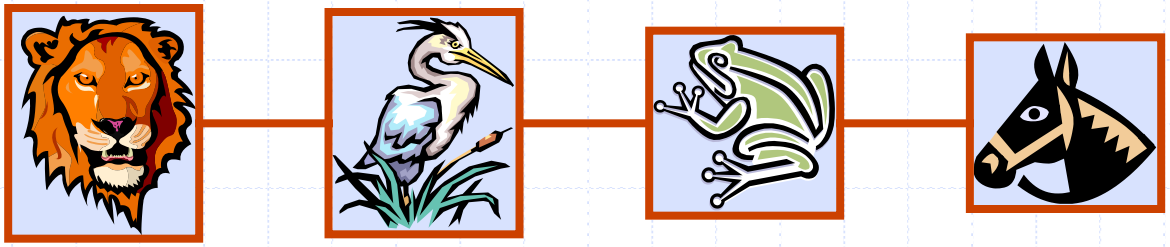
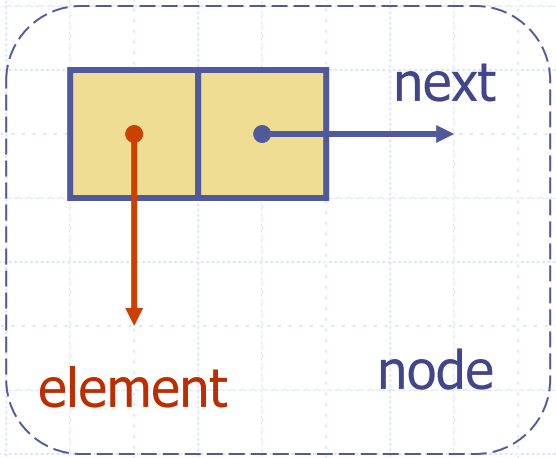
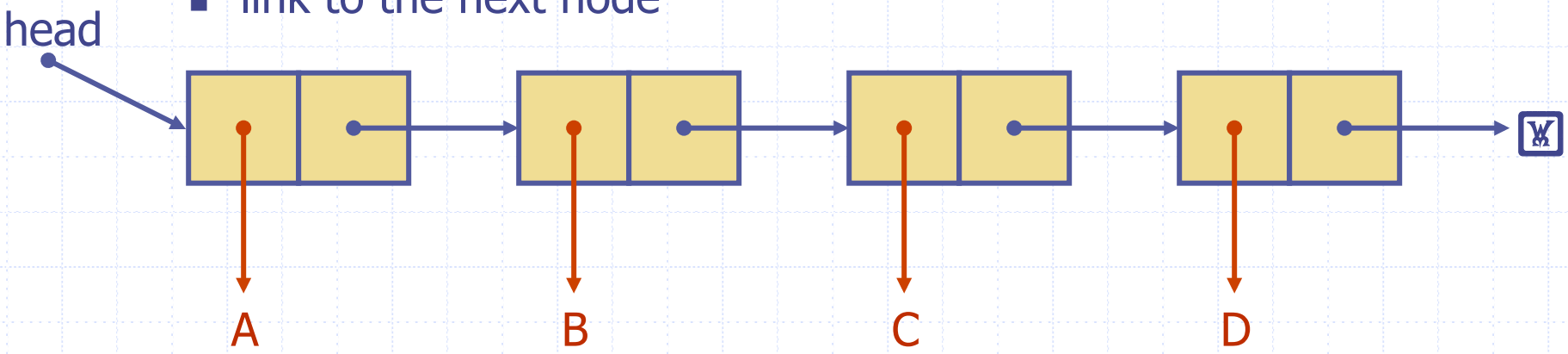
Linked Lists



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





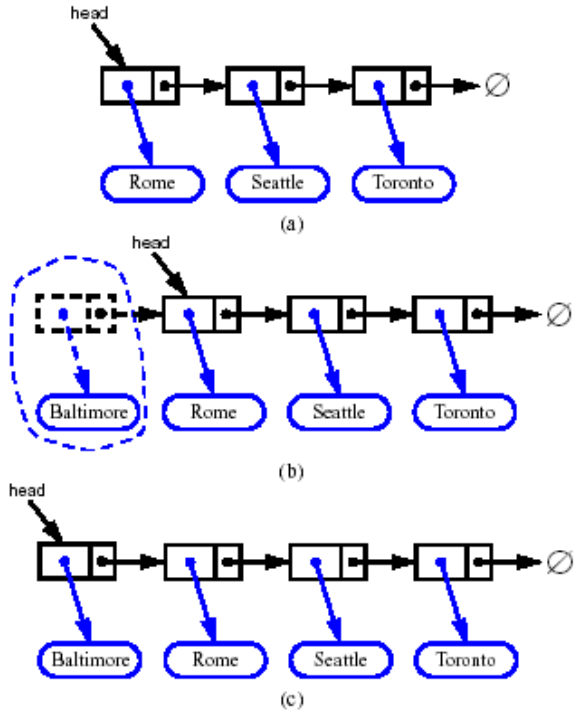
The SinglyLinkedNode Class

```
class SinglyLinkedNode:

    def __init__( self, element, next ):
        self.element = element
     self.next = next
```

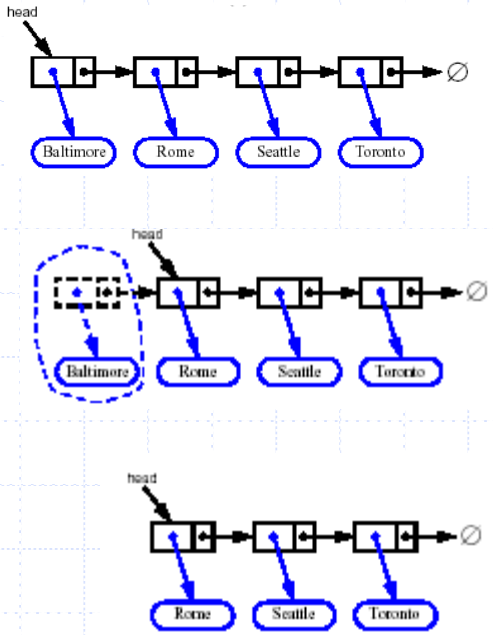
Inserting at the Head

1. Allocate a new node
2. Insert new element
3. Have new node point to old head
4. Update head to point to new node



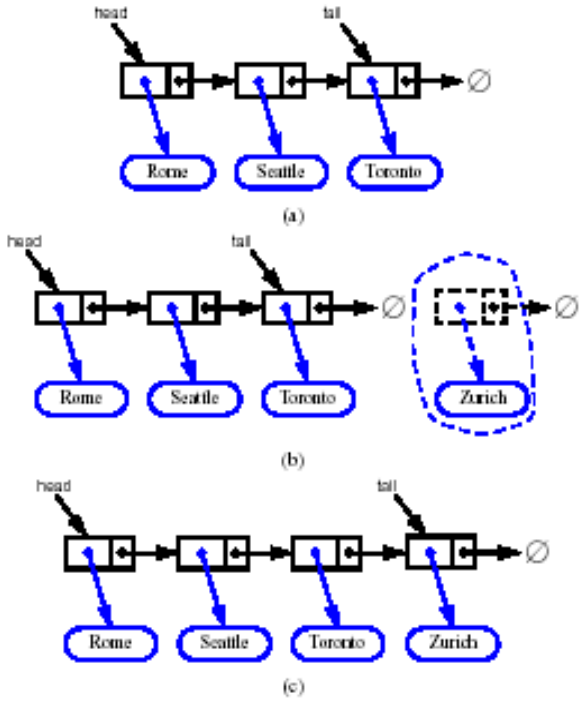
Removing at the Head

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



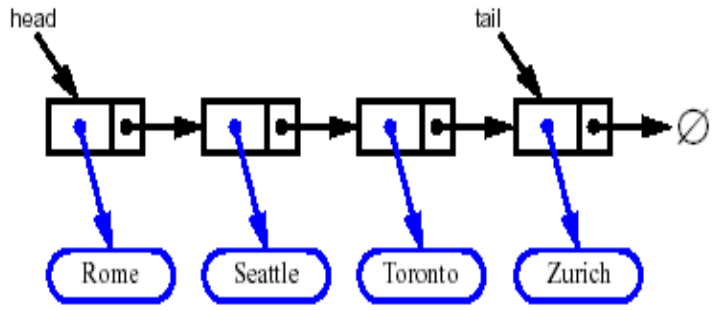
Inserting at the Tail

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



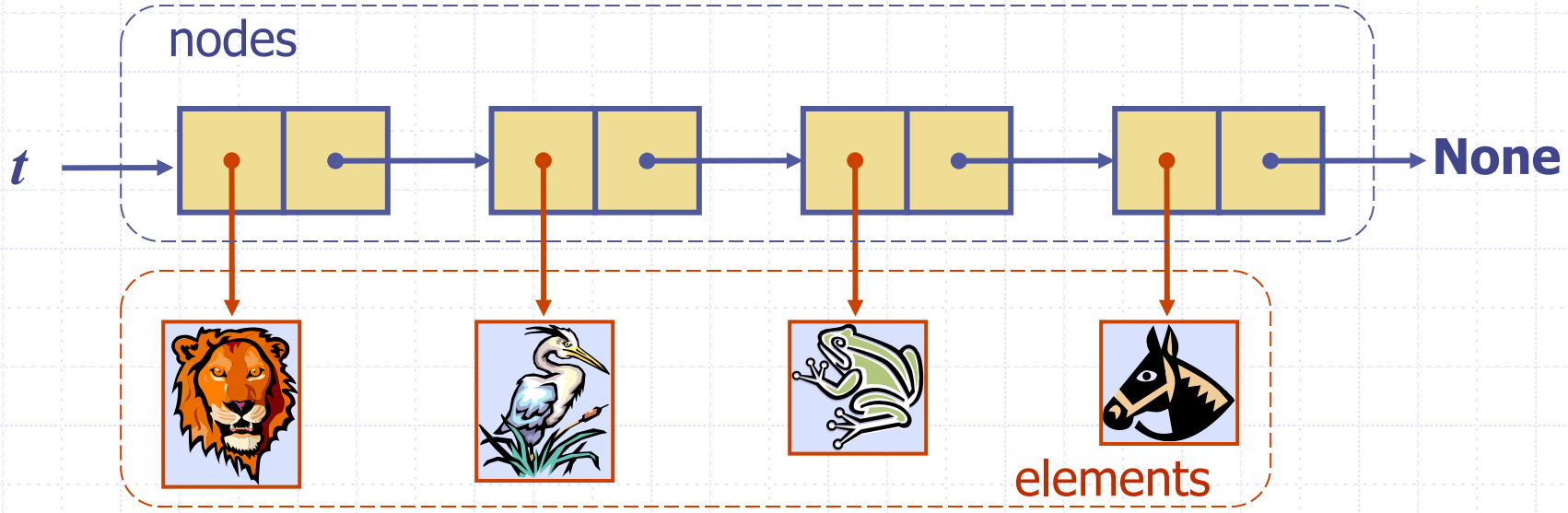
Removing at the Tail

- 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



SinglyLinkedListStack…

```
from SinglyLinkedList import SinglyLinkedList

class SinglyLinkedListStack:

    #implements the ADT Stack (Stack.py)
    #uses SinglyLinkedList (SinglyLinkedList.py)
    def __init__( self ):
        self._A = SinglyLinkedList()

    def __len__( self ):
        return len( self._A )

    def is_empty( self ):
        return len( self._A ) == 0
```

SinglyLinkedListStack

```
    def __str__( self ):
        if self._A.is_empty():
            return "[](size = 0)[top = None]"
        else:
            pp = str( self._A )
            pp += "[top = 0]"
            return pp

#push obj
def push( self, obj ):
    self._A.insert( obj )

#pop
def pop( self ):
    return self._A.remove( 1 )

#top
def top( self ):
    return self._A.first()
```

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