Nodes: A Basis for Implementing Linear Data Structures

CMPT 145

Objectives

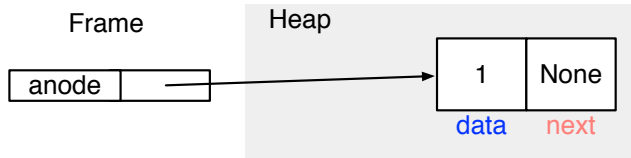
After this topic, students are expected to

1. Explain the way records can link to records of the same type
2. Use linked records in simple expressions.
3. Draw diagrams of linked records.
4. Use linked records in simple algorithms.

Data Structure: Node records

A node is a very simple dictionary:

```
1 anode = {'data': 1,
2          'next': None}
```



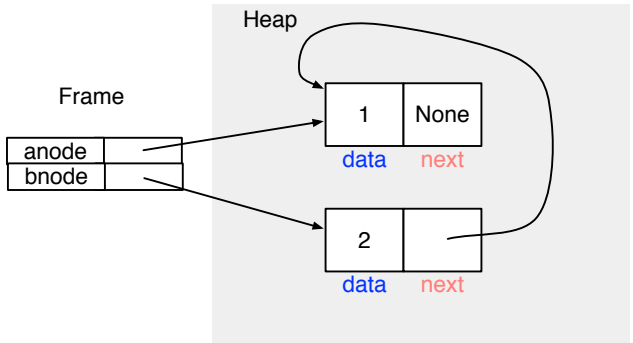
It stores 2 values only.

1. A data value
2. A reference to another node (or `None`)

Node records link nodes together

Two nodes, linked together:

```
1 anode = {'data': 1,
2          'next': None}
3 bnode = {'data': 2,
4          'next': anode}
```

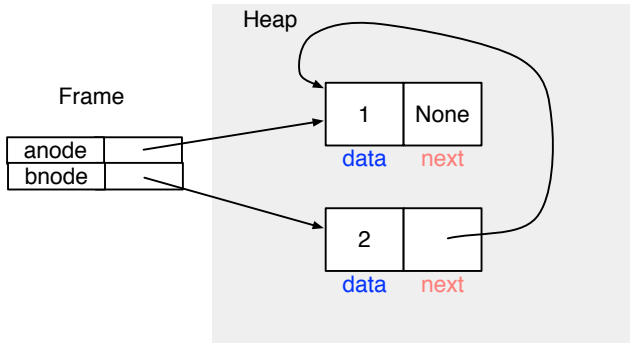


We use the dictionary key `'next'` to store a reference to another node.

An ADT hides all this nicely

Two nodes, linked together:

```
1 import node as node
2 anode = node.create(1,None)
3 bnode = node.create(2,anode)
```



We use this ADT to implement other kinds of things!

Node ADT

- Purpose:
  - Store data sequences.
- Implementations:
  - Dictionary with 2 values:
    1. A data value
    2. A reference to another node (or `None`)
- Operations:
  - Create a node
  - Set the data value for a given node
  - Set the reference to the next node for a given node
  - Return the data value of a given node
  - Return the reference to the node of a given node

Code Walk Through

Python keyword arguments

- Normal function parameters are based on position.

```
1 def fun3(a, b, c):
2     pass
3 fun3(1,2,3)
```

- keyword arguments use the parameter name:

```
1 def fun2(a, b, c=0):
2     pass
3 fun2(1,2,c=3)
```

- The assignment in the parameter list establishes a default value

Python keyword arguments

- You only need to give a value if you want something other than the default:

```
1 fun2(1,2,c=3)   # ignore the default value
2 fun2(1,2)       # use the default value
```

- Position based parameters must precede keyword arguments in the definition.

Exercise

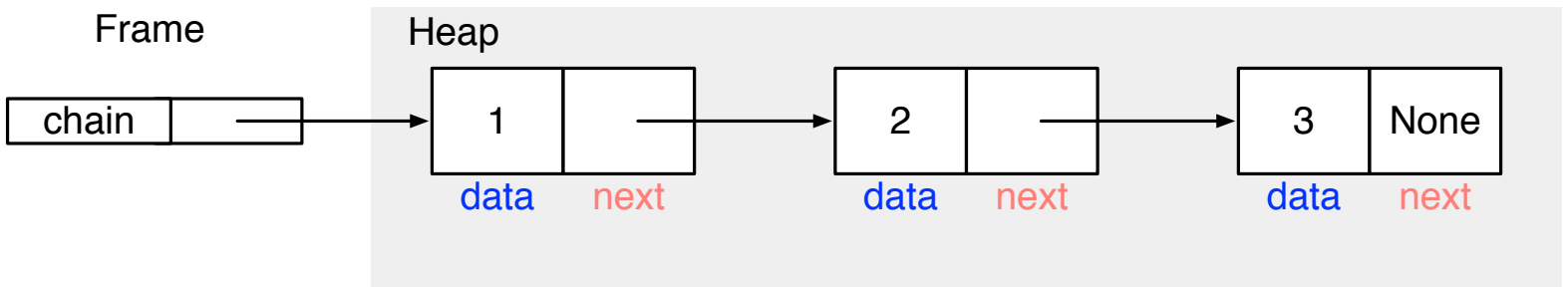
Draw a diagram for the following code sequence:

```
1  x = node.create(5, None)
2
3  y = node.create(1, x)
4
5  z = node.create(8, y)
6
7  print(node.get_data(x))
8  print(node.get_data(node.get_next(z)))
9  print(node.get_data(node.get_next(node.get_next(z))))
```

You cannot do this reliably in your head. Draw a diagram.

Exercise

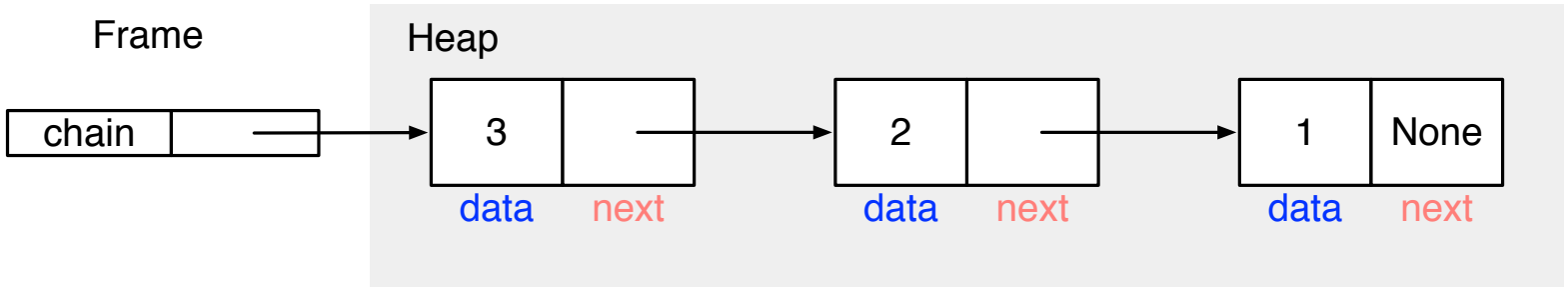
Write the code to produce the following sequence:



Use the ADT only!

Exercise

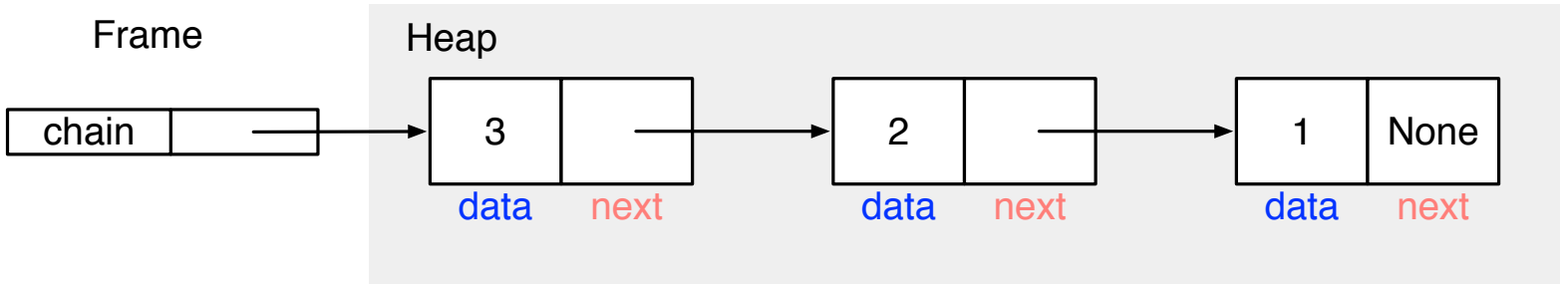
Write the code to produce the following sequence:



Use the ADT only!

Exercise

Given the following sequence:



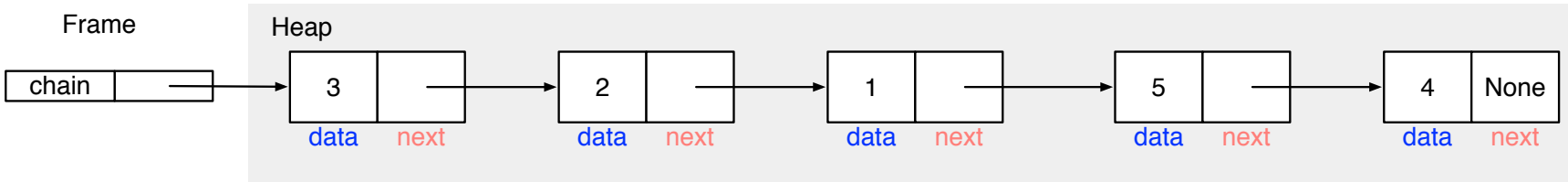
Write a print statement using the above chain that:

1. Evaluates to 1
2. Evaluates to 6
3. Evaluates to 9

Use the ADT only!

Simple algorithms on Node records

Suppose the variable `chain` is a reference to the first node in the sequence:

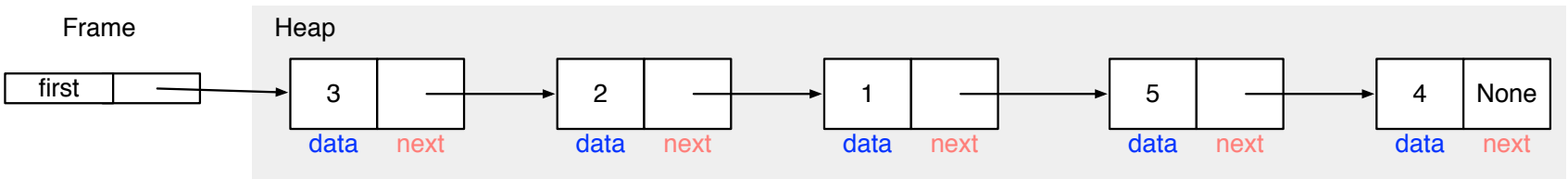


Use the Node ADT to:

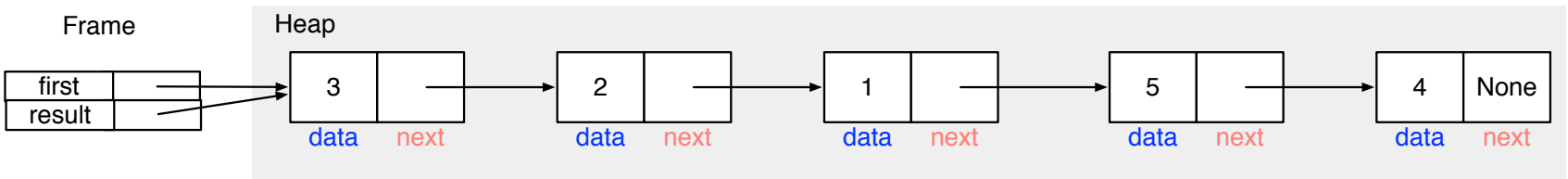
1. Remove the 3 from the sequence
2. Add a new value 6 at the beginning of the sequence
3. Add a new value 7 at the end of the sequence

Use the ADT only!

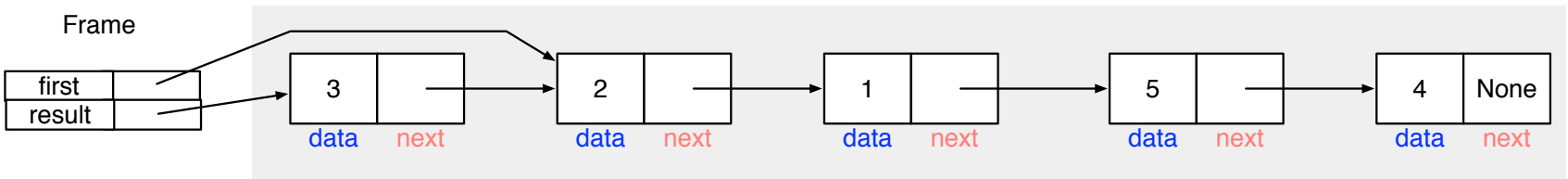
Removing 3 from the sequence



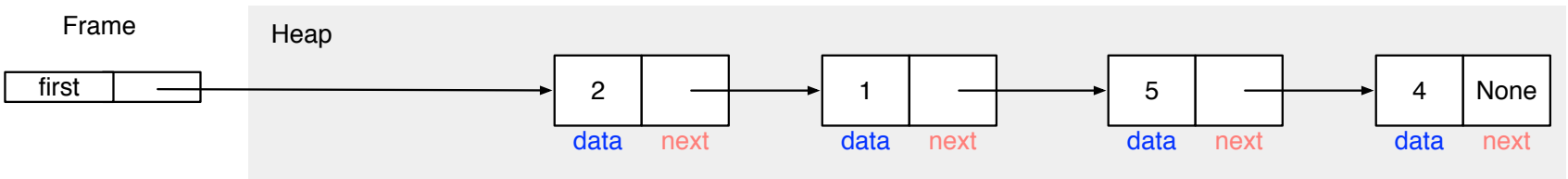
Removing 3 from the sequence



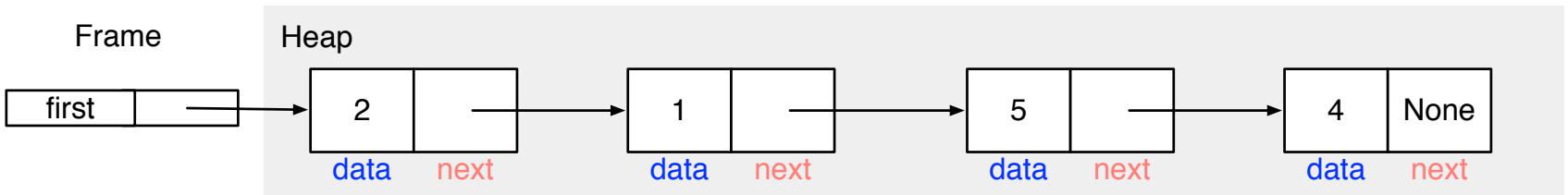
Removing 3 from the sequence



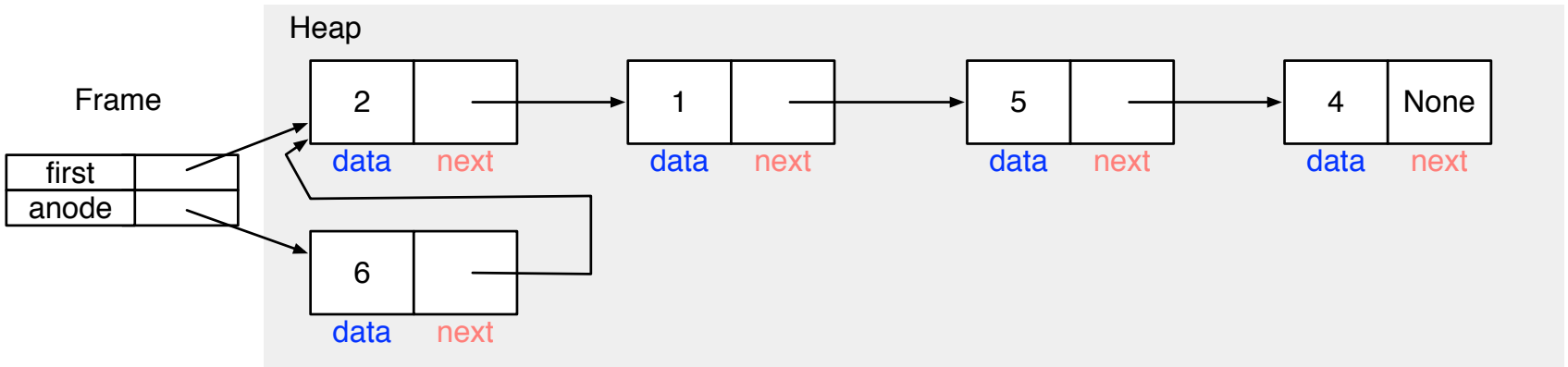
Removing 3 from the sequence



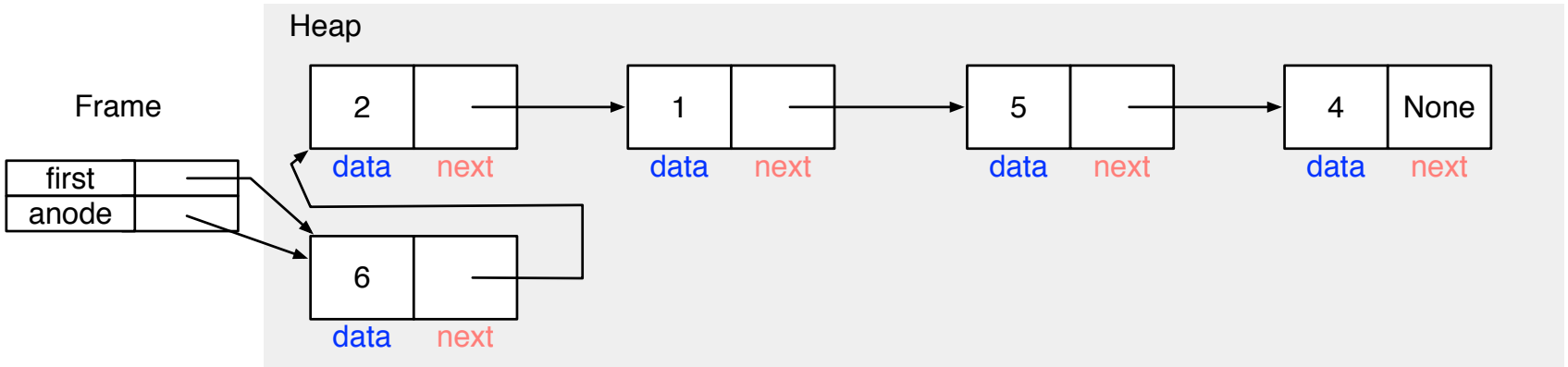
Add 6 at the beginning



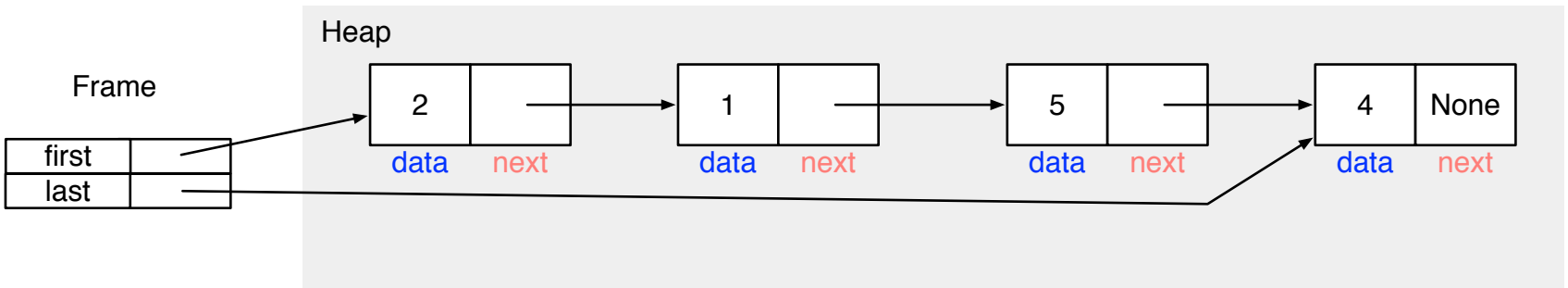
Add 6 at the beginning



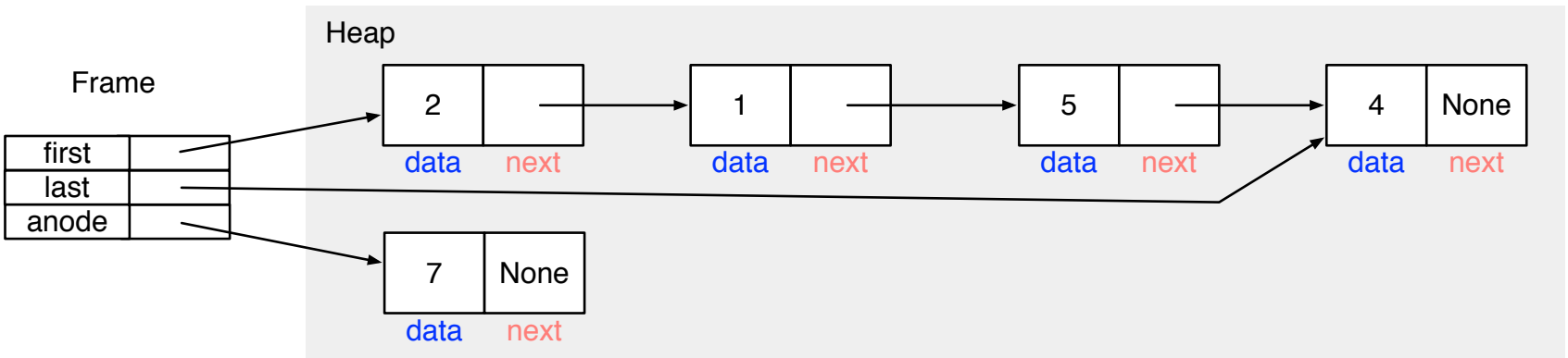
Add 6 at the beginning



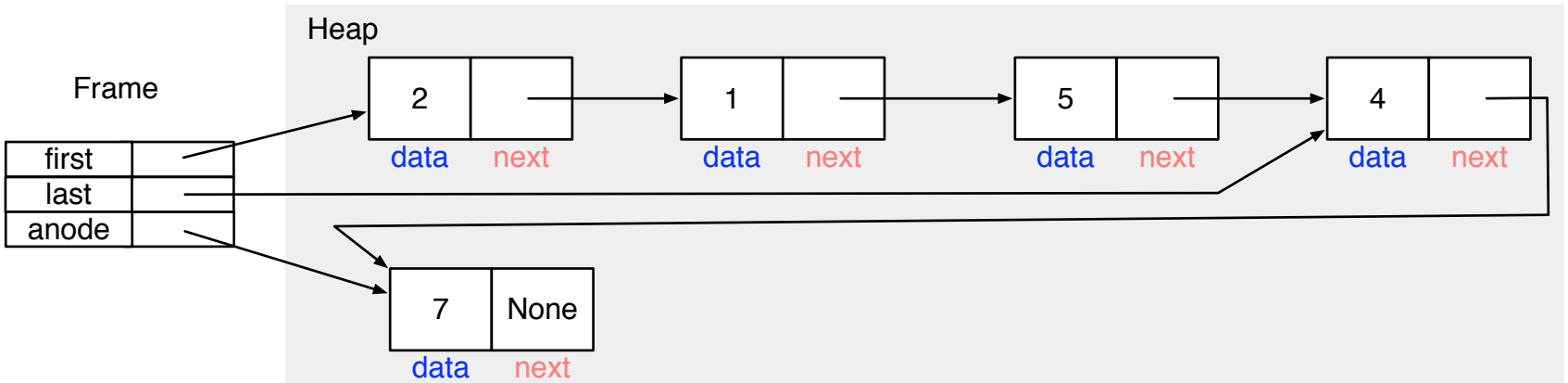
Add 7 at the end



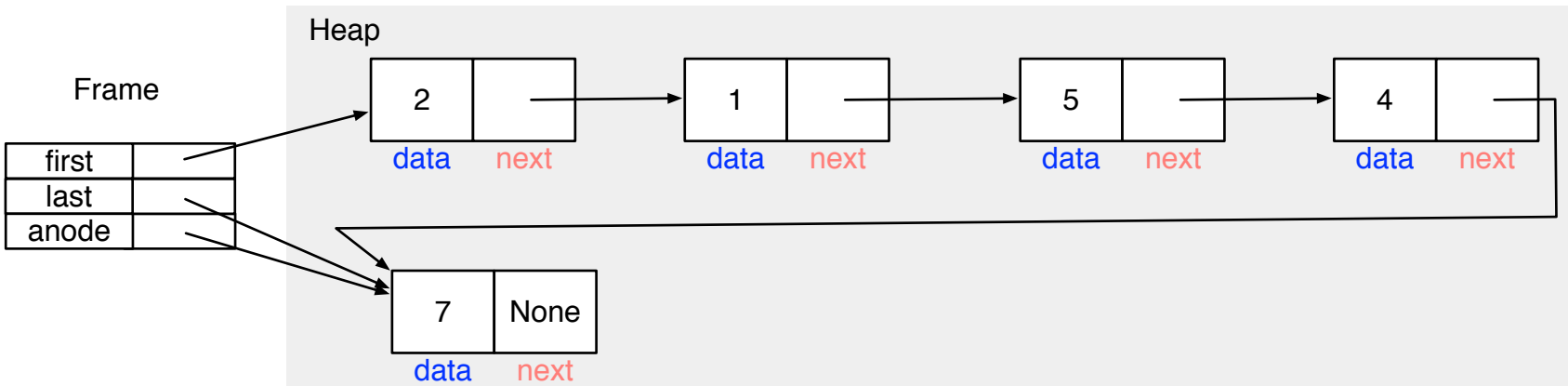
Add 7 at the end



Add 7 at the end



Add 7 at the end



The Node ADT

- A simple data structure, hidden behind an interface.
- Chaining nodes together creates a sequence.
- Stacks and queues can be implemented using nodes.
- Nodes are seriously valuable!