CSCE 121

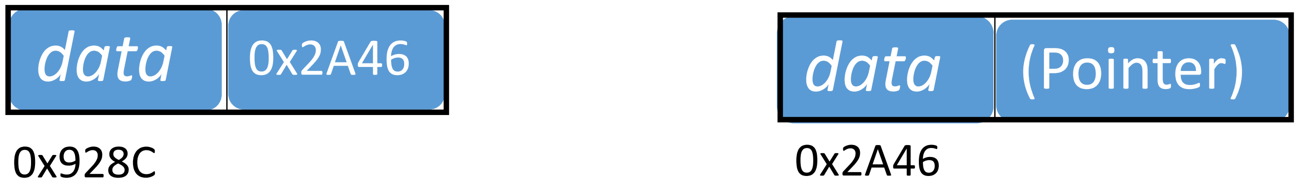
Linked Lists

Introduction

- In the most generic sense, a list is simply a finite sequence of elements
  - More appropriately, it is a storage structure for a finite sequence of elements
- An array is one example of a list
- BUT
  - Arrays tend to be inflexible
  - Usually need to know the maximum size that is ever needed
    - But that’s hard to guarantee
    - Can easily waste storage if not fully utilized

Nodes

- This is where we are going to really put pointers and dynamic memory to work
- Pointer variables are often part of a structure called a ***node***
- The node has two parts:
  - Information, or data, and
  - A pointer to another node



Linked Lists

- Another way to organize data items
  - Place them within objects—usually called nodes
  - Linked together into a “chain,” one after the other

Kind of like a freight train

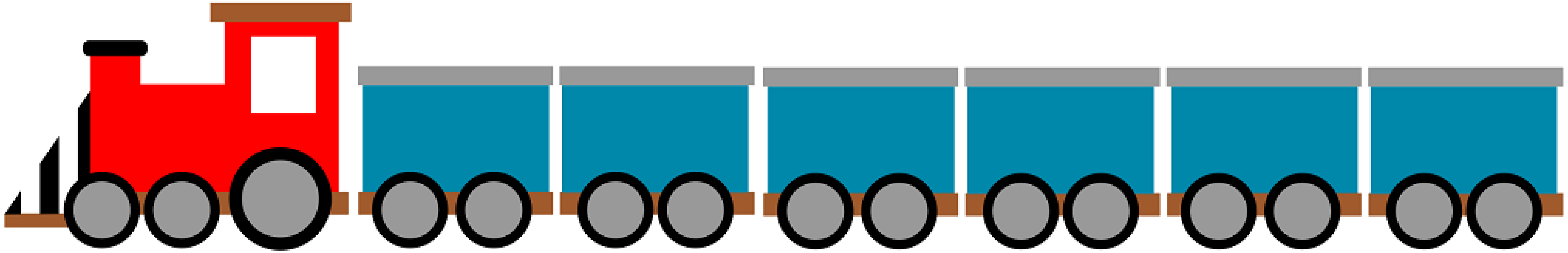
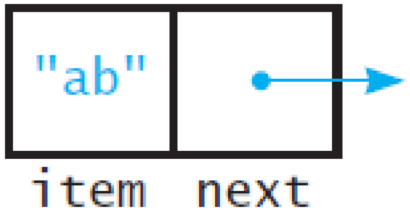


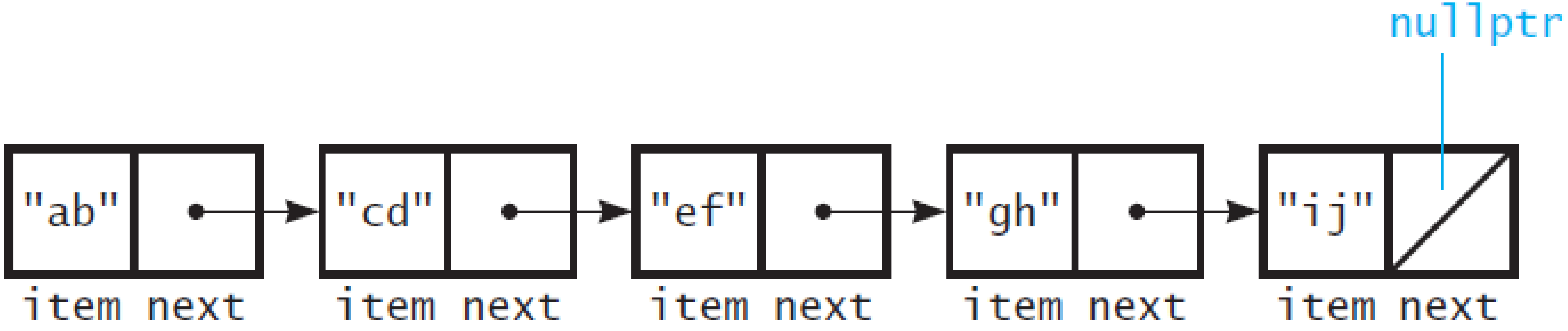
Image from freesvg.org

Linked Lists

A node, with a string as the data item



Several nodes linked together to form a linked list

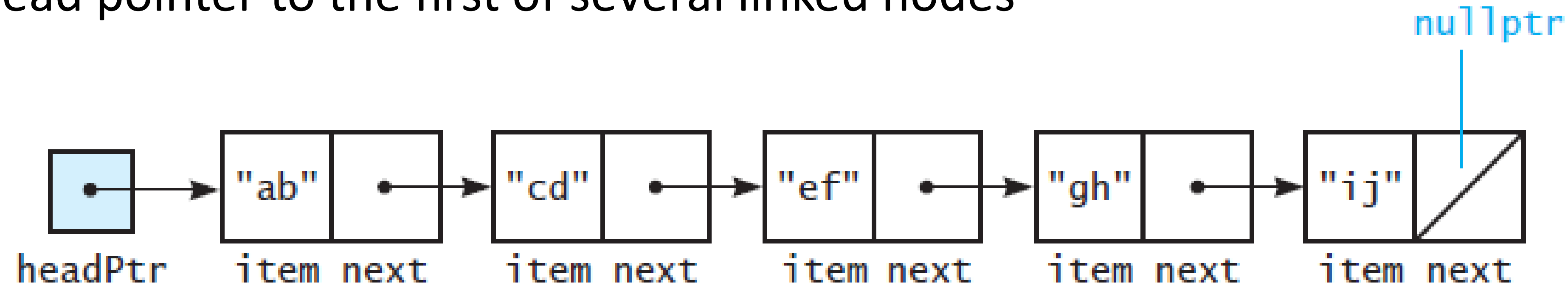


This and most subsequent list diagrams are modified from *Data Abstraction and Problem Solving with Java*, by Carrano and Prichard

Linked Lists

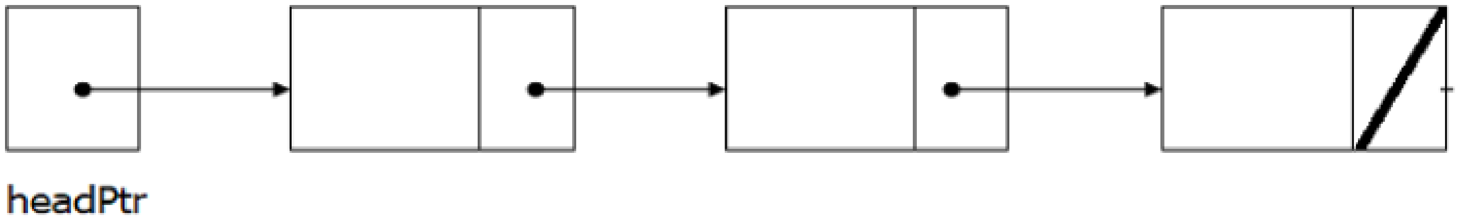
- There is also a special pointer, called the *head pointer*.
- It points to the first node in the list
- The last node in the list contains the null pointer in its next field.

A head pointer to the first of several linked nodes



Linked Lists vs. Arrays and Vectors

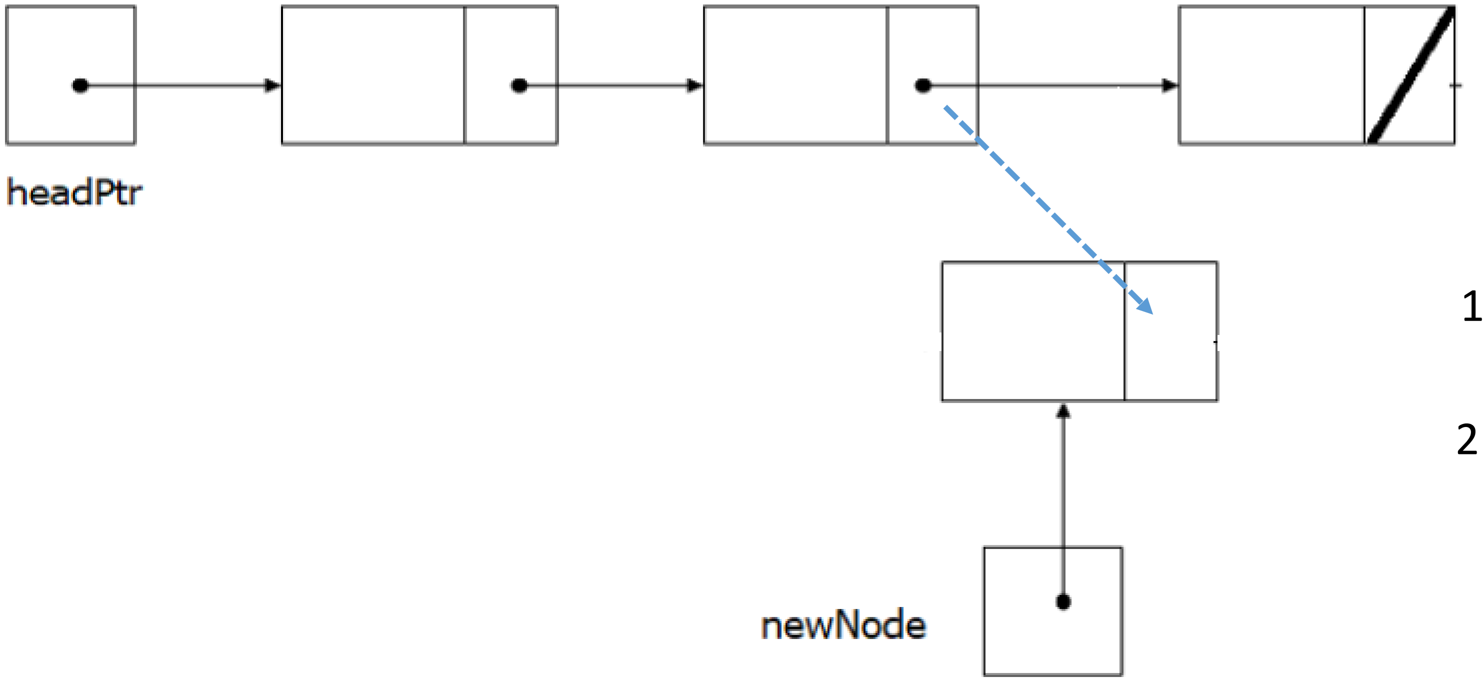
- Linked lists can grow and shrink as needed, unlike arrays, which have a fixed size
- Linked lists can insert a node between other nodes easily



Linked Lists vs. Arrays and Vectors (2)

- Linked lists can grow and shrink as needed, unlike arrays, which have a fixed size

- Linked lists can insert a node between other nodes easily

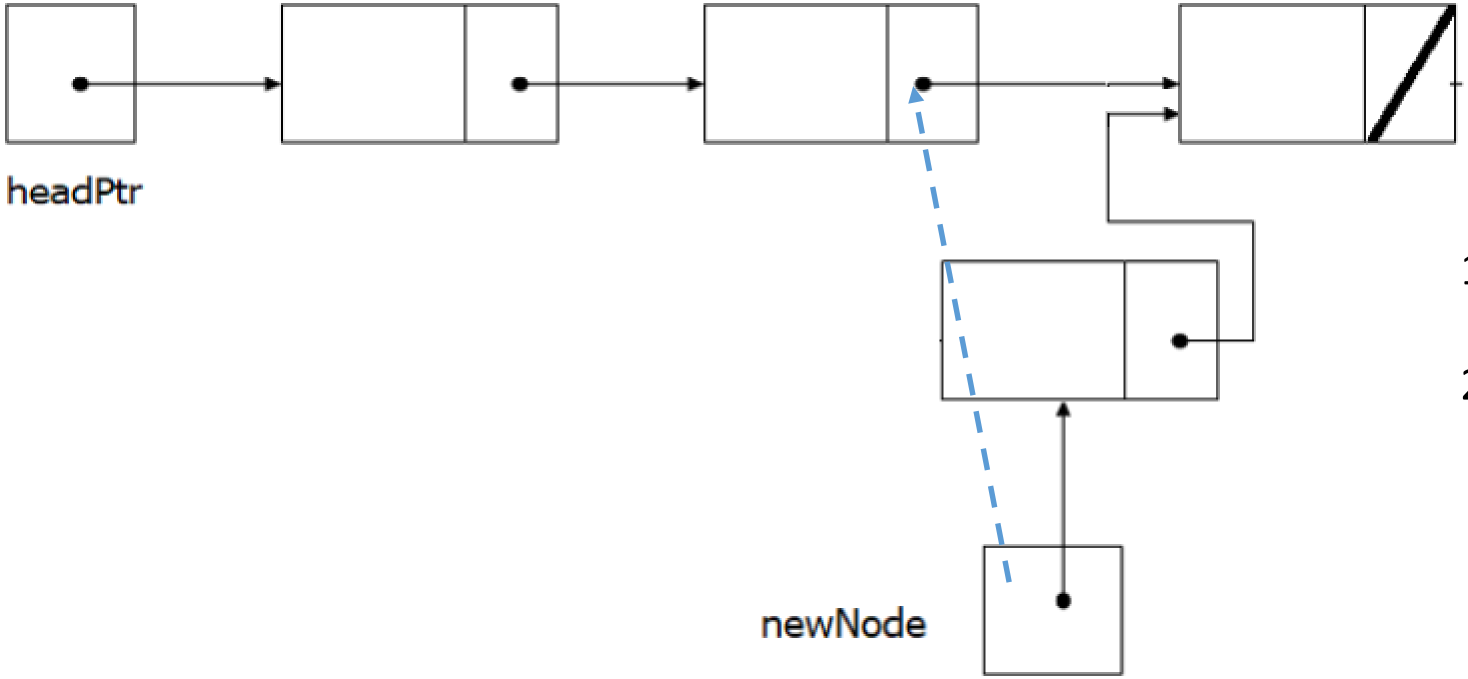


1. Create a new node with the data you want
2. Link the new node to the rest of the list

Linked Lists vs. Arrays and Vectors (3)

- Linked lists can grow and shrink as needed, unlike arrays, which have a fixed size

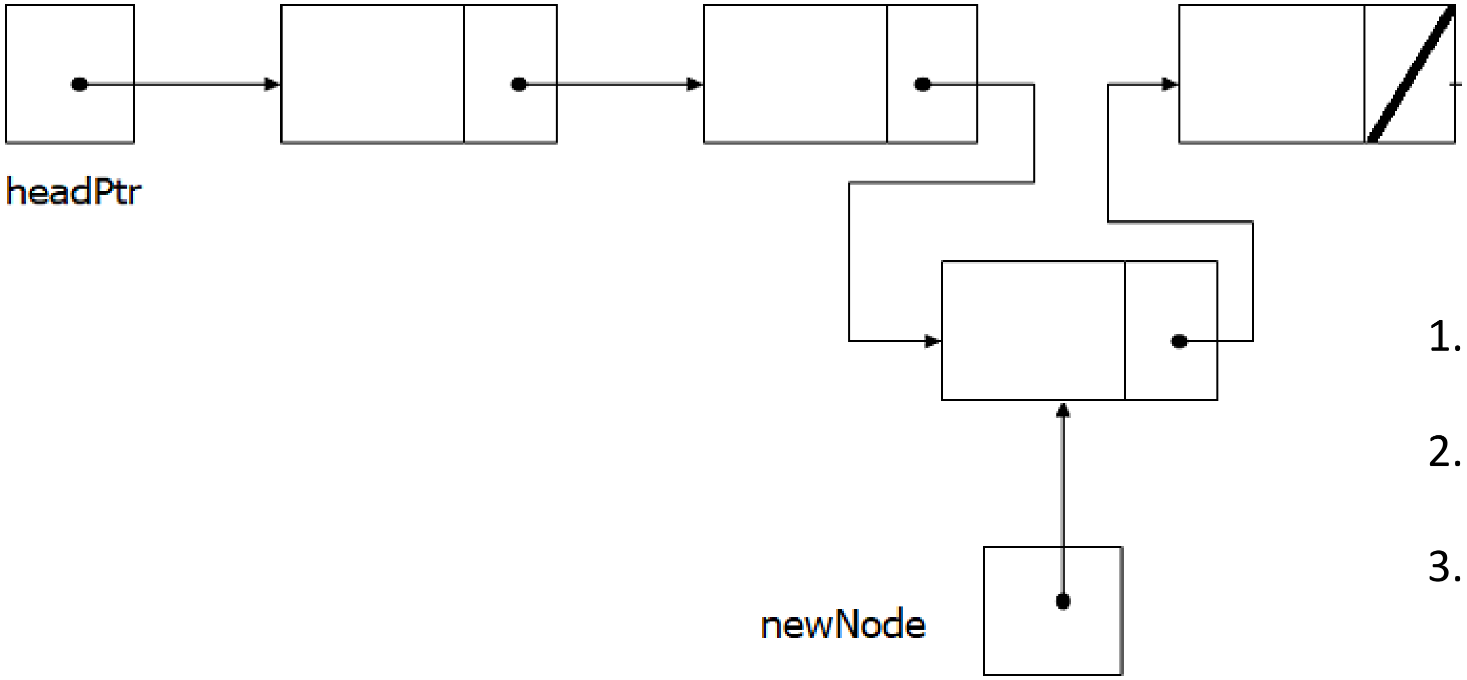
- Linked lists can insert a node between other nodes easily



1. Create a new node with the data you want
2. Link the new node to the rest of the list
3. Link the previous node to the new node

Linked Lists vs. Arrays and Vectors (4)

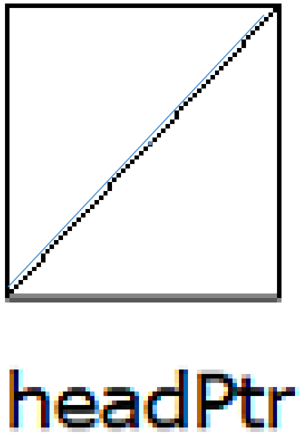
- Linked lists can grow and shrink as needed, unlike arrays, which have a fixed size
- Linked lists can insert a node between other nodes easily



1. Create a new node with the data you want
2. Link the new node to the rest of the list
3. Link the previous node to the new node

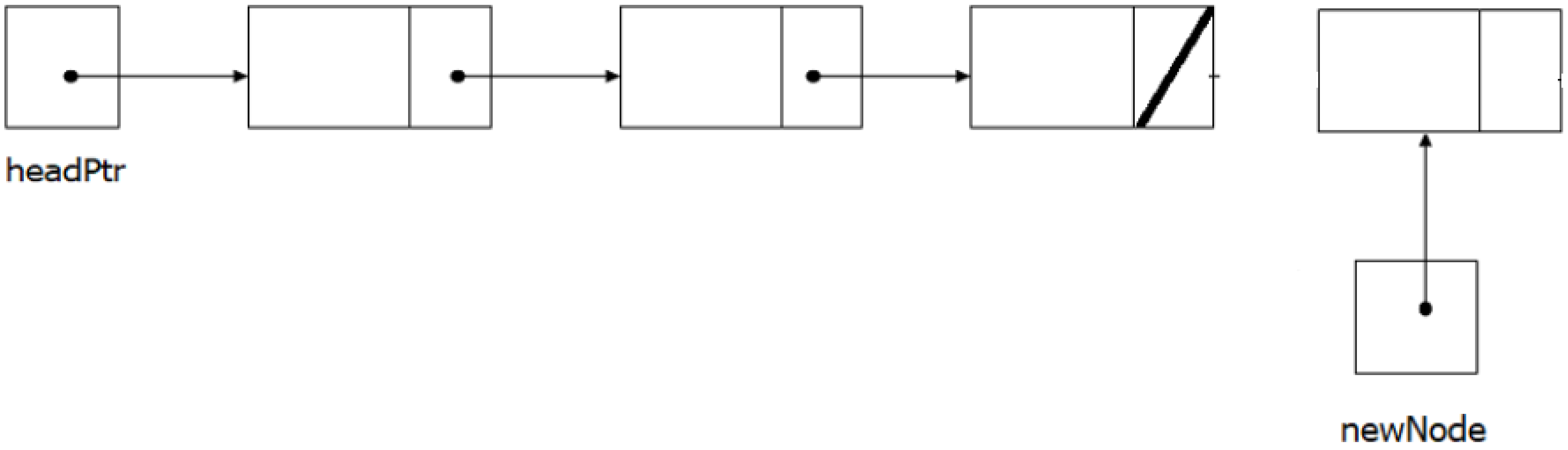
Empty List

- If a list currently contains 0 nodes, the list still exists
- It is the empty list
- In this case the head pointer is the nullptr



Special Cases?

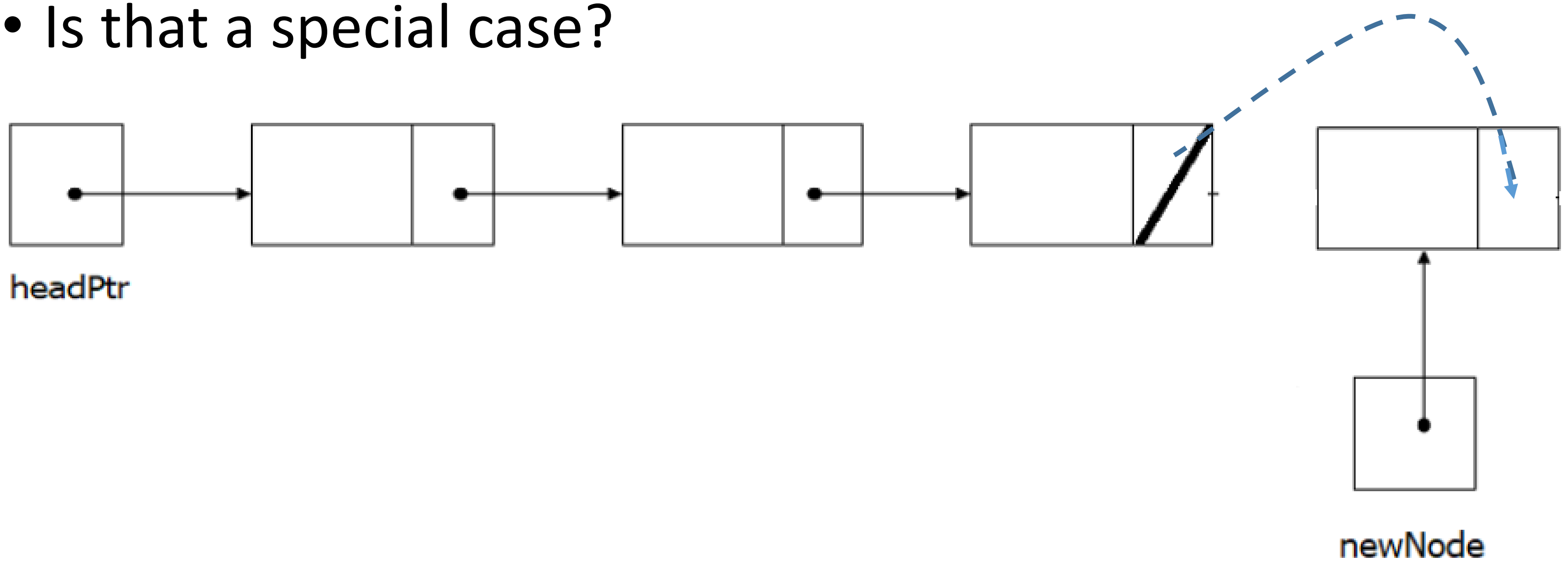
- What about inserting a node at the end of a linked list?
- Is that a special case?



1. Create a new node with the data you want

Special Cases? (2)

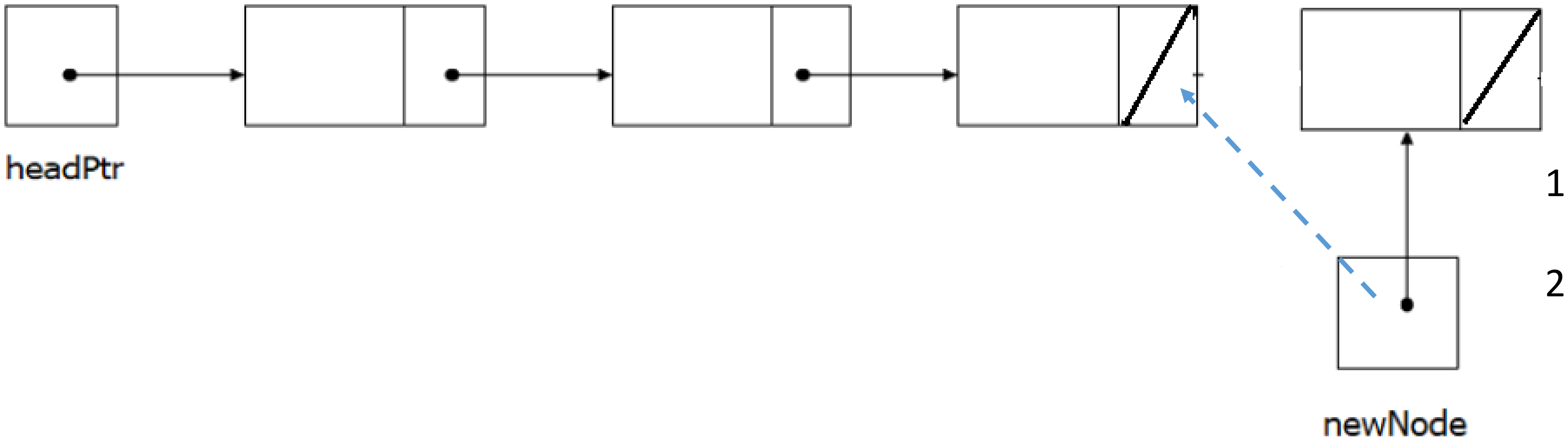
- What about inserting a node at the end of a linked list?
- Is that a special case?



1. Create a new node with the data you want
2. Link the new node to the rest of the list (which is empty.

Special Cases? (3)

- What about inserting a node at the end of a linked list?
- Is that a special case?

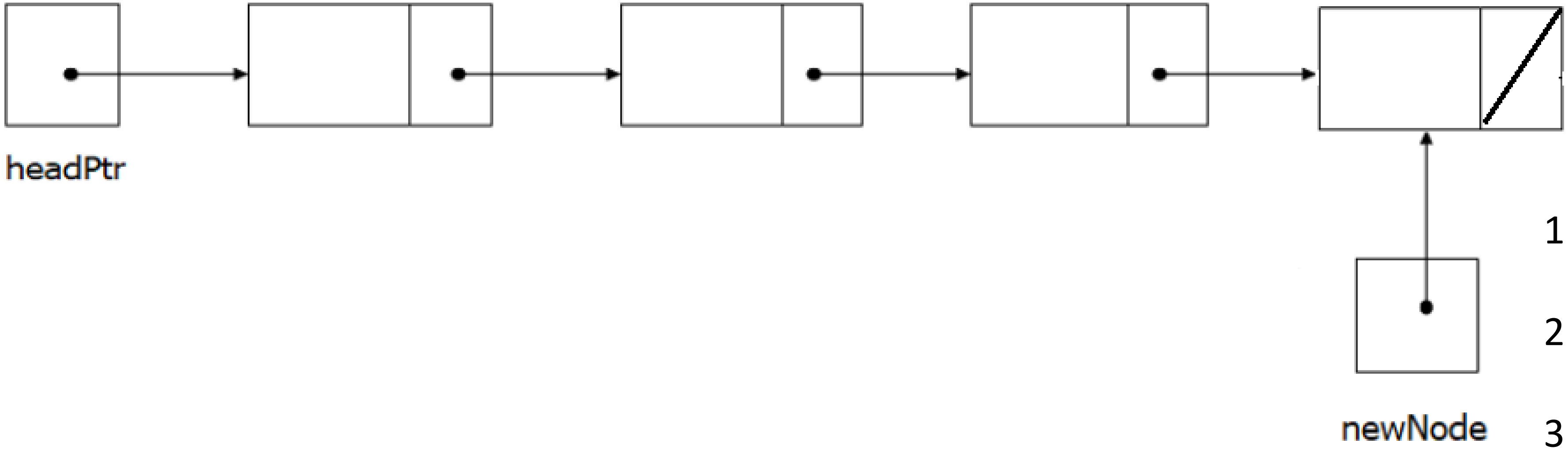


1. Create a new node with the data you want
2. Link the new node to the rest of the list

Special Cases? (4)

- What about inserting a node at the end of a linked list?
- Is that a special case?

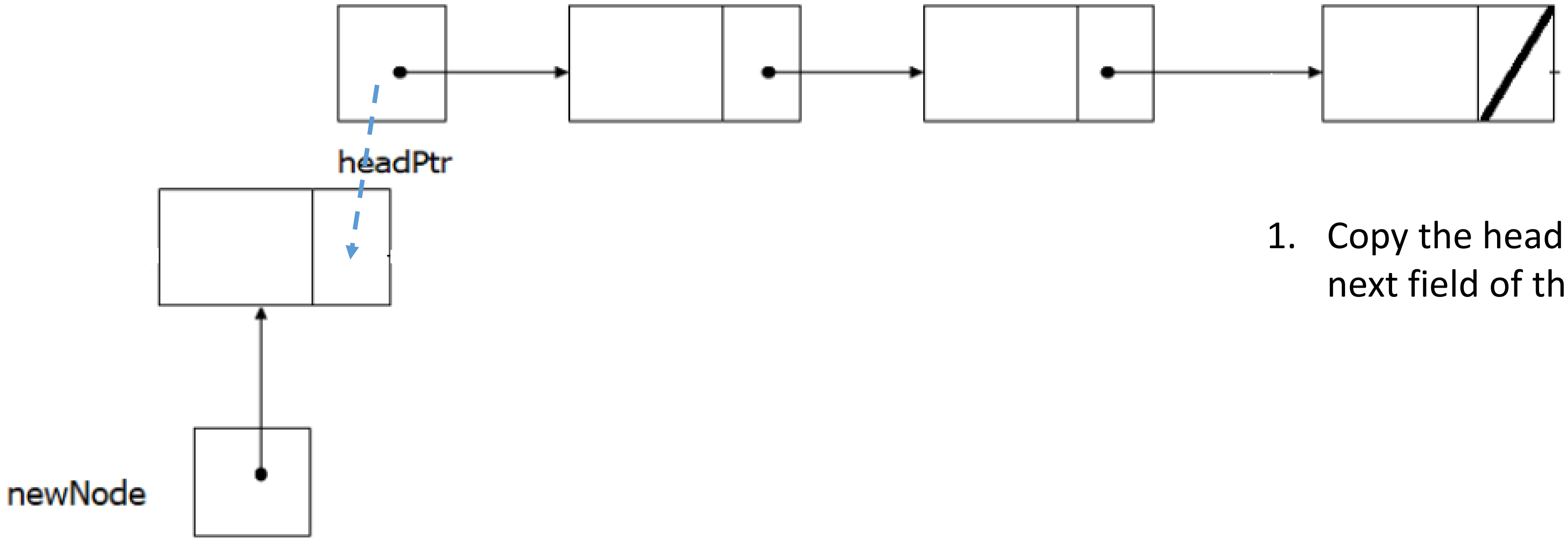
So, no, it's not a special case.



1. Create a new node with the data you want
2. Link the new node to the rest of the list
3. Link the previous node to the new node

Special Cases? (5)

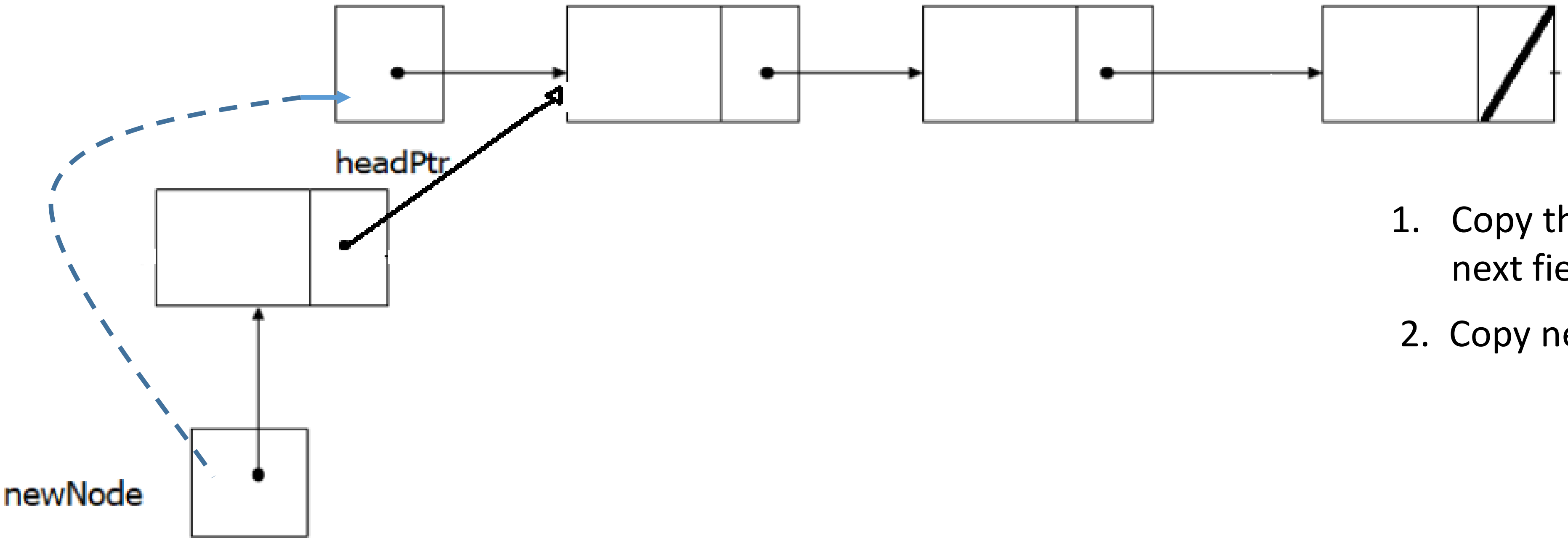
- What about inserting a node at the beginning of a linked list?
- Is that a special case?



1. Copy the head pointer into the next field of the new node

Special Cases? (6)

- What about inserting a node at the beginning of a linked list?
- Is that a special case?

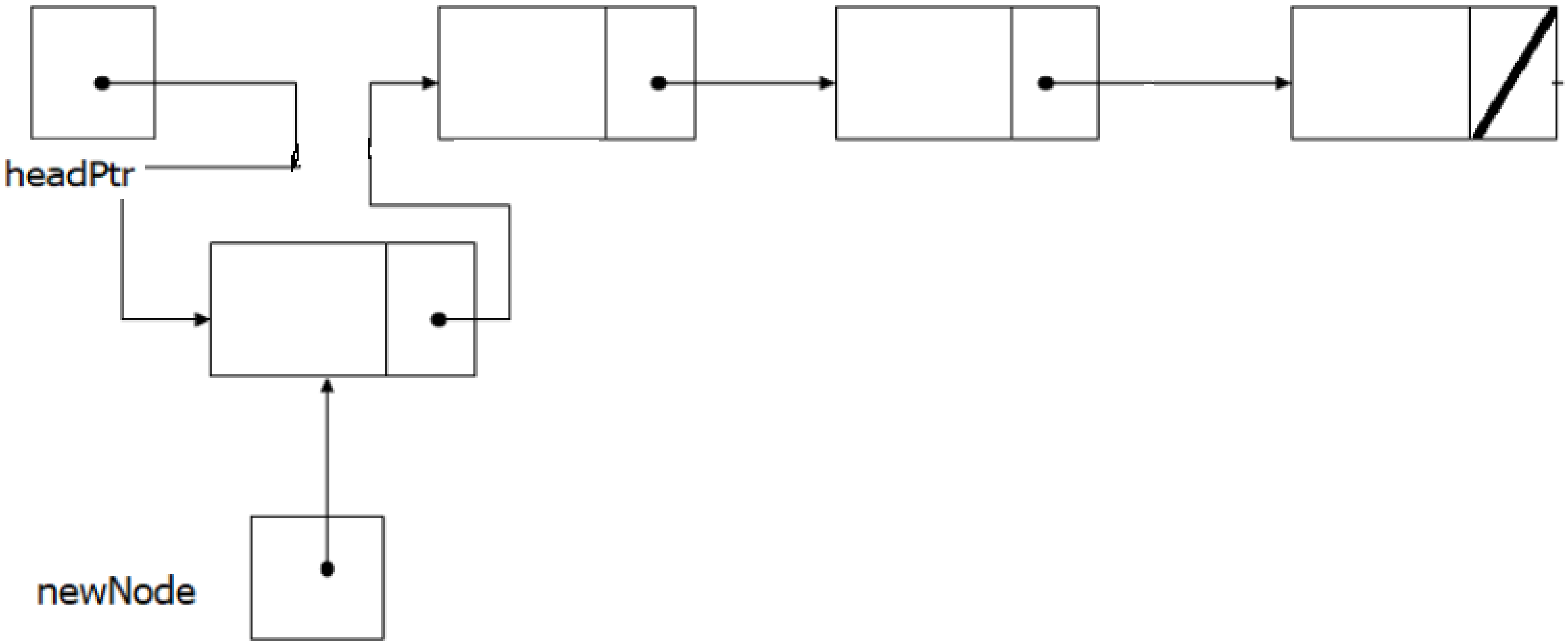


1. Copy the head pointer into the next field of the new node
2. Copy newNode into the headPtr

Special Cases? (7)

- What about inserting a node at the beginning of a linked list?
- Is that a special case?

So, yes, it's a special case.

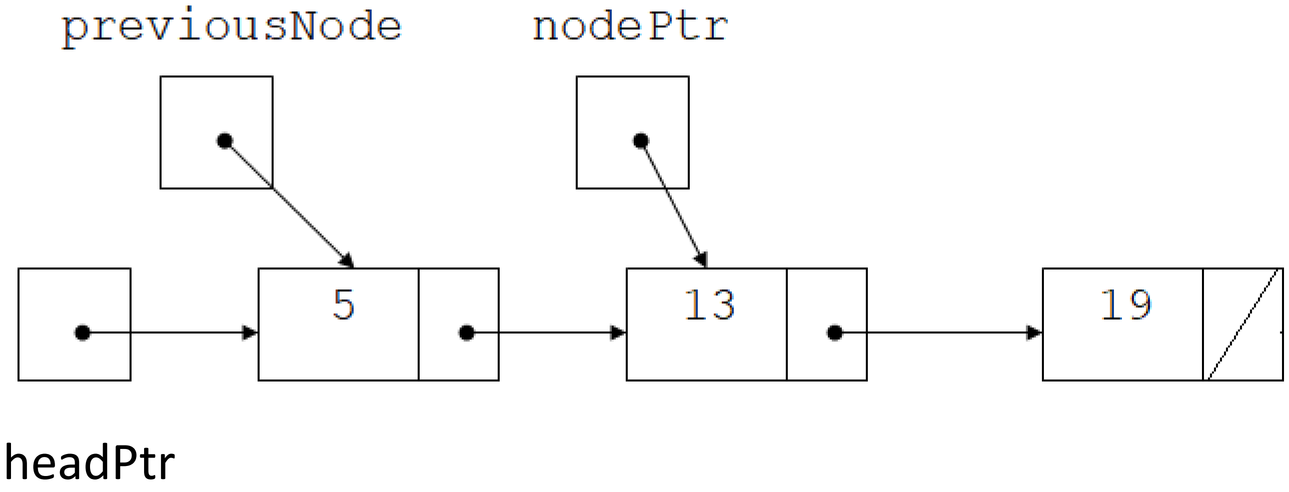


1. Copy the head pointer into the next field of the new node
2. Copy newNode into the headPtr
3. We have to change the headPtr

Deleting a Node

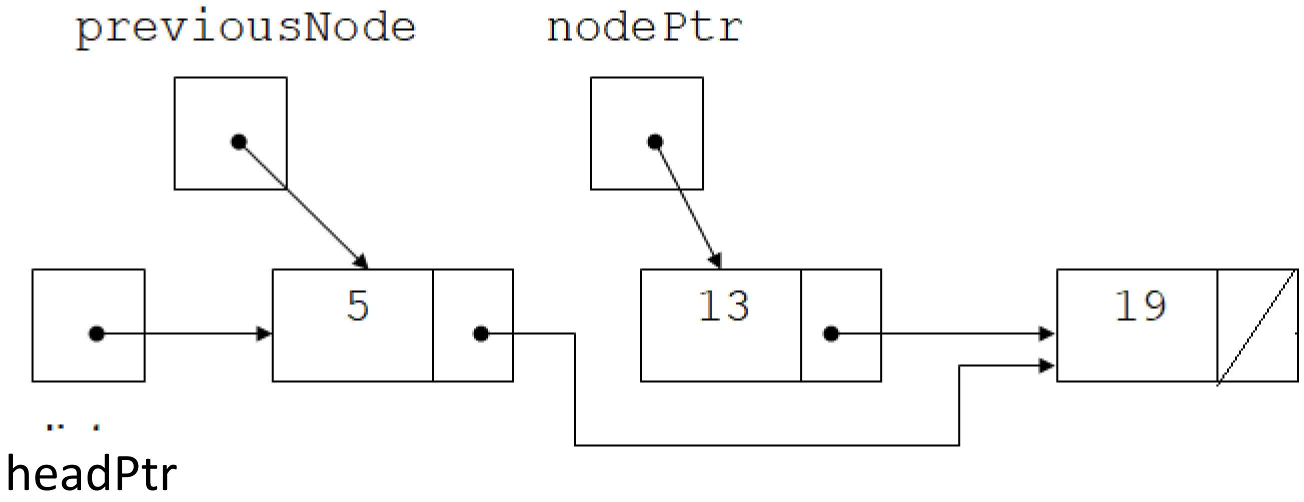
- Used to remove a node from a linked list
- If list uses dynamic memory, then delete node from memory
- Requires two pointers: one to locate the node to be deleted, one to point to the node before the node to be deleted

Deleting a Node (2)



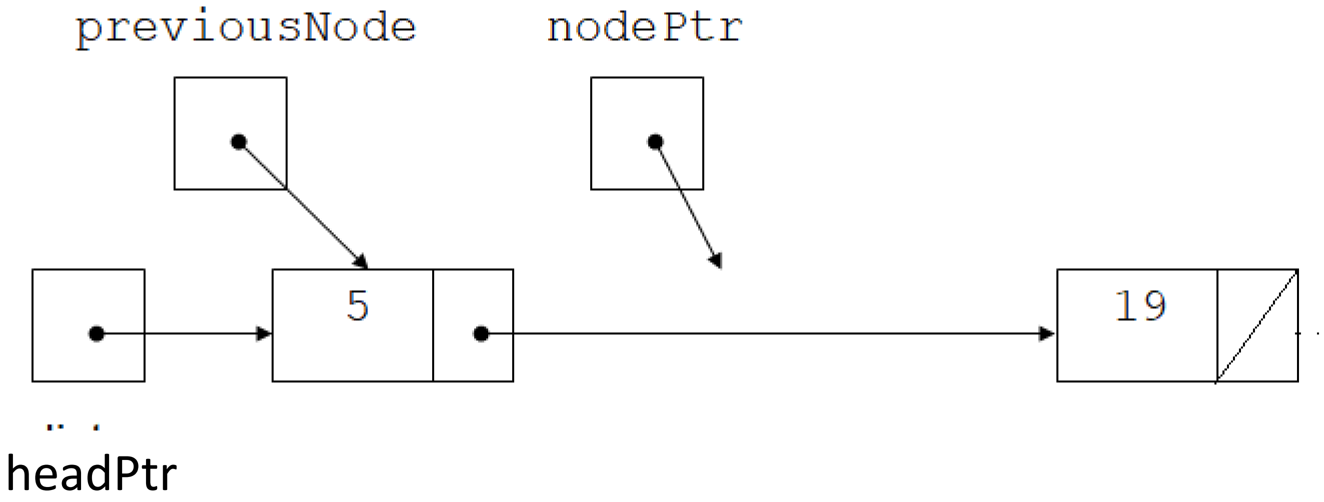
Locating the node containing `13`

Deleting a Node (3)



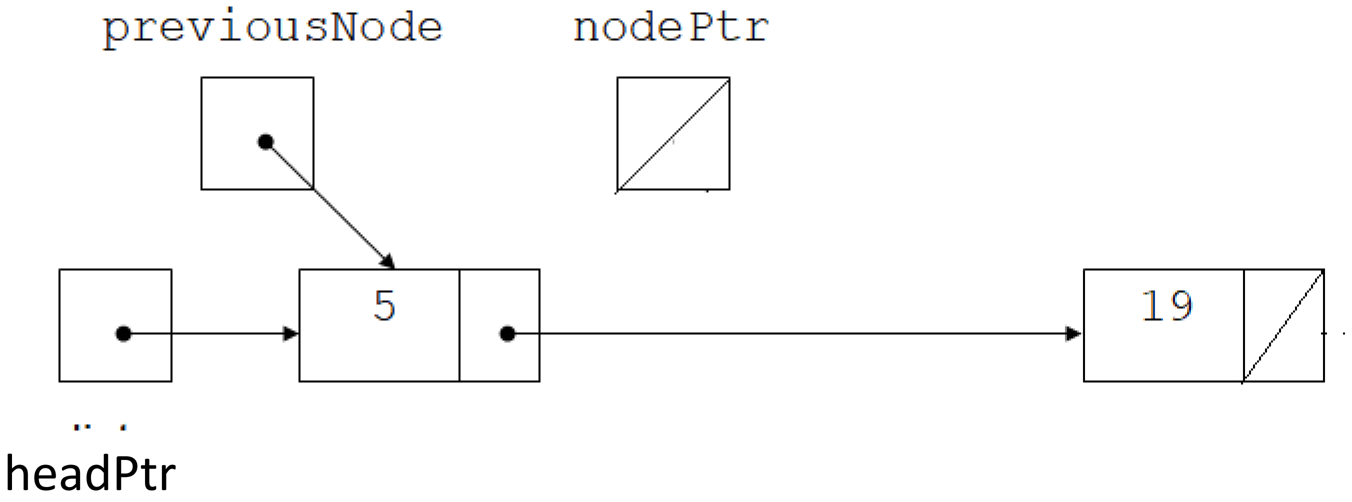
Adjusting pointer around the node to be deleted

Deleting a Node (4)



Linked list after deleting the node containing 13

Deleting a Node

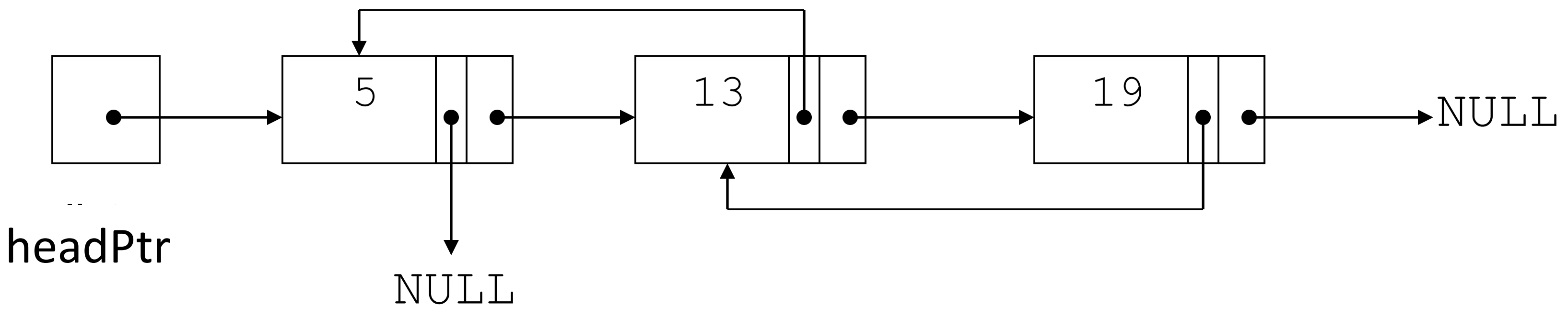


Linked list after deleting the node containing 13

Variations of the Linked List

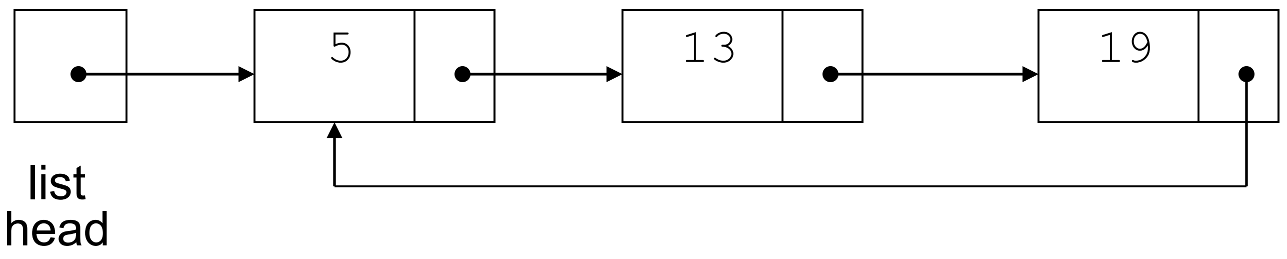
Variations of the Linked List (1)

- Other linked list organizations:
  - doubly-linked list: each node contains two pointers: one to the next node in the list, one to the previous node in the list



Variations of the Linked List (2)

- Other linked list organizations:
  - circular linked list: the last node in the list points back to the first node in the list, not to `nullPtr`
  - Note that the head can move


5
13
19
list head