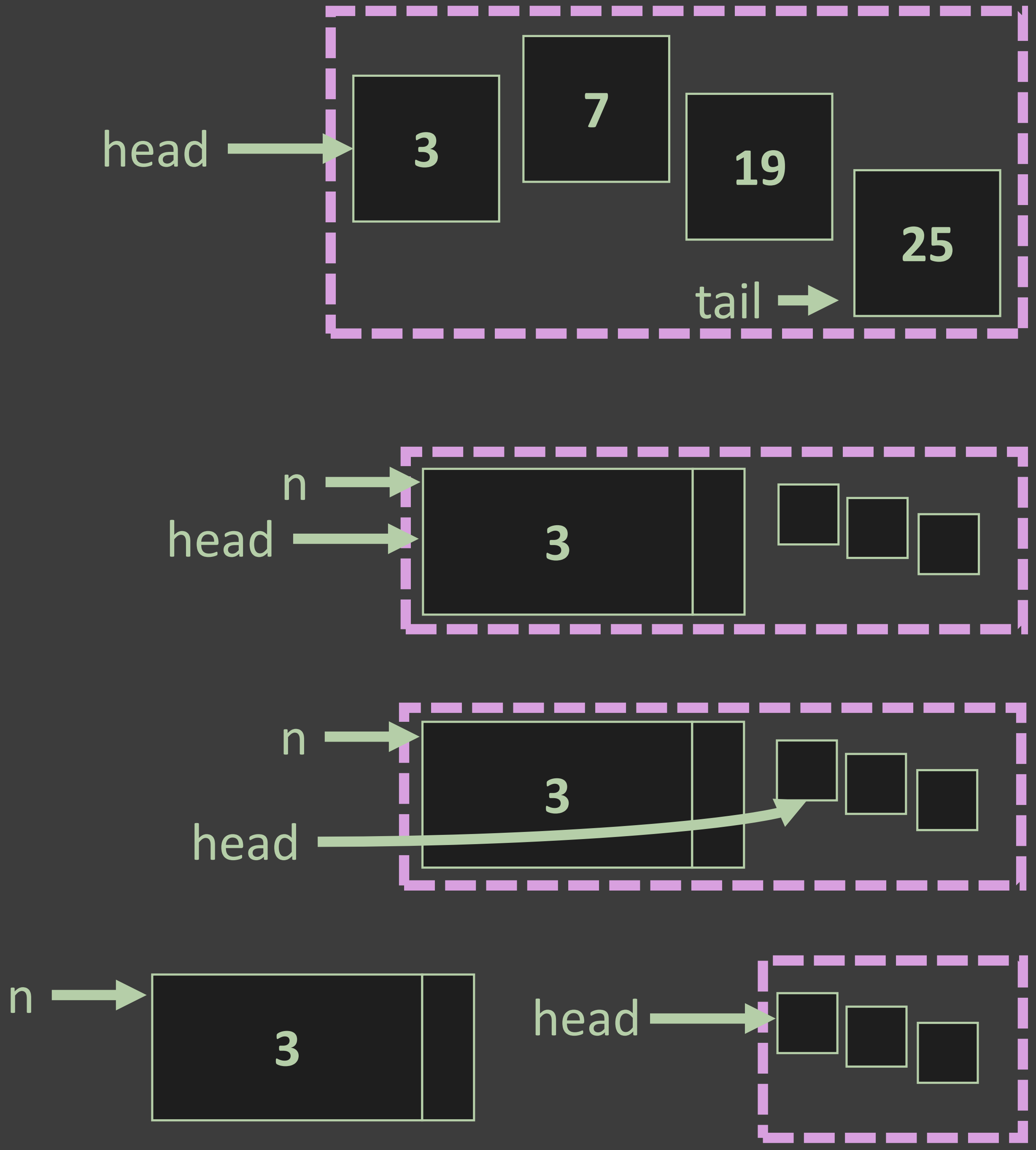
Adding Nodes



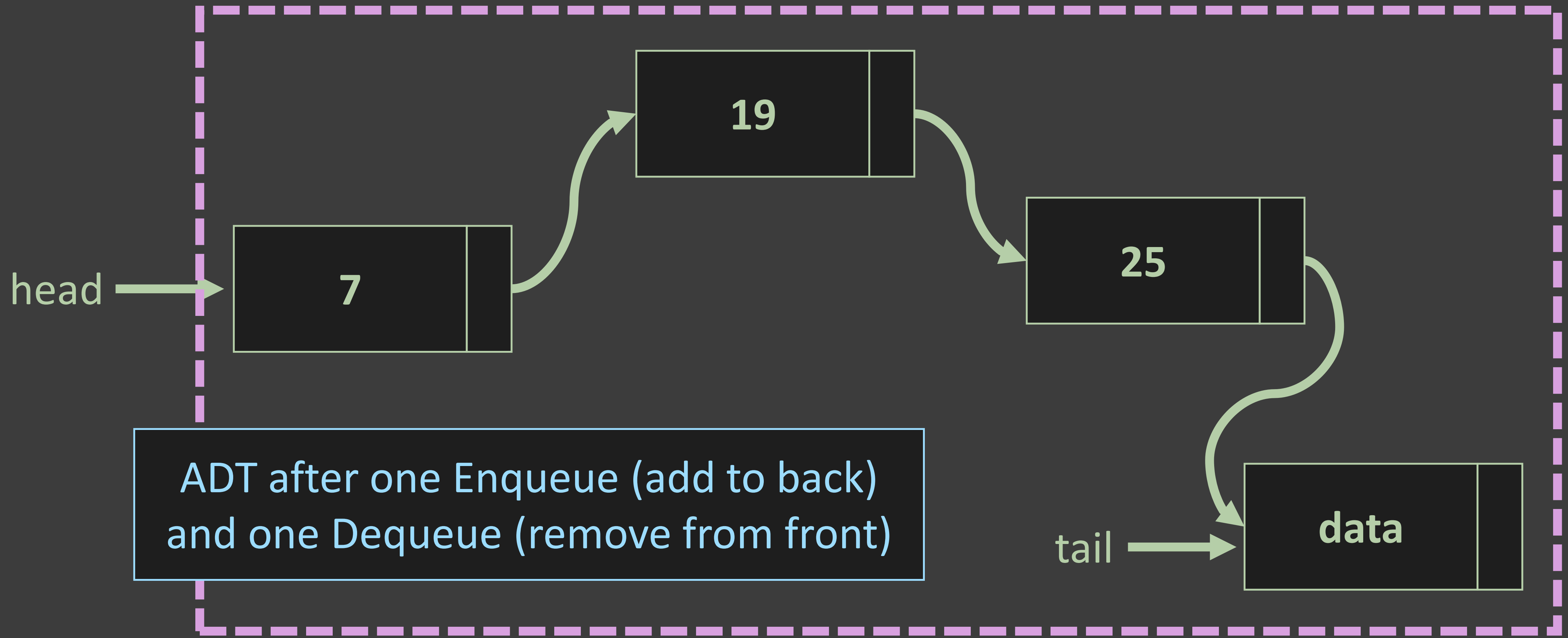

```
public void Enqueue(T item)
{
   Node<T> n = new Node<T>(item);
   tail.next = n;
   tail = n;

   // if queue is empty?
}
```

Removing Nodes

```
public T Dequeue()
{
  // if queue is empty?

  Node<T> n = head;
  head = head.next;
  n.next = null;
  return n.data;
}
```



Adding and Removing Nodes



ADT after one Enqueue (add to back)
and one Dequeue (remove from front)

Homework

Read the end of section 1.3 in textbook (pages 142-153) as a recap of this lecture

Create a queue library and test program

public class MyQueue<T>	adapted from p 151
MyQueue()	create an empty queue
void Enqueue(T item)	add an item
T Dequeue()	remove the least recently added item
bool IsEmpty()	is the queue empty?
int Count	number of items in the queue