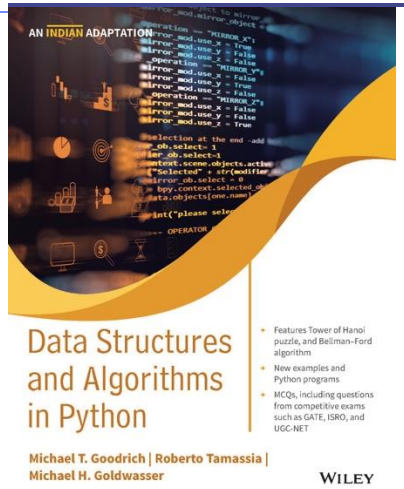


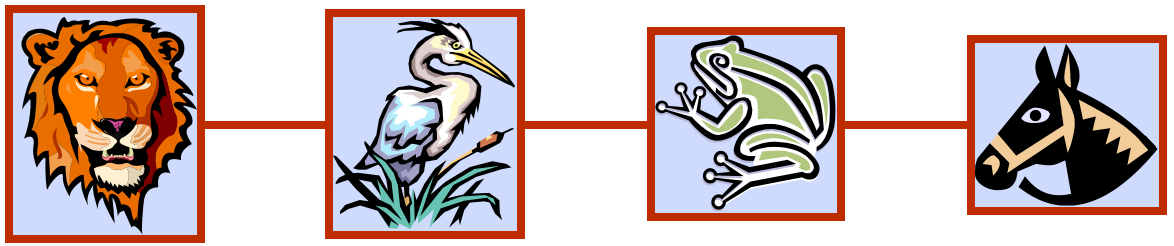
Data Structures and Algorithms in Python

Michael T. Goodrich/ Roberto Tamassia/ Michael H. Goldwasser

Adapted by Subhrakanta Panda

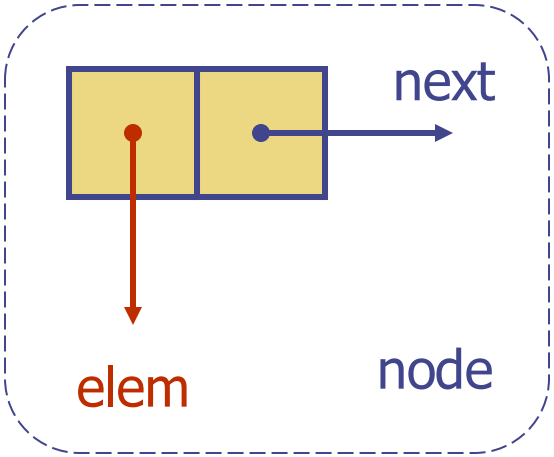
Chapter 8
Linked Lists

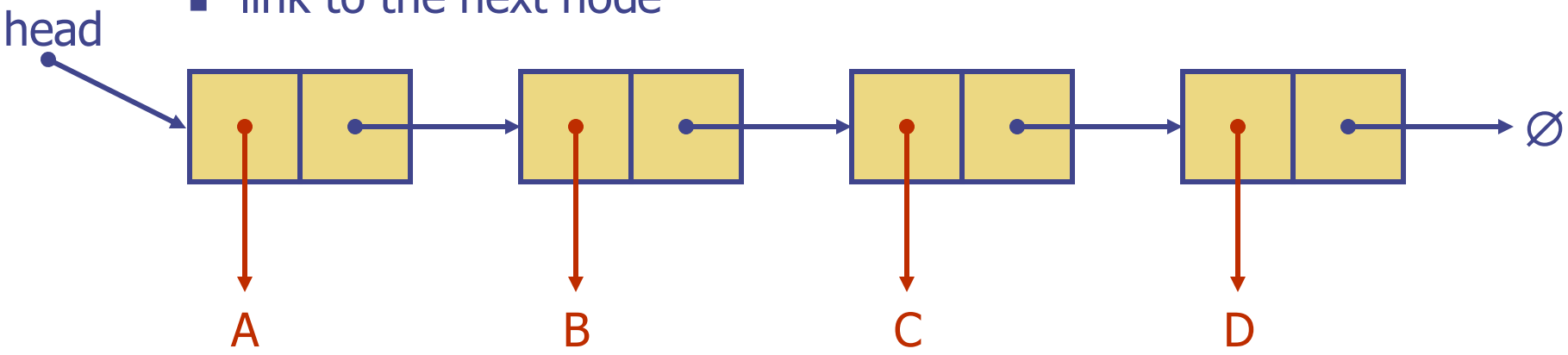
Linked Lists



Singly Linked List

- A singly linked list is a concrete data structure consisting of a sequence of nodes, starting from a head pointer
- Each node stores
  - element
  - link to the next node



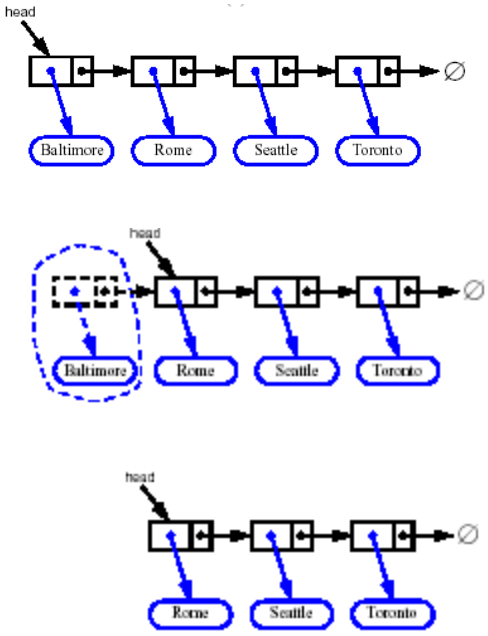


The _Node Class and its Methods

Refer to Class Notes

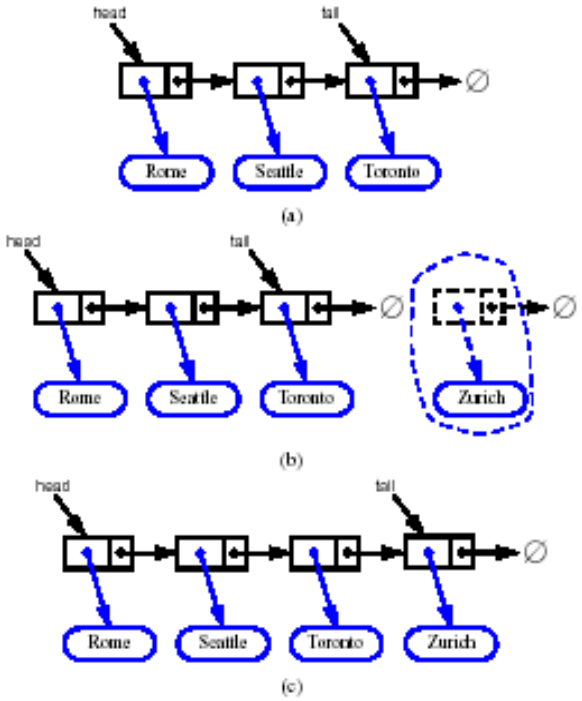
Removing at the Head

1. Update head to point to next node in the list
2. Allow garbage collector to reclaim the former first node



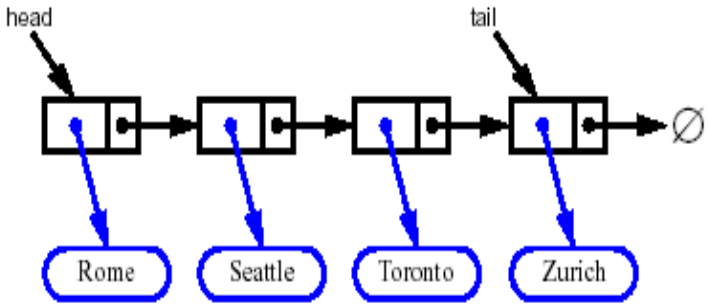
Inserting at the Tail

1. Allocate a new node
2. Insert new element
3. Have new node point to null
4. Have old last node point to new node
5. Update tail to point to new node



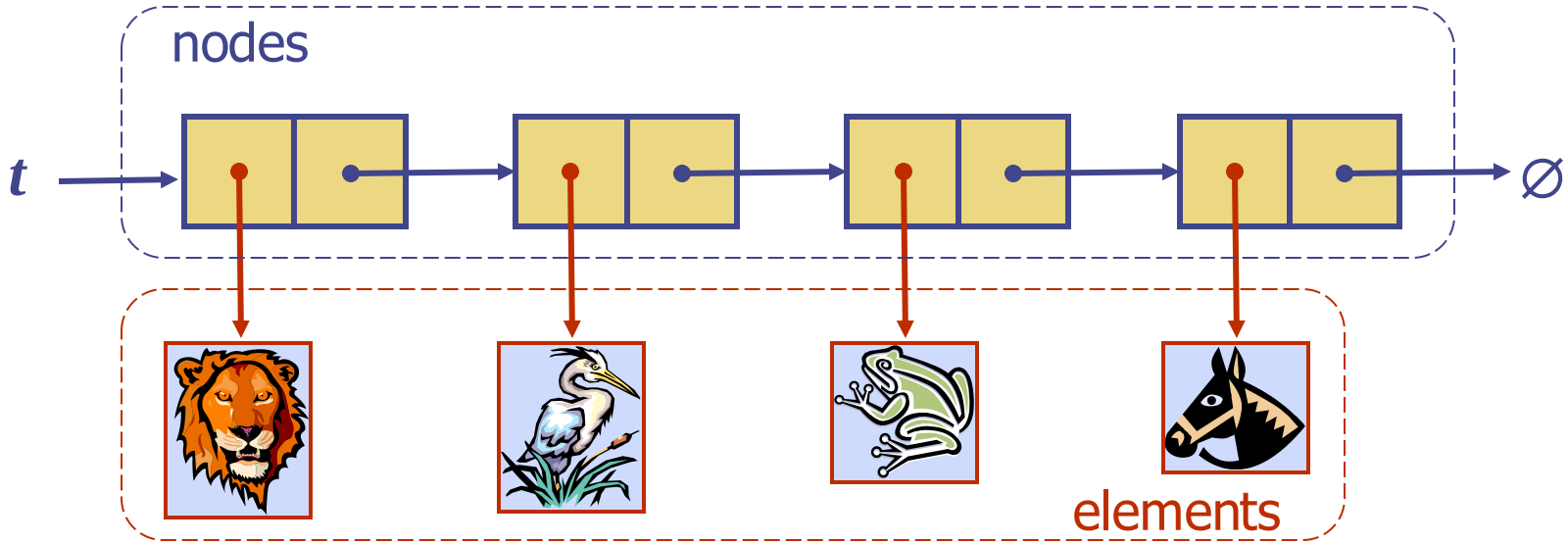
Removing at the Tail

- Removing at the tail of a singly linked list is not efficient!
- There is no constant-time way to update the tail to point to the previous node



Stack as a Linked List

- We can implement a stack with a singly linked list
- The top element is stored at the first node of the list
- The space used is $O(n)$ and each operation of the Stack ADT takes $O(1)$ time



Linked-List Stack in Python

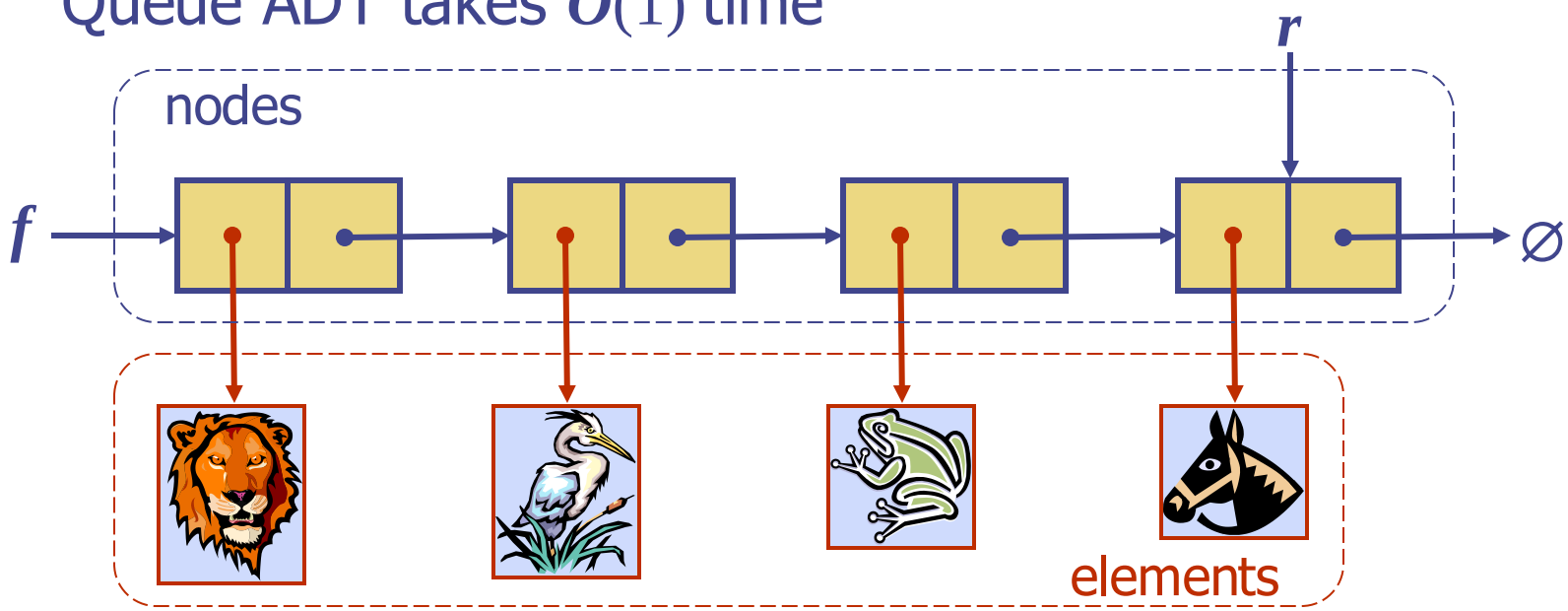
```
1  class LinkedStack:
2    """LIFO Stack implementation using a singly linked list for storage."""
3
4    #---------------------------- nested _Node class ----------------------------
5    class _Node:
6      """Lightweight, nonpublic class for storing a singly linked node."""
7      __slots__ = '_element', '_next'         # streamline memory usage
8
9      def __init__(self, element, next):      # initialize node's fields
10       self._element = element               # reference to user's element
11       self._next = next                     # reference to next node
12
13   #------------------------------ stack methods ------------------------------
14   def __init__(self):
15     """Create an empty stack."""
16     self._head = None                       # reference to the head node
17     self._size = 0                          # number of stack elements
18
19   def __len__(self):
20     """Return the number of elements in the stack."""
21     return self._size
22
```

```
23   def is_empty(self):
24     """Return True if the stack is empty."""
25     return self._size == 0
26
27   def push(self, e):
28     """Add element e to the top of the stack."""
29     self._head = self._Node(e, self._head)     # create and link a new node
30     self._size += 1
31
32   def top(self):
33     """Return (but do not remove) the element at the top of the stack.
34
35     Raise Empty exception if the stack is empty.
36     """
37     if self.is_empty():
38       raise Empty('Stack is empty')
39     return self._head._element                 # top of stack is at head of list
```

```
40   def pop(self):
41     """Remove and return the element from the top of the stack (i.e., LIFO).
42
43     Raise Empty exception if the stack is empty.
44     """
45     if self.is_empty():
46       raise Empty('Stack is empty')
47     answer = self._head._element
48     self._head = self._head._next              # bypass the former top node
49     self._size -= 1
50     return answer
```

Queue as a Linked List

- We can implement a queue with a singly linked list
  - The front element is stored at the first node
  - The rear element is stored at the last node
- The space used is $O(n)$ and each operation of the Queue ADT takes $O(1)$ time



Linked-List Queue in Python

```
1  class LinkedQueue:
2    """FIFO queue implementation using a singly linked list for storage."""
3
4    class _Node:
5      """Lightweight, nonpublic class for storing a singly linked node."""
6      (omitted here; identical to that of LinkedStack._Node)
7
8    def __init__(self):
9      """Create an empty queue."""
10     self._head = None
11     self._tail = None
12     self._size = 0                          # number of queue elements
13
14   def __len__(self):
15     """Return the number of elements in the queue."""
16     return self._size
17
18   def is_empty(self):
19     """Return True if the queue is empty."""
20     return self._size == 0
21
22   def first(self):
23     """Return (but do not remove) the element at the front of the queue."""
24     if self.is_empty():
25       raise Empty('Queue is empty')
26     return self._head._element              # front aligned with head of list
```

```
27   def dequeue(self):
28     """Remove and return the first element of the queue (i.e., FIFO).
29
30     Raise Empty exception if the queue is empty.
31     """
32     if self.is_empty():
33       raise Empty('Queue is empty')
34     answer = self._head._element
35     self._head = self._head._next
36     self._size -= 1
37     if self.is_empty():                     # special case as queue is empty
38       self._tail = None                     # removed head had been the tail
39     return answer
40
41   def enqueue(self, e):
42     """Add an element to the back of queue."""
43     newest = self._Node(e, None)            # node will be new tail node
44     if self.is_empty():
45       self._head = newest                   # special case: previously empty
46     else:
47       self._tail._next = newest
48     self._tail = newest                     # update reference to tail node
49     self._size += 1
```