Node-based Stacks and Queues

CMPT 145

Objectives

After this topic, students are expected to be able to:

1. Employ data abstraction to implement a queue.
2. Employ data abstraction to implement a stack.
3. Describe the node-based implementation of stacks and queues.

Recap: Stack ADT

- Purpose:
  - Manage a LIFO-ordered sequence
- Implementation:
  - Data structure:
    - Encapsulated node-chain
  - Essential Operations:
    - Create empty stack
    - Query if stack is empty
    - Query size of stack
    - Push value onto stack
    - Pop value off of stack
    - Peek at the top of the stack

A simple node-based Stack Implementation

- Implementation: A dictionary/record with 2 fields:
  - A reference to a chain of nodes
  - A counter for the number of nodes
- Push: adds a node to the front of the chain
- Pop: removes the front of the chain
- Size: returns the number of items in the chain
- IsEmpty: checks if the chain has no nodes in it

The Stack Data Structure

The Stack data structure is a dictionary with the following keys:

size This keeps track of how many values are in the list.

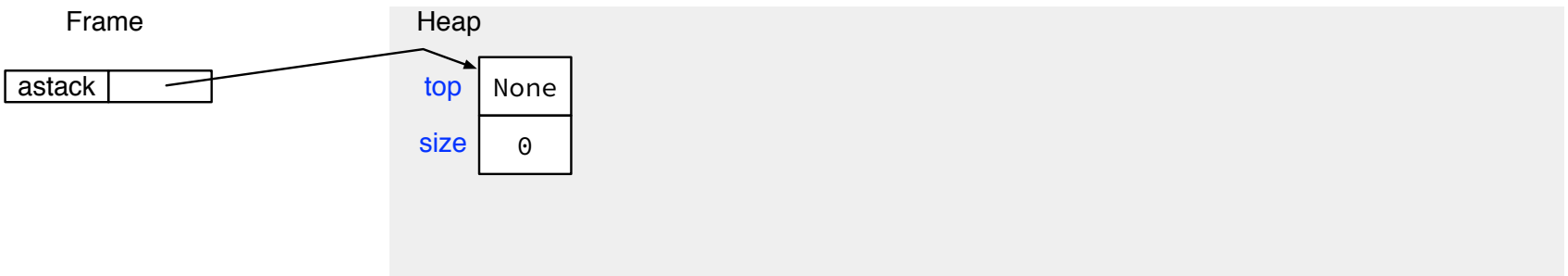
top This is a reference to the first node in the node chain. An empty Stack has no node chain, which we represent with None.

Implementing `create()`

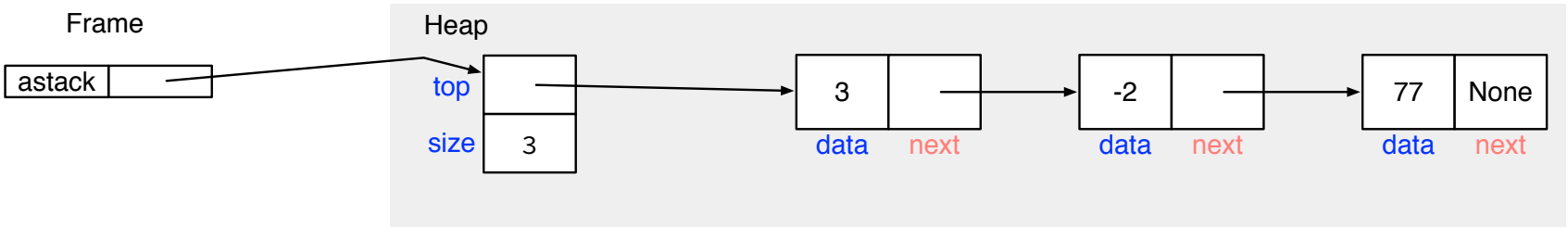
The `create()` operation is as follows:

```
1  def create():
2      """
3      Purpose
4          creates an empty stack
5      Return
6          an empty stack
7      """
8      stack = {}
9      stack['size'] = 0          # how many elements in the stack
10     stack['top'] = None        # the node chain starts here
11     return stack
```

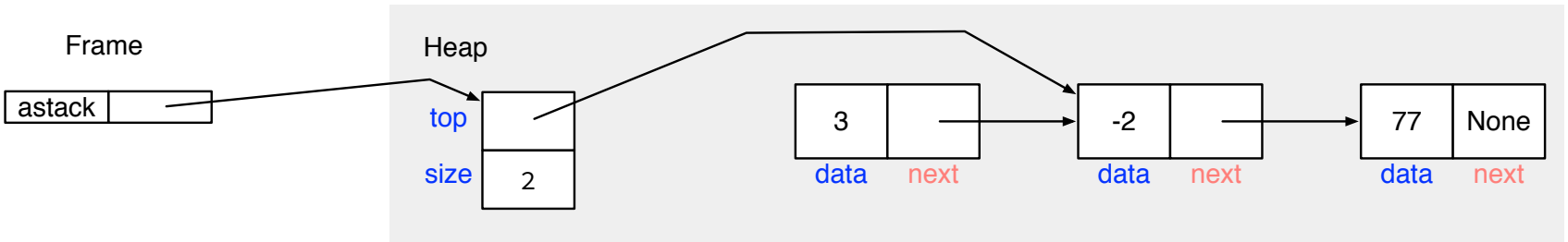
An Empty Stack on the Heap



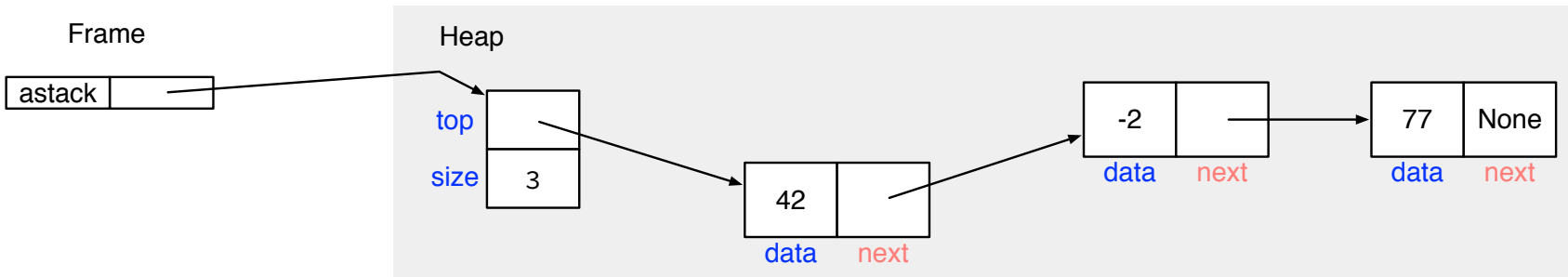
A Non-empty Stack on the Heap



The Pop Operation



The Push Operation



Special Cases

What is the effect of Stack operations on:

- An empty Stack?
- A Stack with one value?
- A Stack with several items?

Special Cases

A special case is special
not because it is rare,
but because
a general rule does not apply.

It is your responsibility to identify special cases as part of your design.

Special Cases for Node-chains

Take special care in these situations:

- Any action on an empty chain.
- Any action on a chain with exactly one node.
- Any action affecting the first node.
- Any action affecting the last node.
- Any linear search when the target is not in the chain.

Code walk-through

Recap: Queue ADT

- Purpose:
  - Manage a FIFO-ordered sequence
- Implementation:
  - Data:
    - Encapsulated node-chain
  - Essential Operations:
    - Create empty queue
    - Query if queue is empty
    - Query size of queue
    - Enqueue value
    - Dequeue value
    - Peek at the first value

A simple node-based Queue Implementation

- Implementation: A dictionary/record with 3 fields:
  - A reference to the start of the chain of nodes
  - A reference to the end of the chain of nodes
  - A counter for the number of nodes
- Enqueue: adds a node to the end of the chain
- Dequeue: removes the front of the chain
- Size: returns the number of items in the chain
- Empty?: checks if the chain has no nodes in it

The Queue Data Structure

The Queue data structure is a dictionary with the following keys:

size This keeps track of how many values are in the list.

front A reference to the first the node chain. An empty Queue has no node chain, which we represent with None.

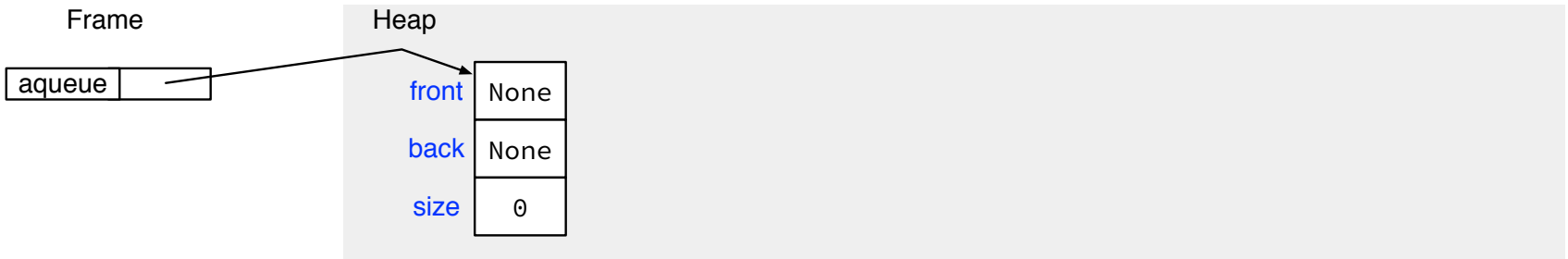
back A reference to the last node in the node chain that stores the queued values. If the queue is empty, this should be the Python value `None`.

Implementing `create()`

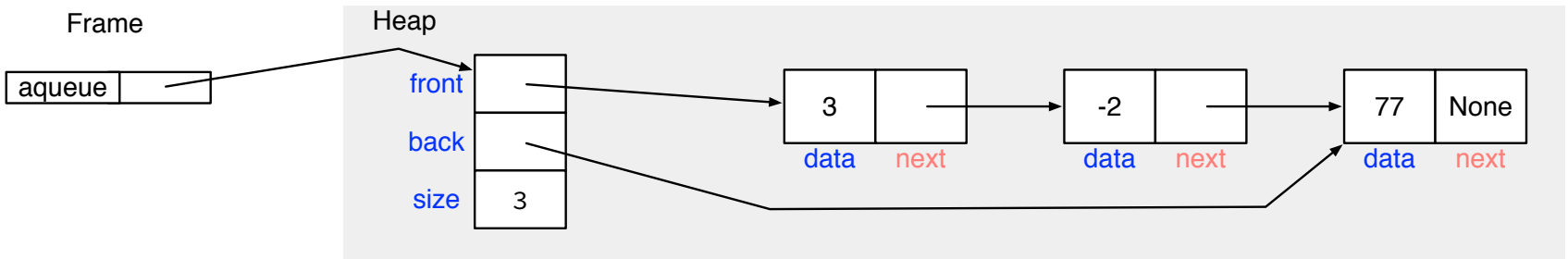
The `create()` operation is as follows:

```
1  def create():
2      """
3      Purpose
4          creates an empty queue
5      Return
6          an empty queue
7      """
8      queue = {}
9      queue['size'] = 0          # how many elements in the queue
10     queue['front'] = None      # the node chain starts here
11     queue['back'] = None       # the node chain ends here
12     return queue
```

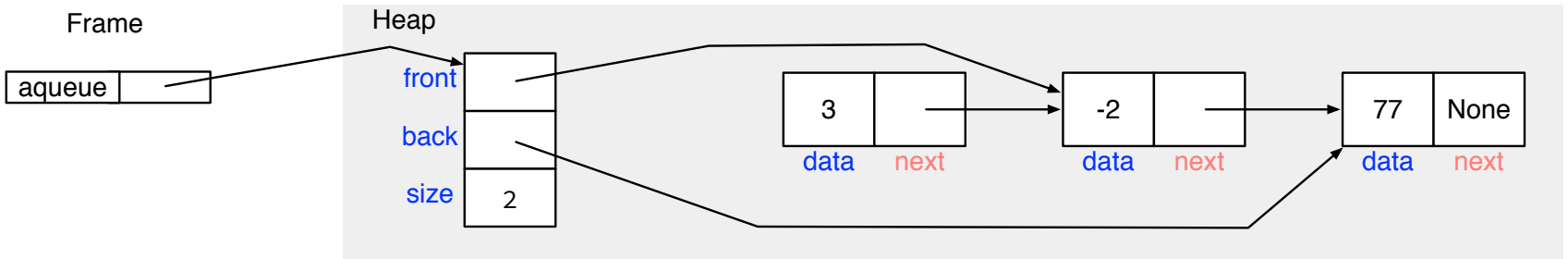
An Empty Queue on the Heap



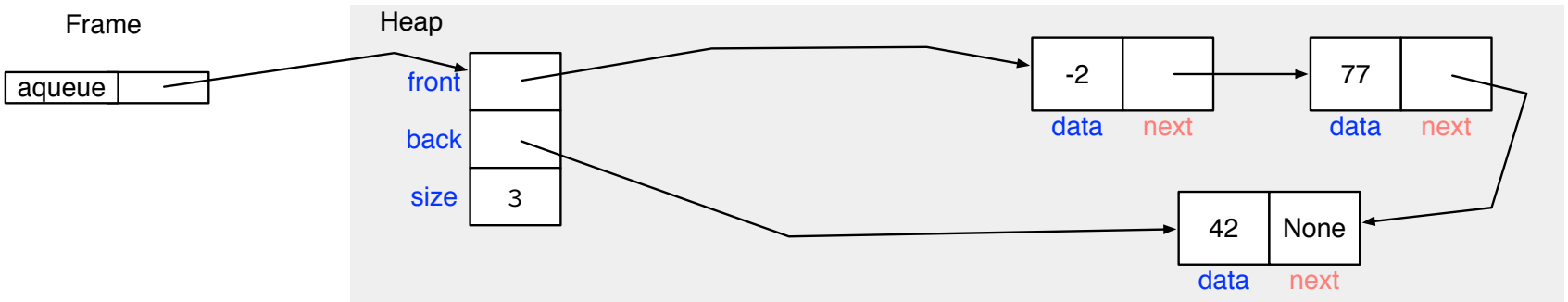
A Non-empty Queue on the Heap



The Dequeue Operation



The Enqueue Operation



Special Cases

What is the effect of Queue operations on:

- An empty Queue?
- A Queue with one value?
- A Queue with several items?

Code walk-through

Questions

- In Queue, how could we enqueue new elements at front, and dequeue elements at back?
- In Stack, how could we push and pop from the far end of the chain?
- Compare Queue and Stack implementations from today and the implementations from Chapter 5 in terms of:
  - Correctness
  - Efficiency
  - Robustness
  - Adaptability
  - Reusability

  Which implementation is better?

Exercise 1

1. Change the Queue code so that the front and back are reversed.
2. Change the Stack code so that the top is on the opposite end of the list.

In both cases, change the ADT implementation only. Code that uses Queue or Stack will not change!