FAIRMONT STATE UNIVERSITY

Linked Lists

COMP 2270 – Data Structures

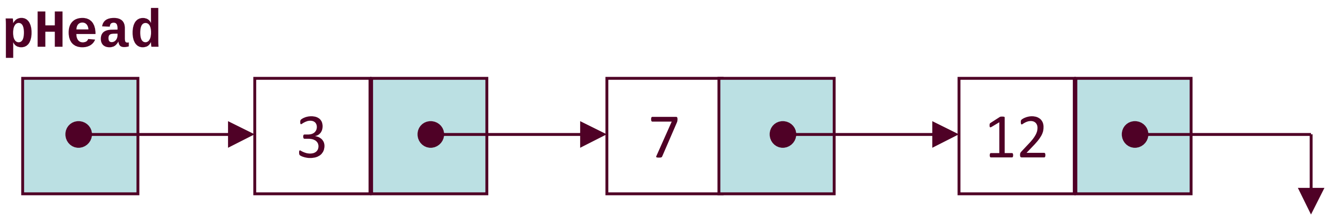
Fall 2014

Dr. Mahmood Hossain

What is a List?

- A list is a varying-length, linear collection of homogeneous elements
- Linear means each list element (except the first) has a unique predecessor, and each element (except the last) has a unique successor
- A list can be implemented by
  - an array stored in consecutive memory locations
  - a linked list not necessarily stored in consecutive memory locations

Linked List of Integers



The **Node** Type

- Each **Node** should store
  - data item(s)
  - the address of the next **Node**

```
struct Node
{
   int data;    // data stored at this node
   Node *next; // pointer to next node
};
```

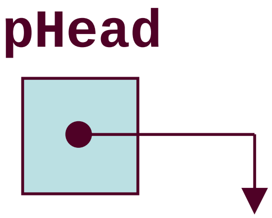
  - you may use a **Node** class as well

Linked List Operations

- Creating a linked list
- Inserting a new node into a linked list
- Deleting a node from a linked list
- Traversing a linked list

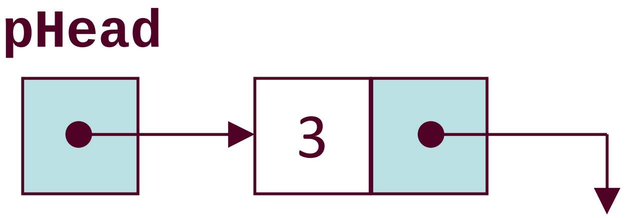
Creating a Linked List

- Declare a head pointer:



Creating a Linked List ...

- Create the first node with data value of 3:

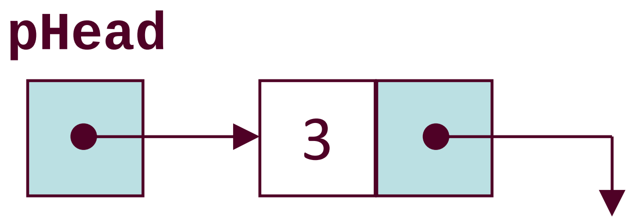


Inserting Nodes

- Insertion can be done in different ways:
  - inserting at the beginning
  - inserting somewhere in the middle
  - inserting at the end

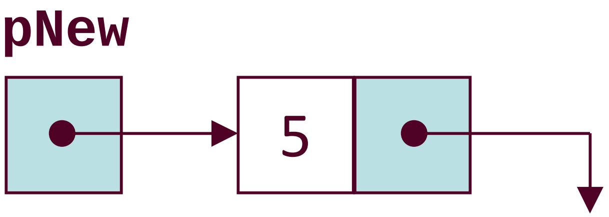
Inserting at the Beginning

- Three steps:
  - create up the new node
  - make the new node point to the first node
  - make the head pointer point to the new first node
- Example - Insert a node with a data value of 5 at the beginning of the following list:



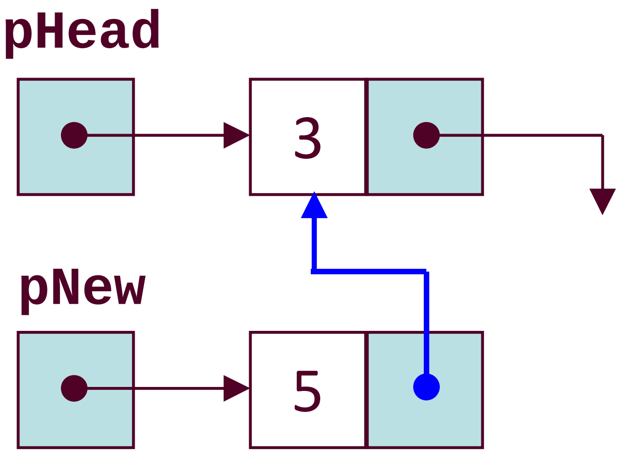
Example – Step 1

- Create up a new node with a data value of 5



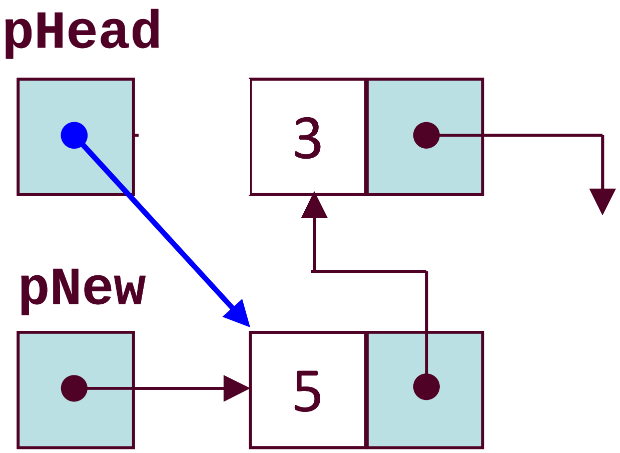
Example – Step 2

- Make the new node point to the first node

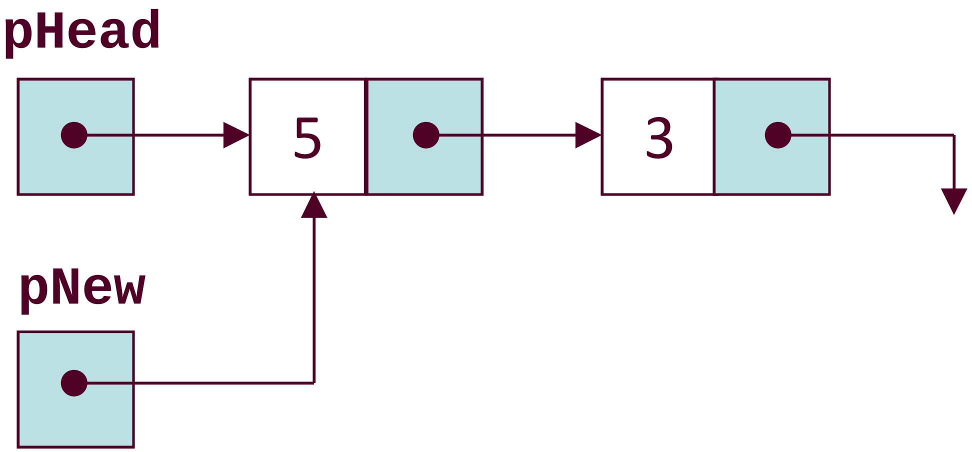


Example – Step 3

- Make the head pointer point to the new first node



Example – After Step 3



What should you do with **pNew**?

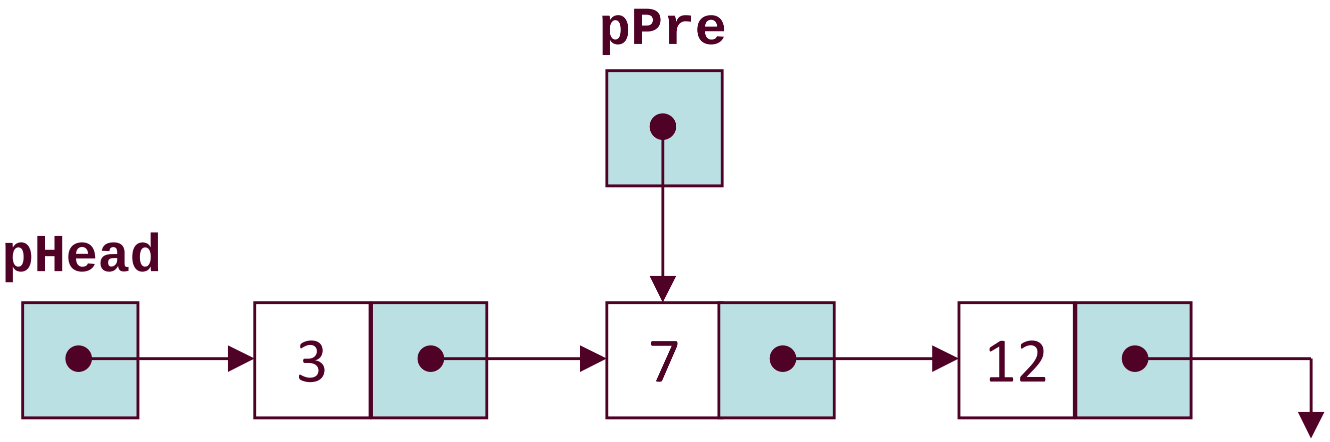
Write the Complete Code

Inserting in the Middle

- You need to have a pointer `pPre` pointing to the node after which insertion is to be made
- Three steps:
  - create up the new node
  - make the new node point to the one the node pointed to by `pPre` was pointing to
  - make the node pointed to by `pPre` point to the new node

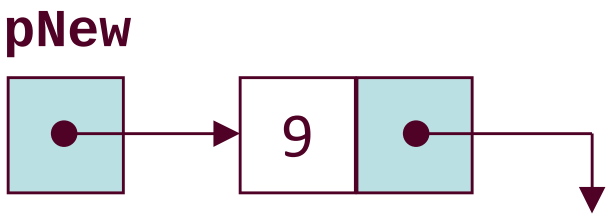
Example

- Insert a node with a data value of 9 after the node pointed to by **`pPre`**



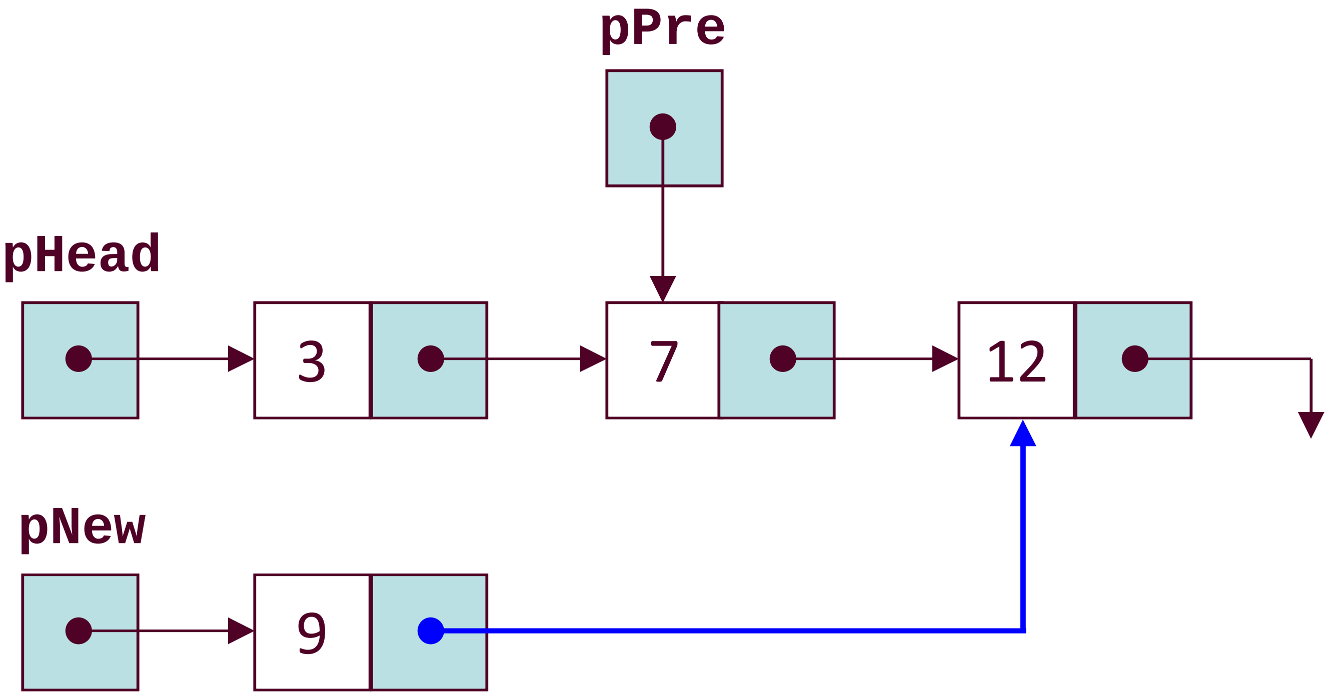
Example – Step 1

- Create up a new node with a data value of 9



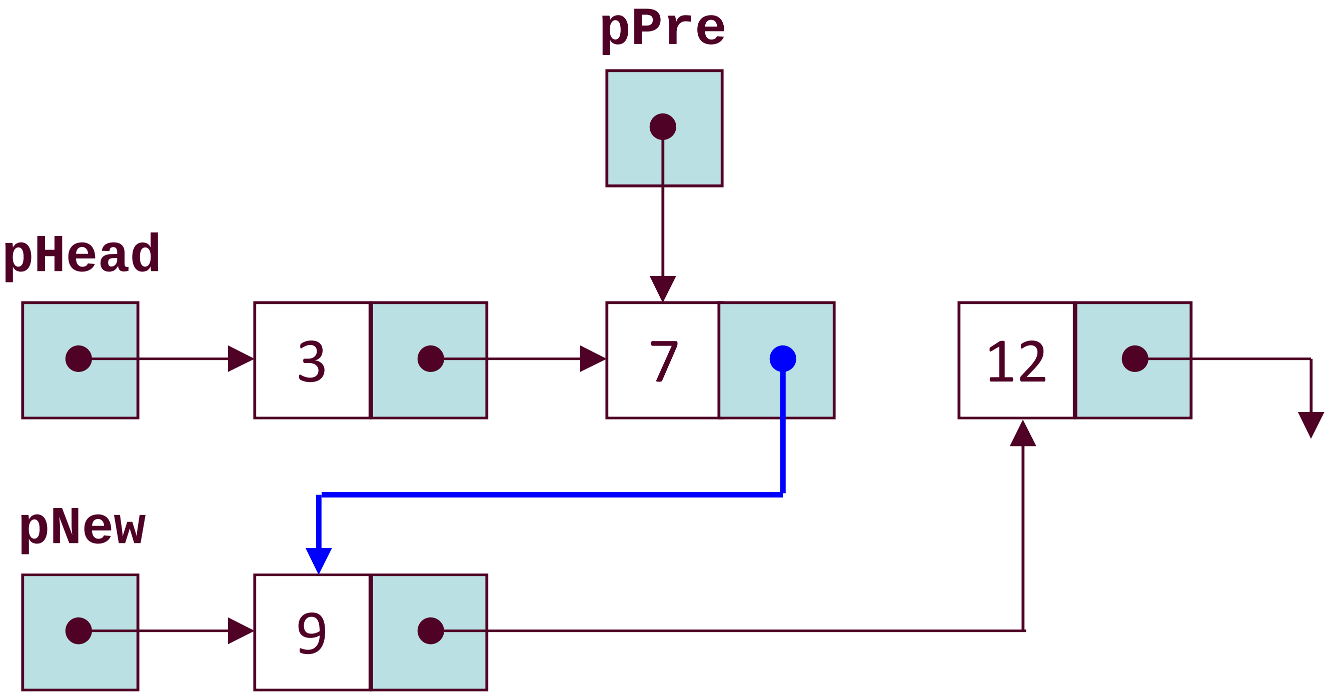
Example – Step 2

- Make the new node point to the node where the node pointed to by **`pPre`** was pointing to

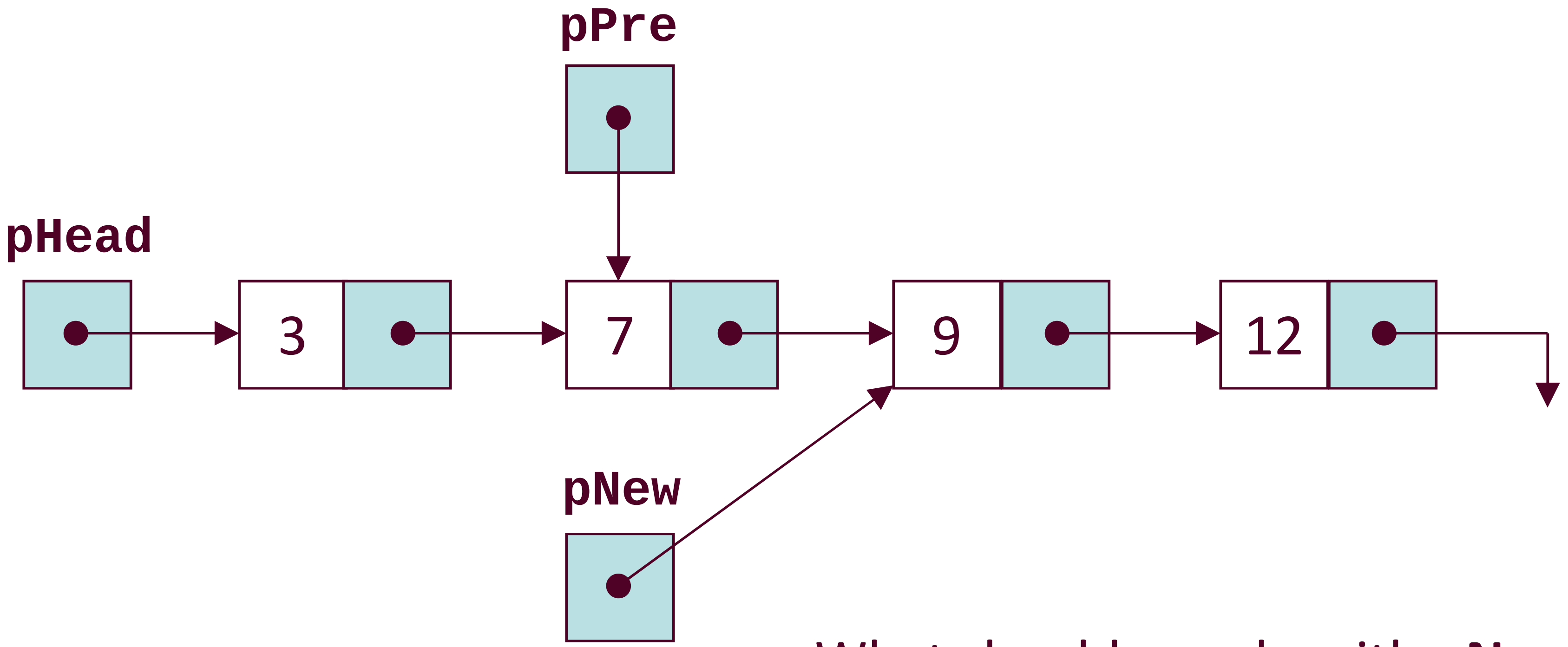


Example – Step 3

- Make the node pointed to by **`pPre`** point to the new node



Example – After Step 3



What should you do with **pNew**?

Inserting at the End

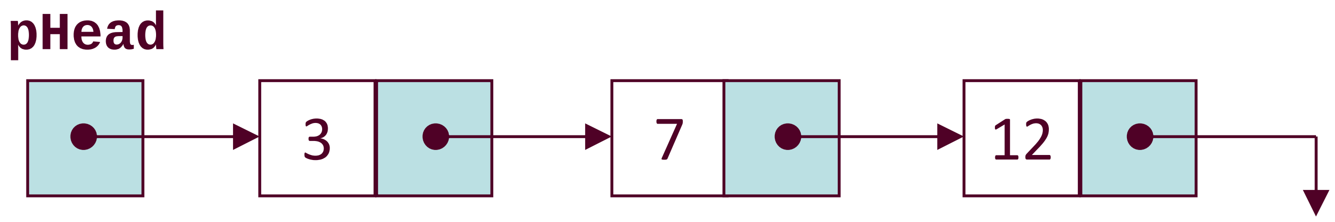
- You need to have a pointer **`pPre`** that is pointing to the last node
- What will be the steps?

Deleting Nodes

- Deletion can be done in different ways:
  - deleting the first node
  - deleting a target node
  - deleting the last node

Deleting the First Node

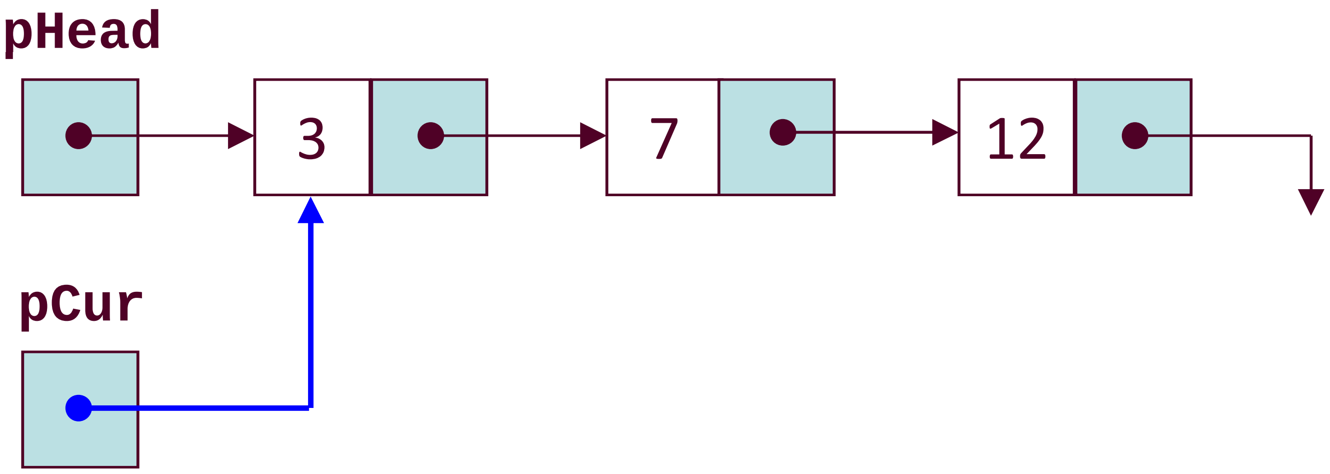
- Delete the first node from the following list



- Three steps:
  - store the pointer to the first node temporarily
  - make the head point to the second node
  - deallocate the space occupied by the original first node

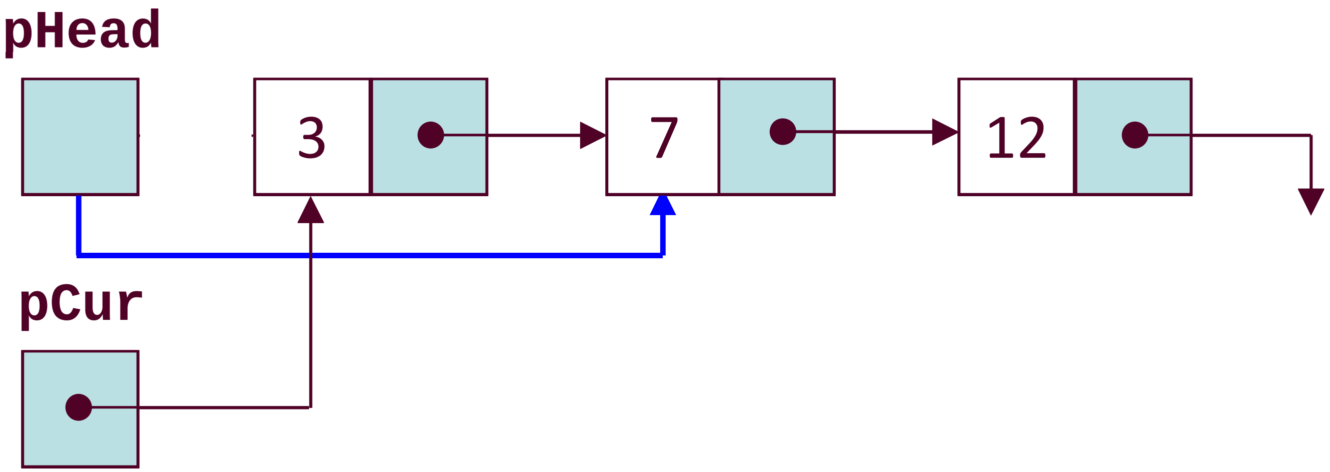
Example – Step 1

- Store the pointer to the first node temporarily



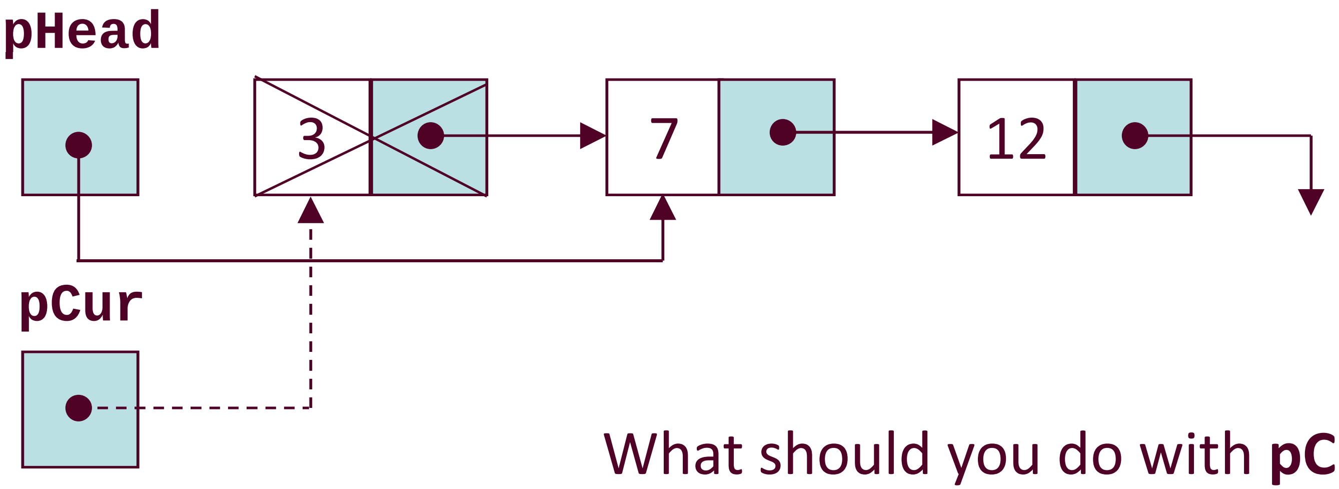
Example – Step 2

- Make the head point to the second node



Example – Step 3

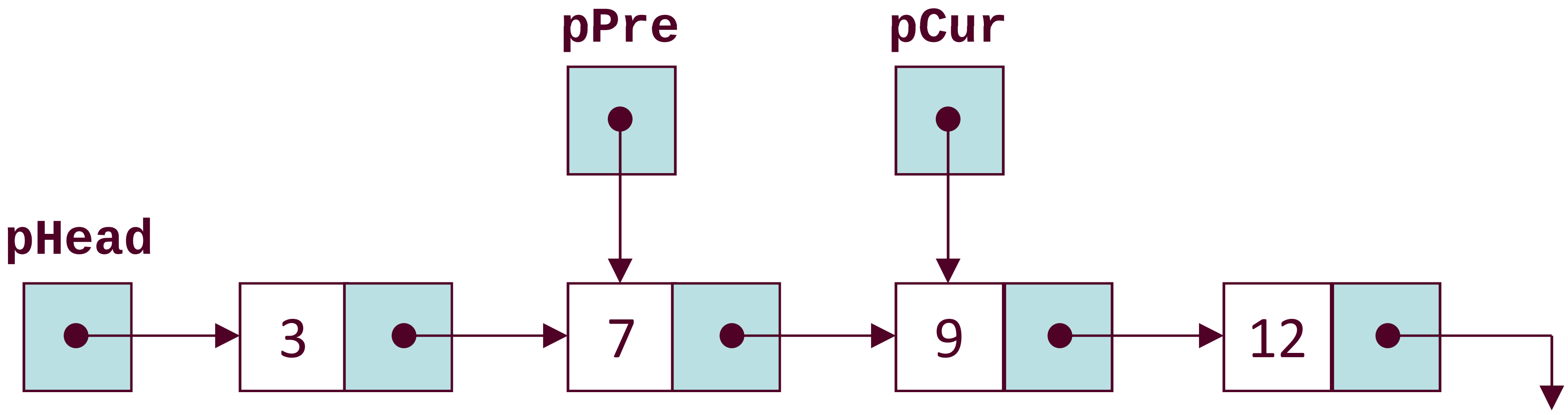
- Deallocate the space occupied by the original first node



What should you do with **pCur**?

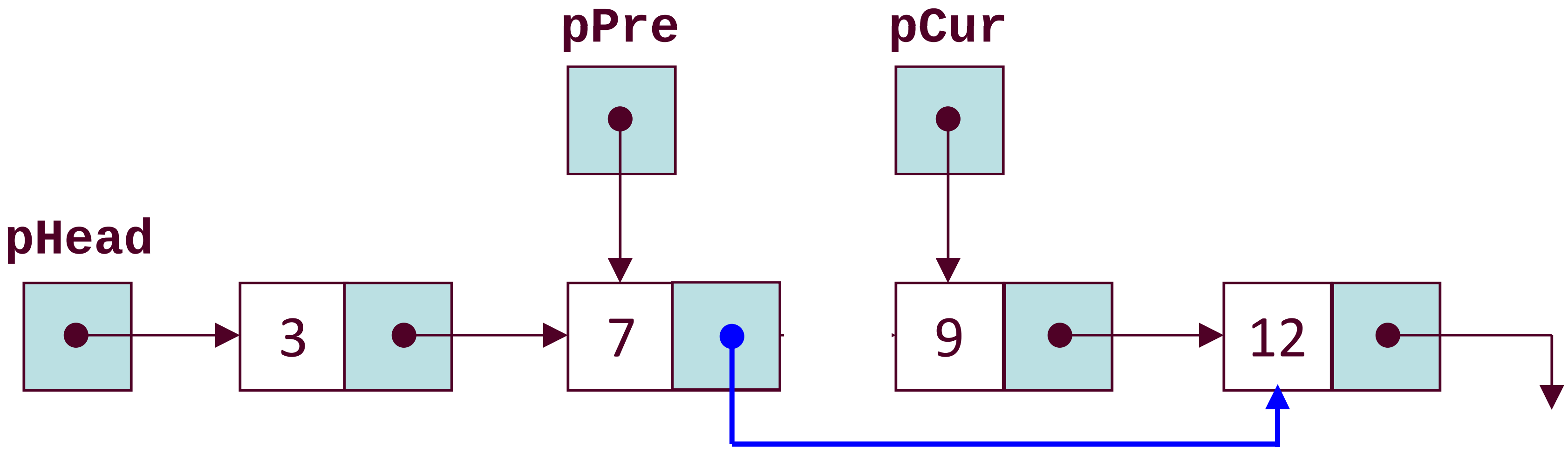
Deleting a Target Node

- Delete the node with the value of 9
- What pointers will you need?



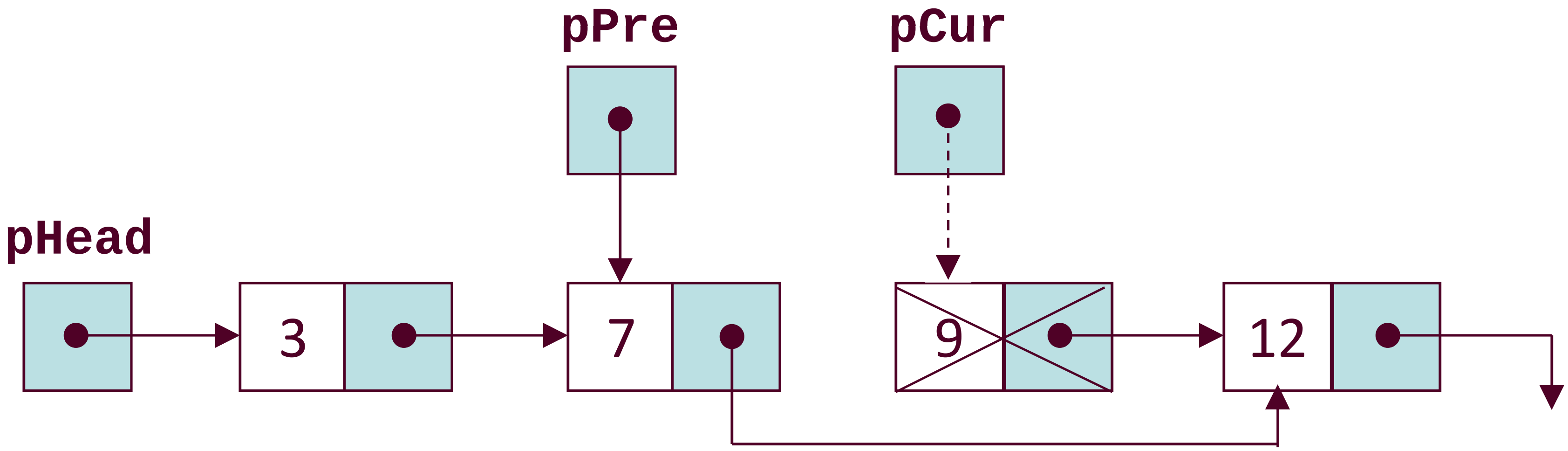
Example – Step 1

- Make the “node before the one to be deleted” point to the “node after the one to be deleted”



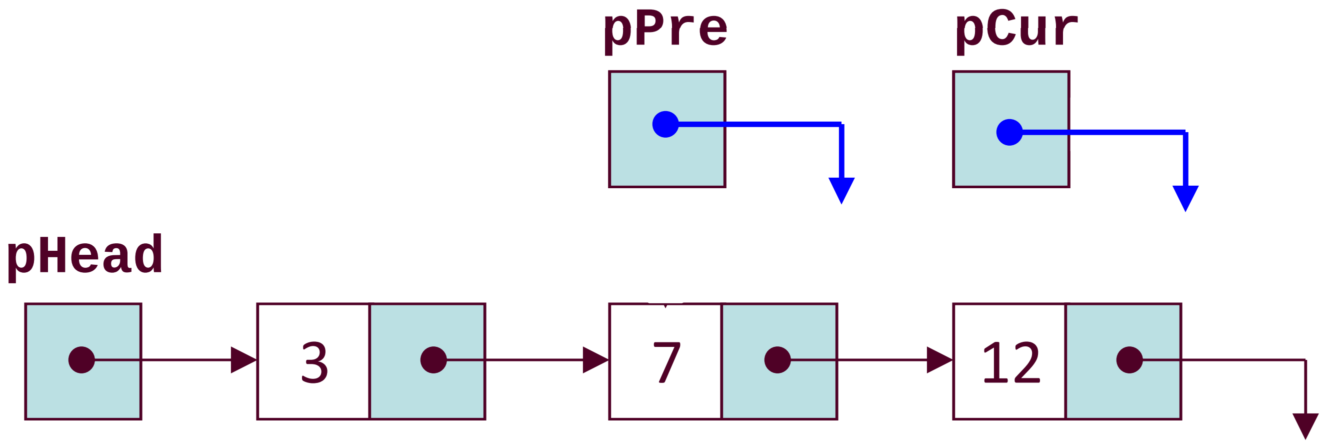
Example – Step 2

- Deallocate the space occupied by the deleted node



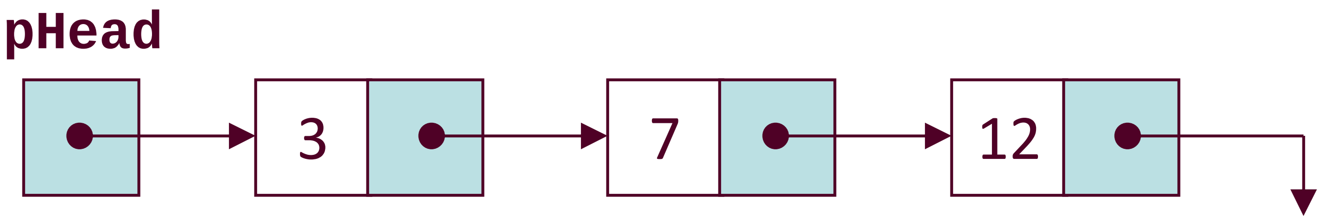
Example – After Deletion

What should you do with **pPre** and **pCur**?



Deleting the Last Node

- What do you need and what will you need to do?



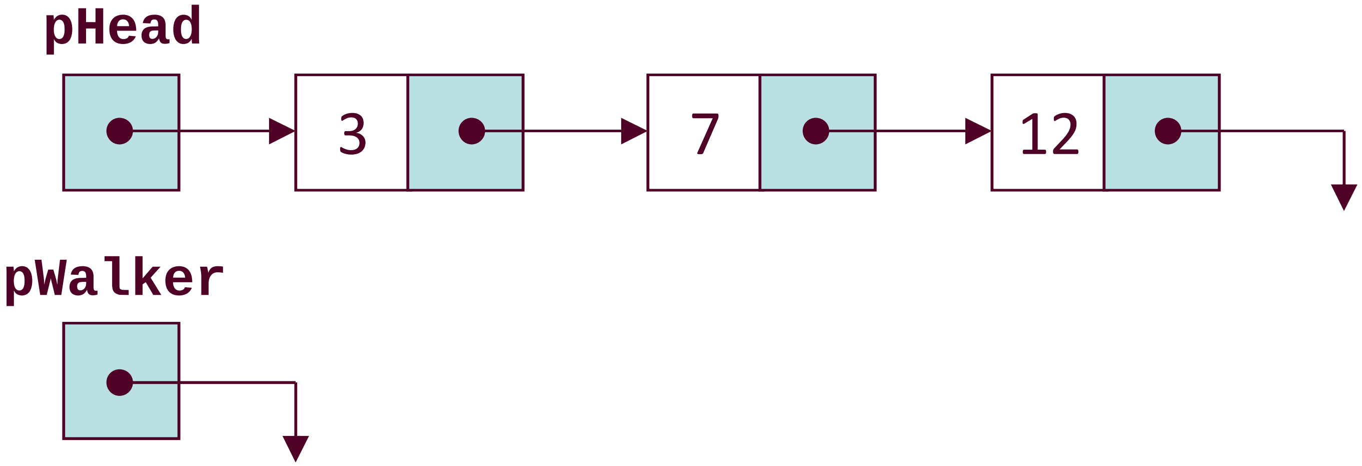
Traversing a Linked List

- Traversal means visiting each node of the linked list, e.g., reading/displaying the list from start to end
- Algorithm
  - start at the first node
  - follow the chain of pointers
  - end at the last node

Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

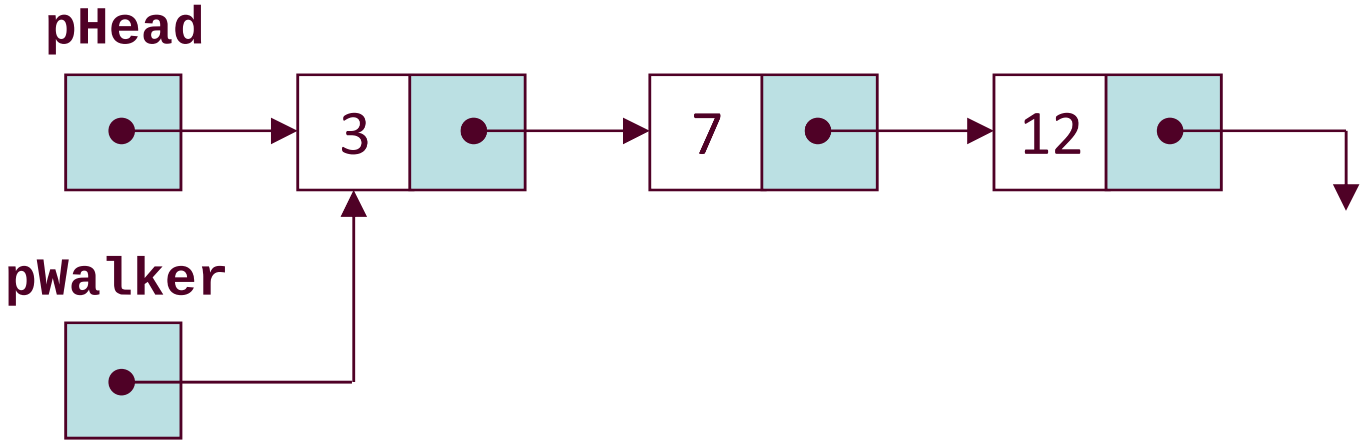
Output



Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

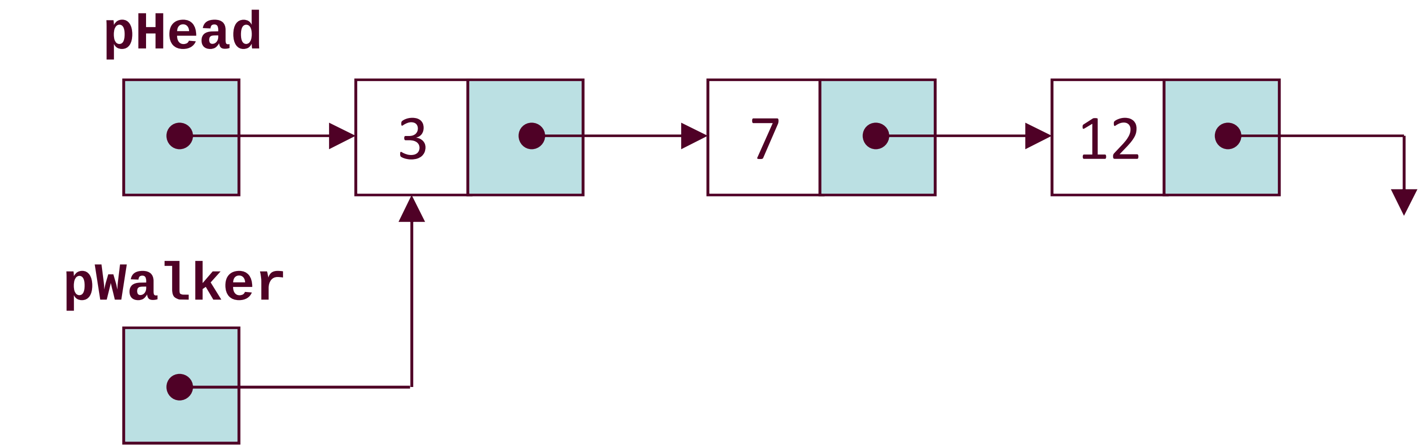
Output



Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
   cout << pWalker->data << endl;
   pWalker = pWalker->next;
}
```

Output

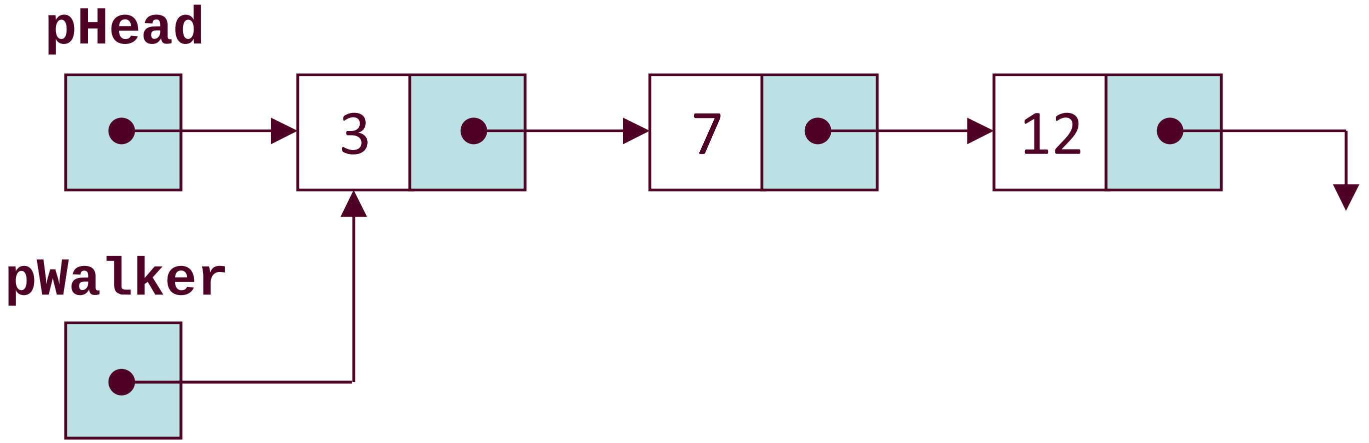


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

Output

```
3
```

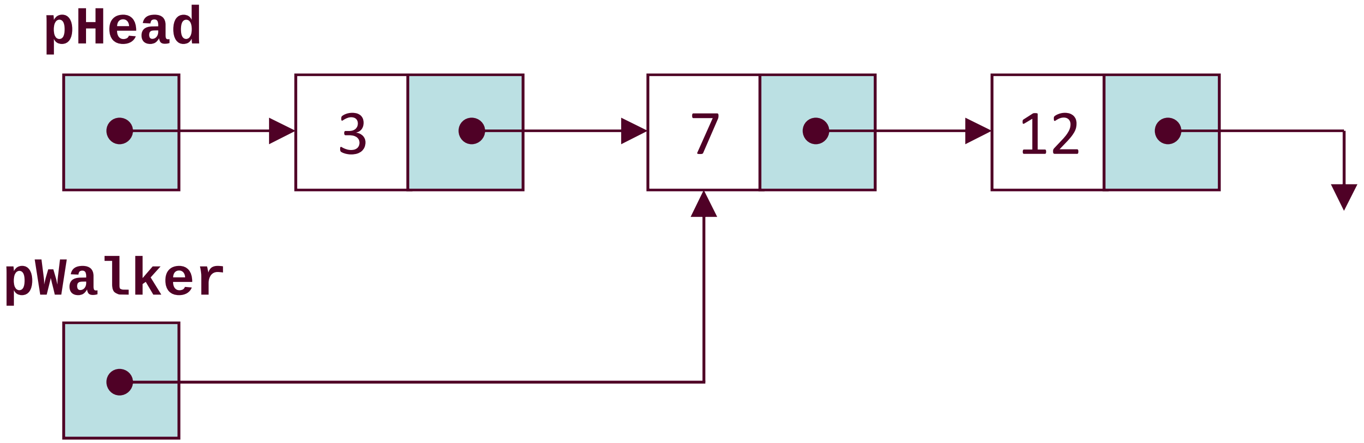


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
   cout << pWalker->data << endl;
   pWalker = pWalker->next;
}
```

Output

```
3
```

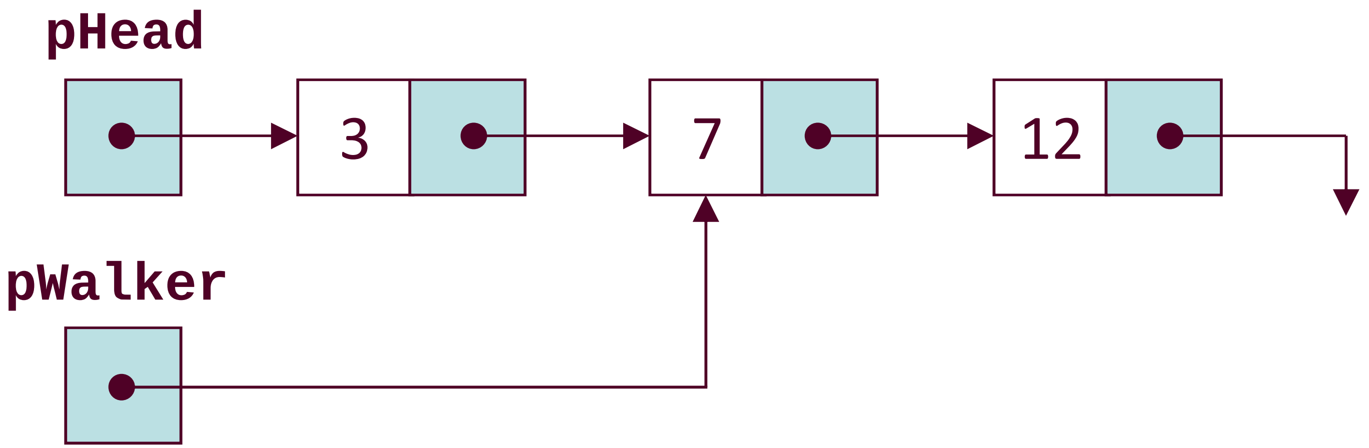


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

Output

```
3
```

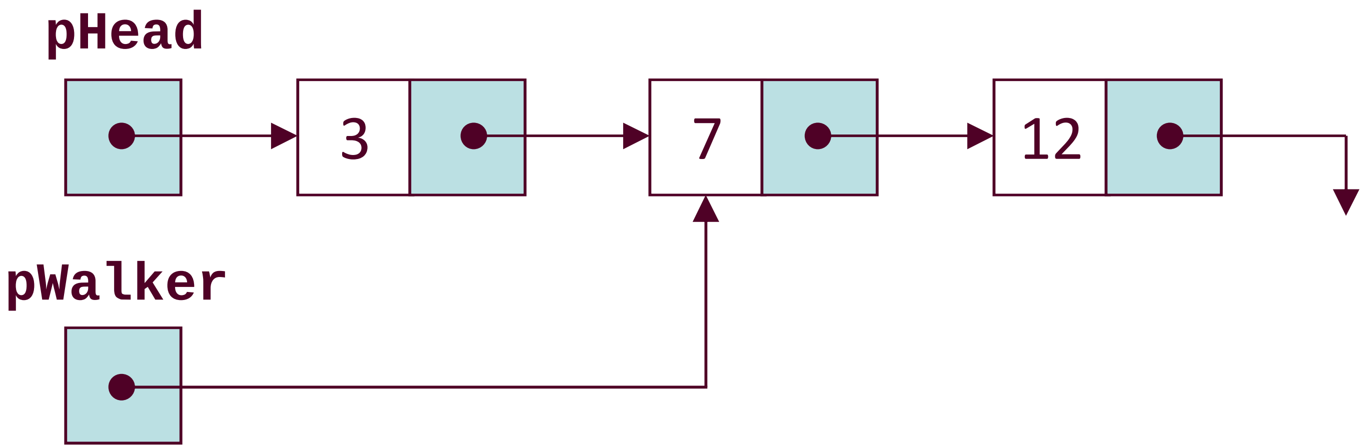


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

Output

```
3
7
```

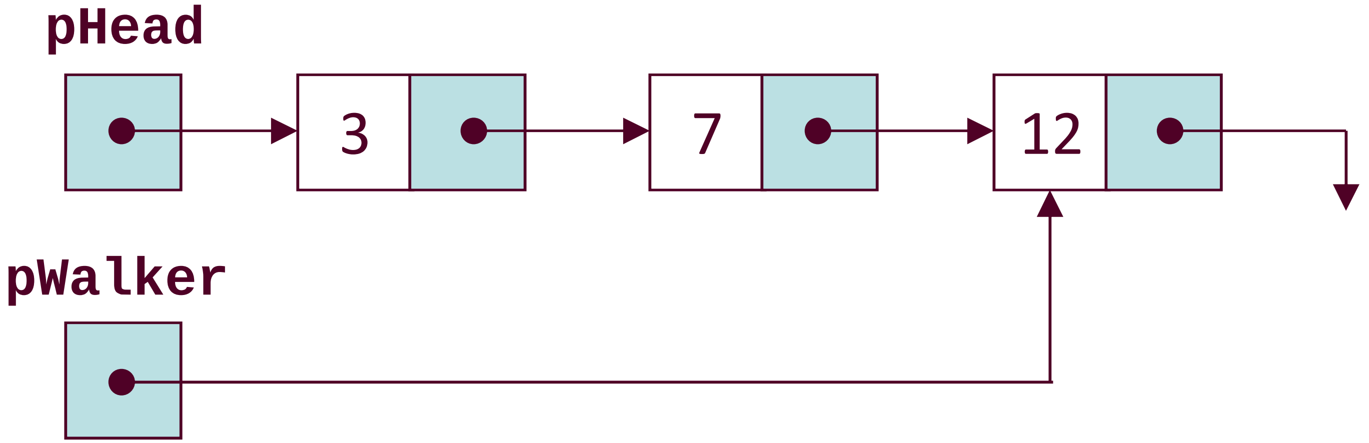


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

Output

```
3
7
```

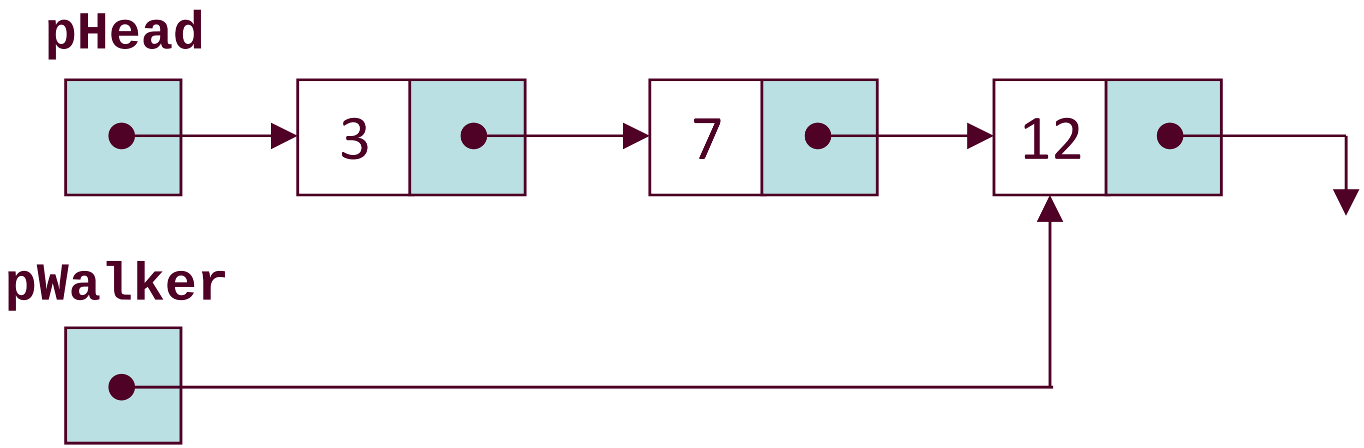


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

Output

```
3
7
```

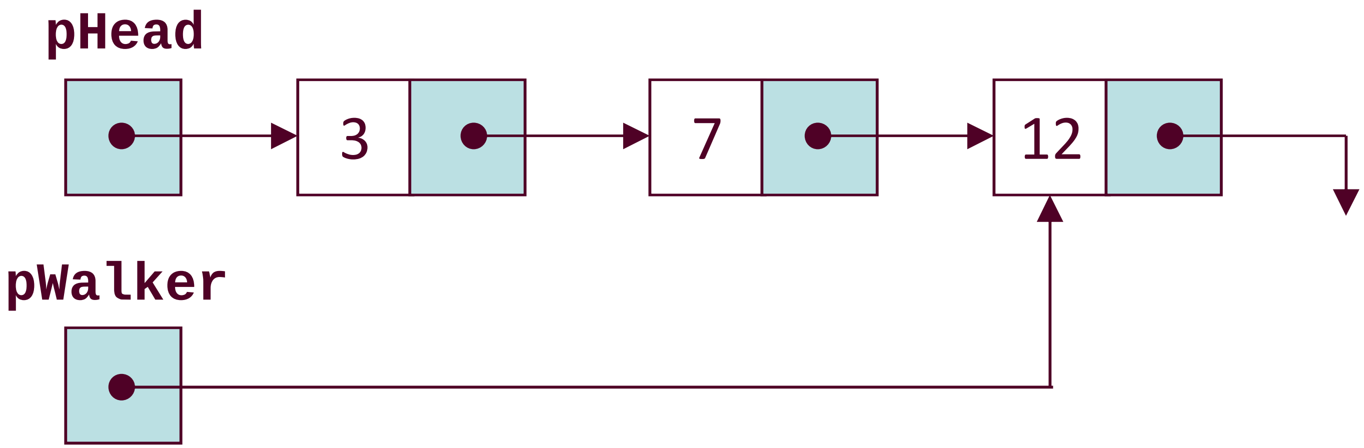


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

Output

```
3
7
12
```


pHead

3

7

12

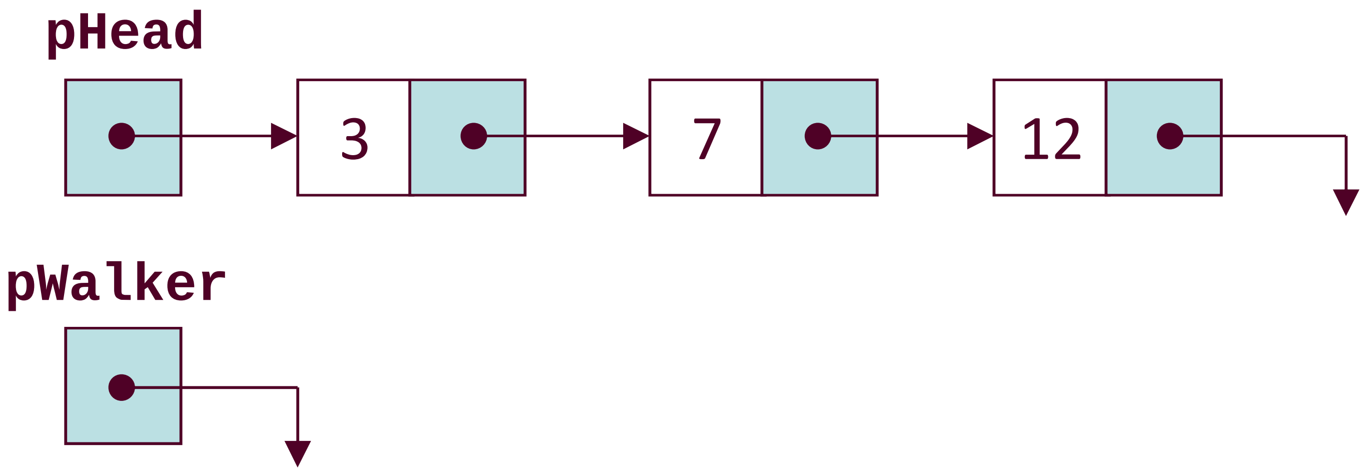
pWalker

Example

```
    Node *pWalker = NULL;
    pWalker = pHead;
    while (pWalker != NULL)
    {
        cout << pWalker->data << endl;
→       pWalker = pWalker->next;
    }
```

Output

```
3
7
12
```

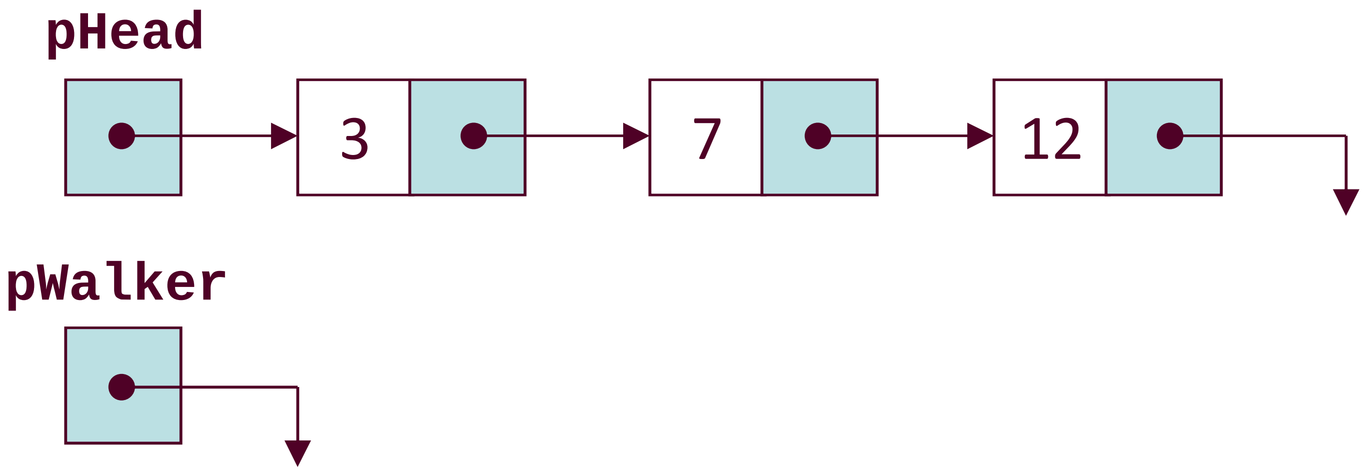


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

Output

```
3
7
12
```

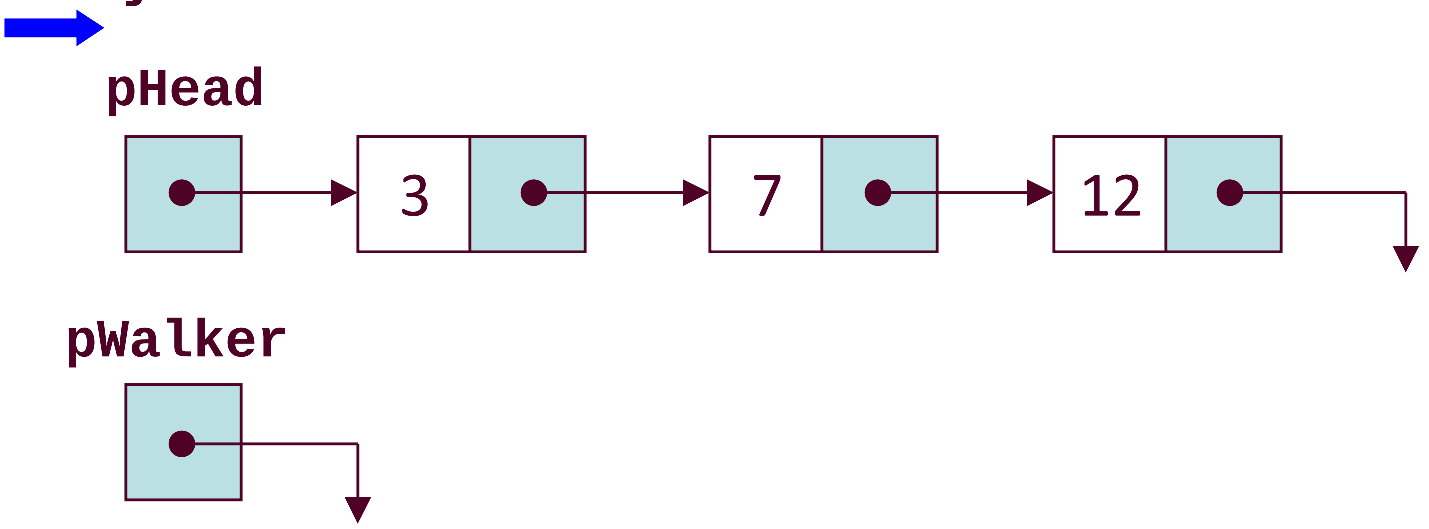


Example

```
Node *pWalker = NULL;
pWalker = pHead;
while (pWalker != NULL)
{
    cout << pWalker->data << endl;
    pWalker = pWalker->next;
}
```

Output

```
3
7
12
```



Searching a Linked List

- Essentially a sequential search
  - start at the beginning of the list and search until you find the target
- Write a function that will
  - accept a linked list and a target
  - return a pointer to the target node (what if the target is not found?)

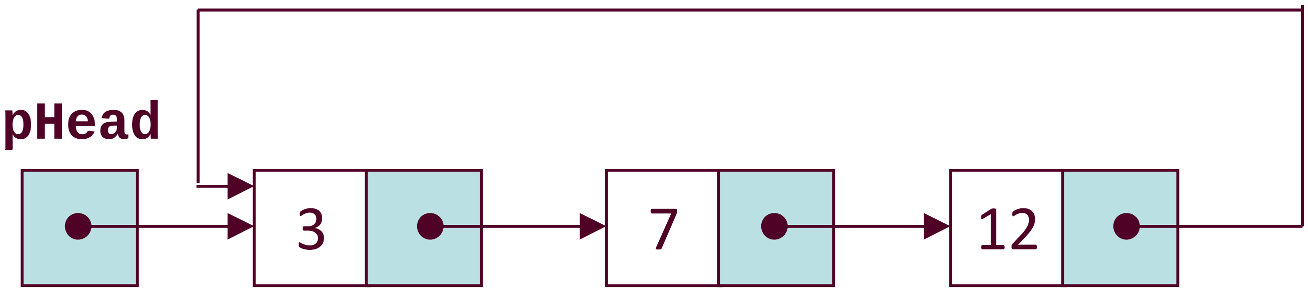
Linked Lists vs. Arrays

- Advantages
  - overflow can never occur unless the memory is actually full
  - insertions and deletions are *faster*
  - with large objects, moving pointers is easier and faster than moving the objects themselves
- Disadvantages
  - the pointers require extra space
  - do not allow random access
  - programming is typically trickier

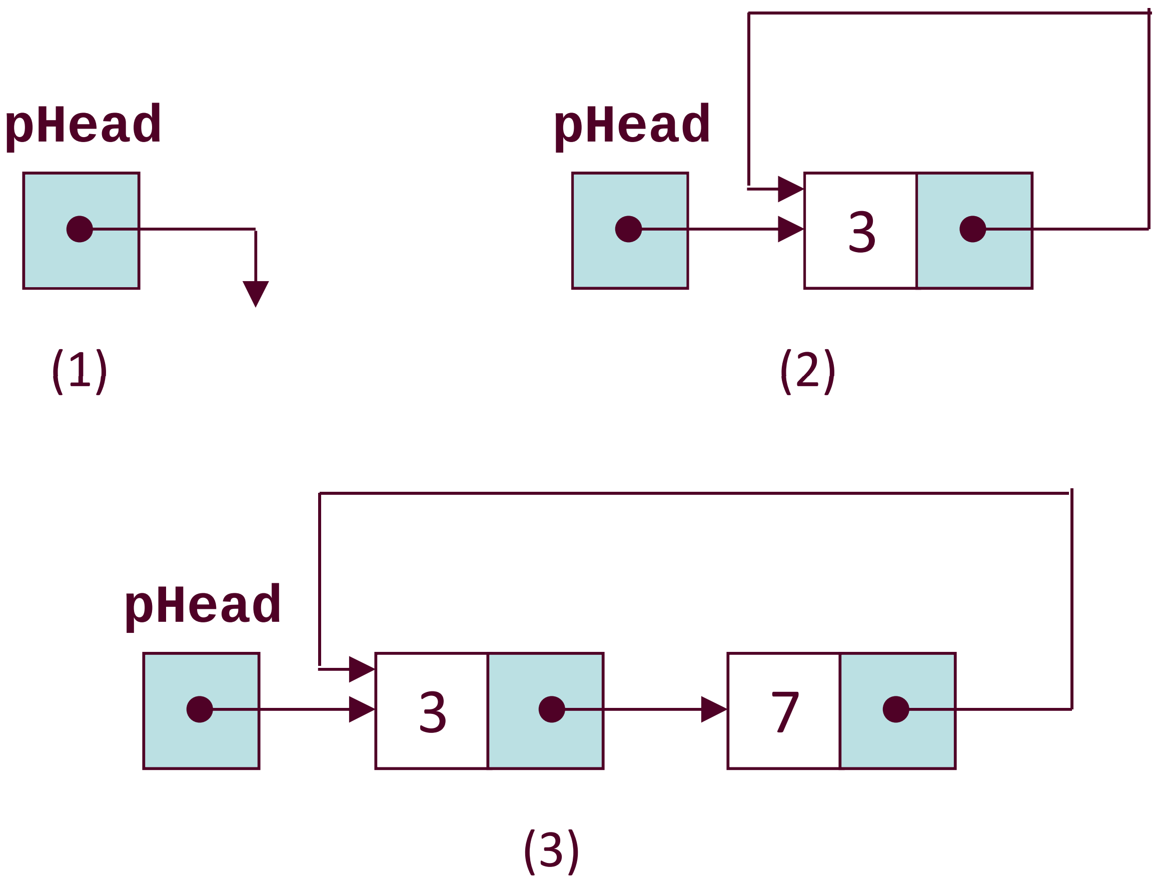
Other Linked List Operations

- Destroying a list
- Building an ordered list
- Copying one list to another
- Appending one list at the end of another
- Swapping two nodes

Circular Linked Lists

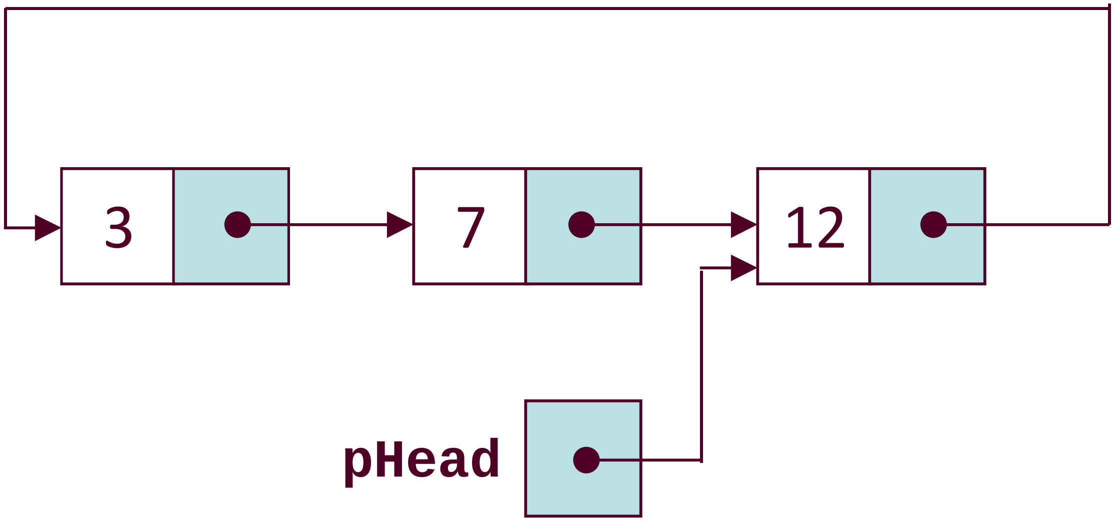


Building a Circular Linked List



Circular Linked Lists

- Easier access with the head pointer pointing to the last node



Two-way (Doubly) Linked Lists

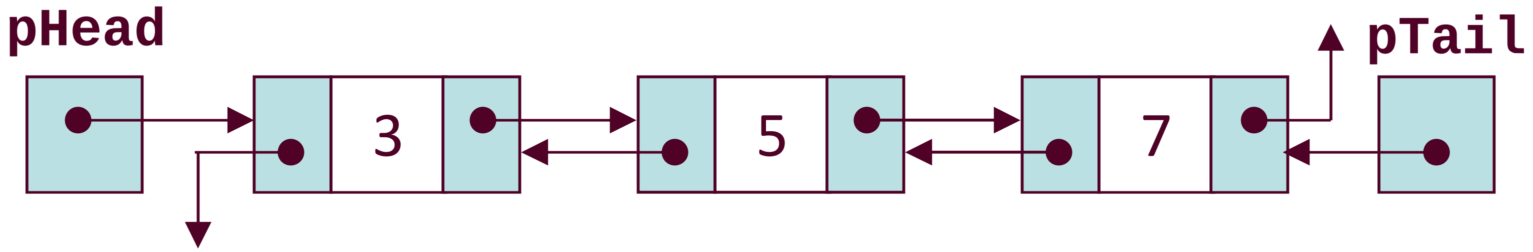
Two-way Linked Lists

- One that can be traversed in both directions – forward and backward
- Every node has two pointers:
  - one pointing to the next node (except the last node)
  - one pointing to the previous node (except the first node)
- Two external pointers - head and tail

Two-way vs. One-way

- Advantages
  - list can be traversed in both directions
  - an object can be accessed to the left/right
  - some insertion and deletion become easier
  - supports everything that can be done by a one-way list
- Disadvantages
  - more space required for each node
  - programming a bit more complicated

Two-way Linked List of Integers

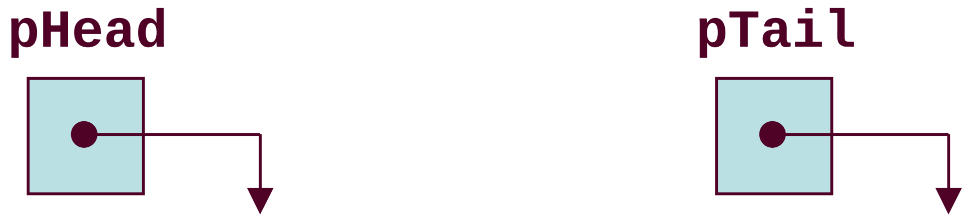


Redefining the **Node**

```
struct Node
{
   int data;      // data stored at this node
   Node *next; // pointer to next node
   Node *prev; // pointer to previous node
};
```

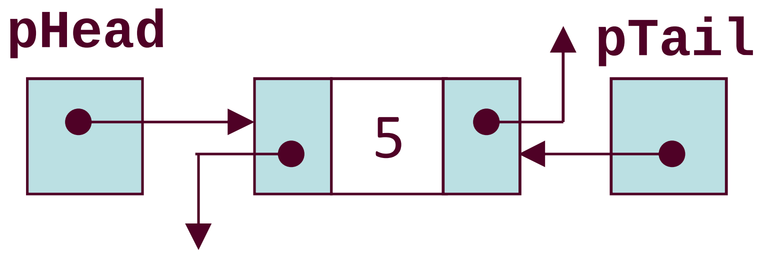
Creating a 2-way Linked List

- Declare head and tail pointers:



Creating a 2-way Linked List ...

- Create the first node with data value of 5:

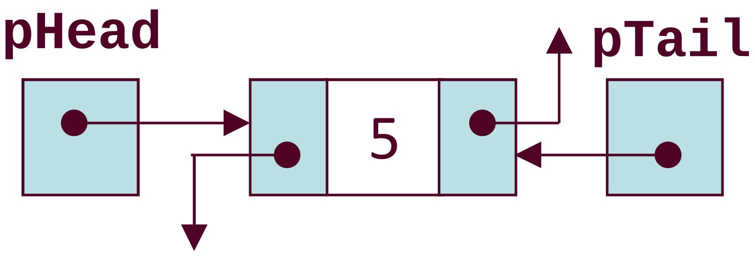


Inserting Nodes

- Requires more pointer changes than in one-way lists
- Steps:
  - find the place to insert
  - create the new node
  - adjust the pointers
- Can be done in different ways:
  - inserting at the beginning
  - inserting after a target node
  - inserting at the end

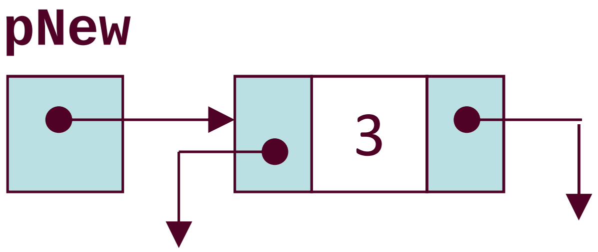
Inserting at the Beginning

- Steps:
  - create up the new node
  - adjust the pointers
- Example - Insert a node with a data value of 3 at the beginning of the following list



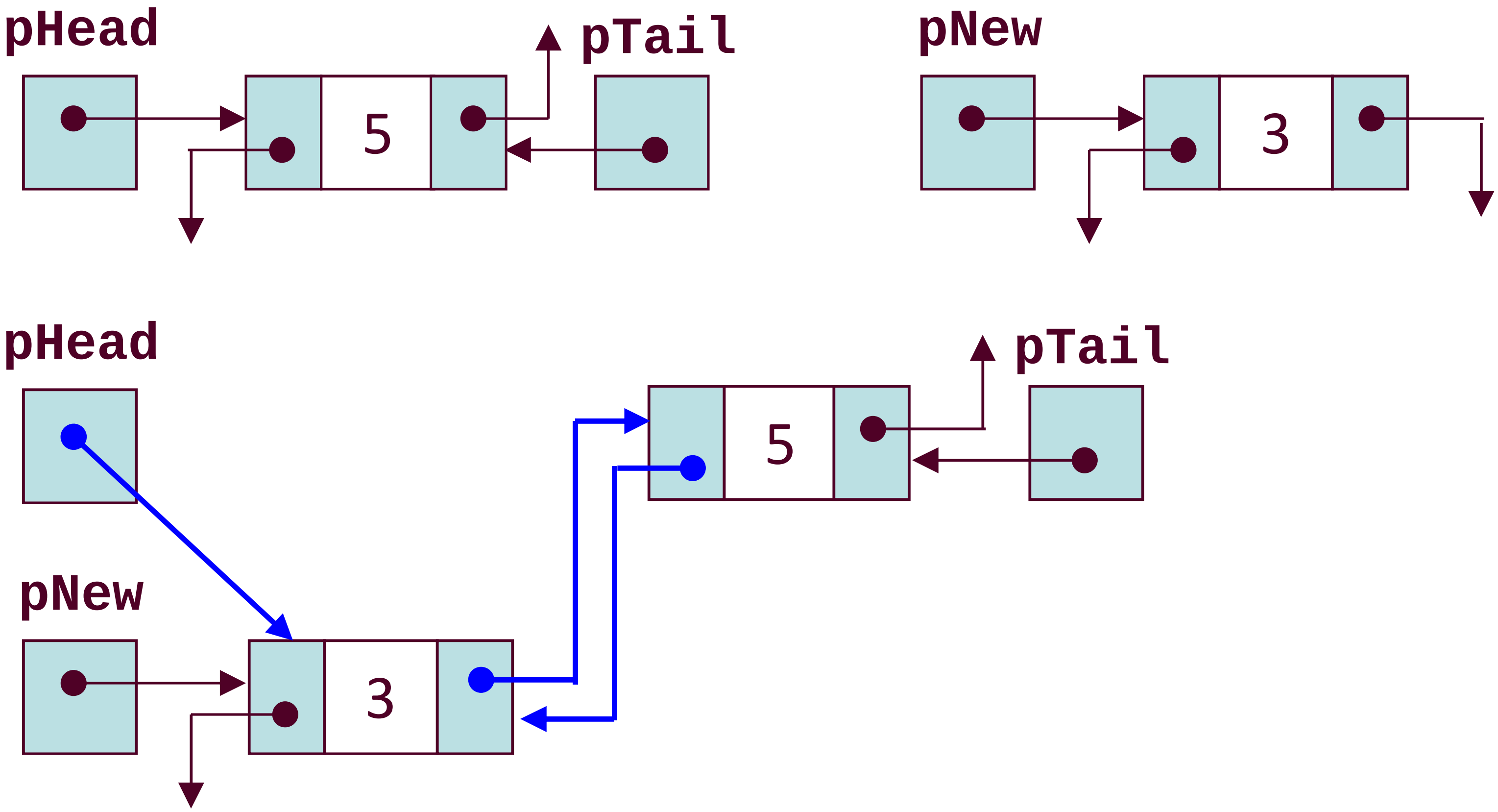
Example – Step 1

- Create up a new node with a data value of 3

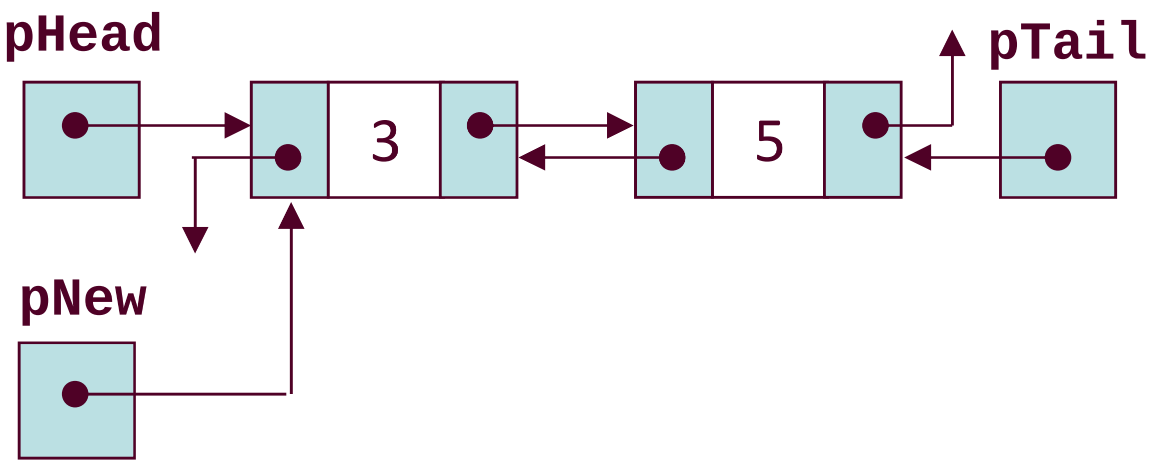


Example – Step 2

- Adjust the pointers



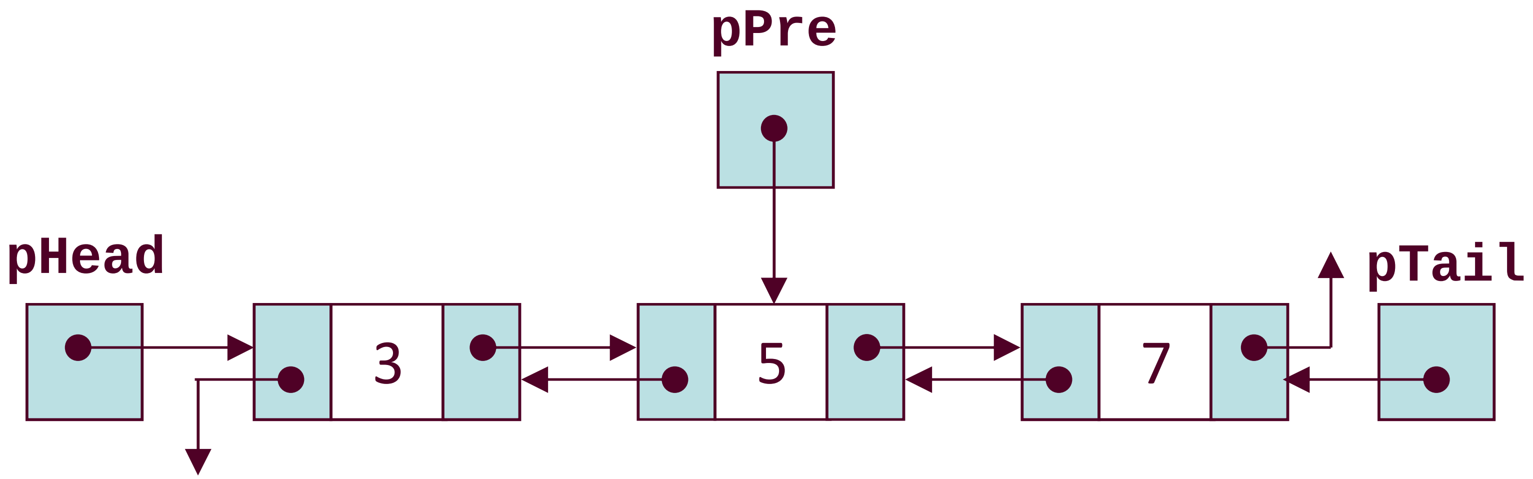
Example – After Step 2



What should you do with **pNew**?

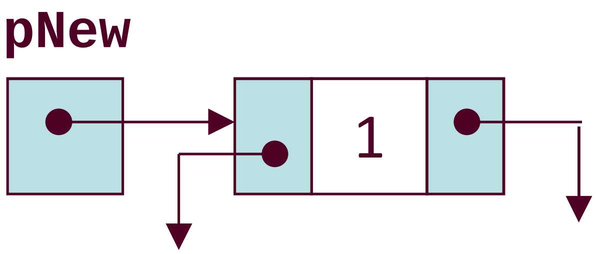
Inserting in the Middle

- Insert a node with a data value of 1 after the node pointed to by **`pPre`**



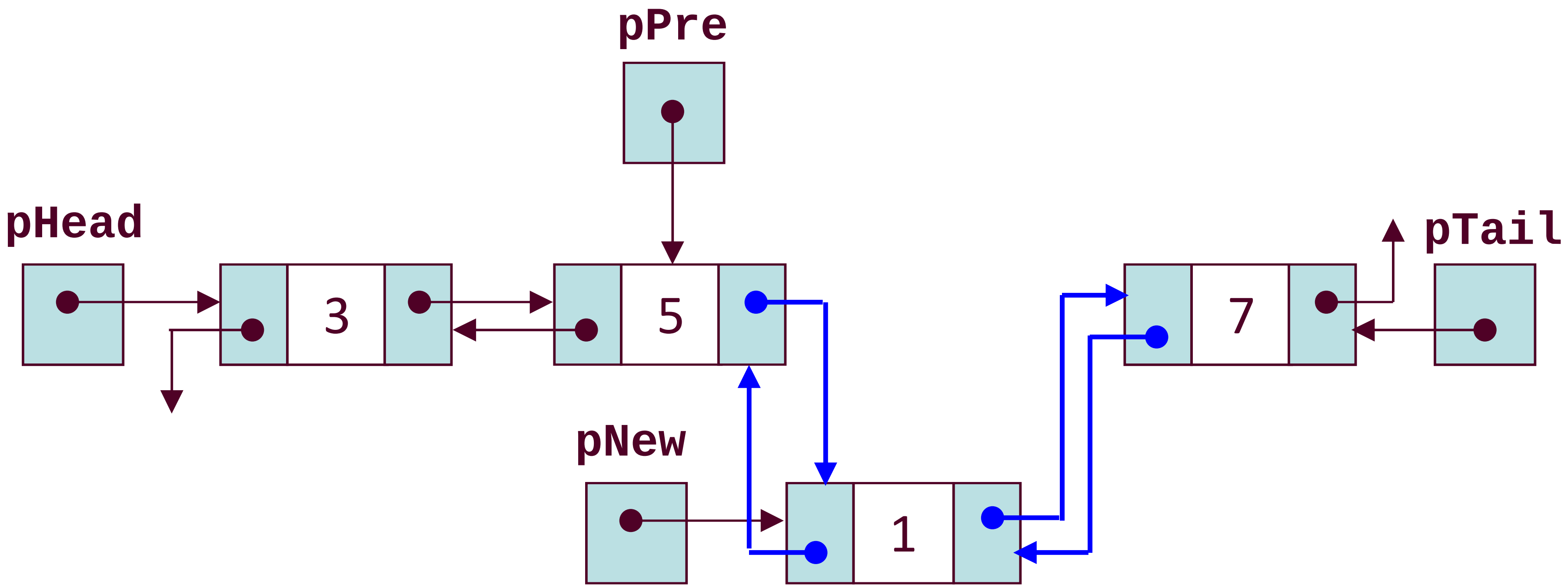
Example – Step 1

- Create up a new node with a data value of 1

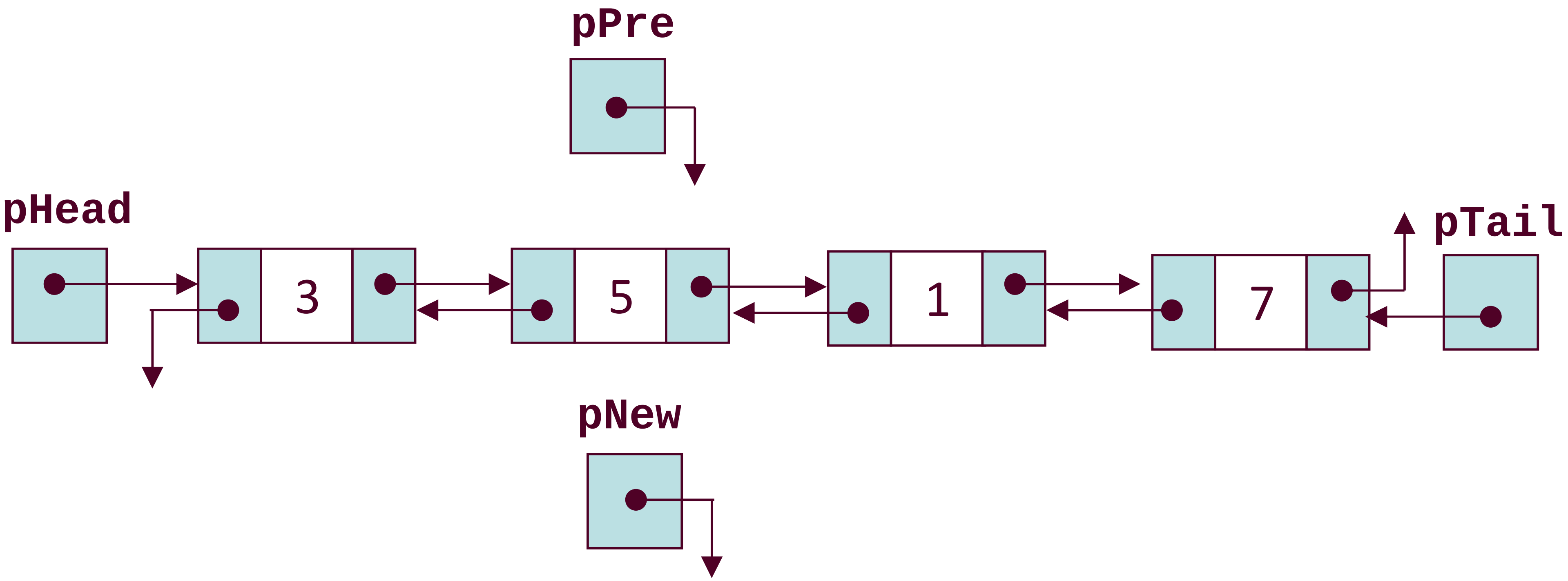


Example – Step 2

- Adjust the pointers



Example – After Step 2



Insertion at the End

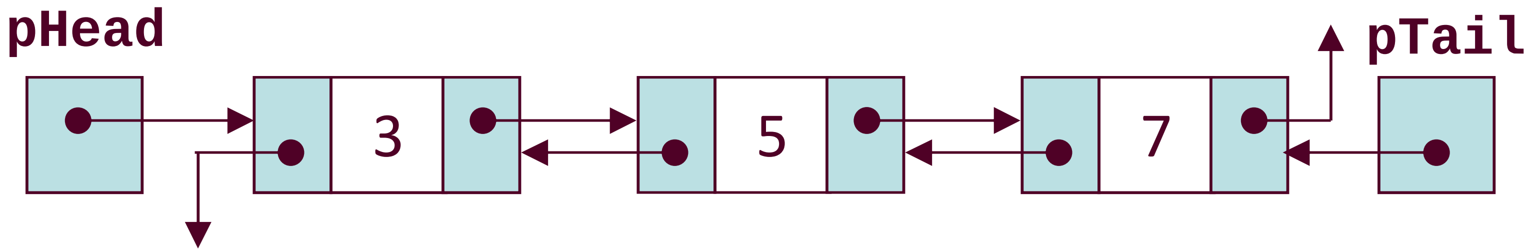
- What is the advantage over a one-way list?

Deleting Nodes

- Requires more pointer changes than in one-way lists
- Can be done in different ways:
  - deleting the first node
  - deleting a node somewhere in the middle
  - deleting the last node

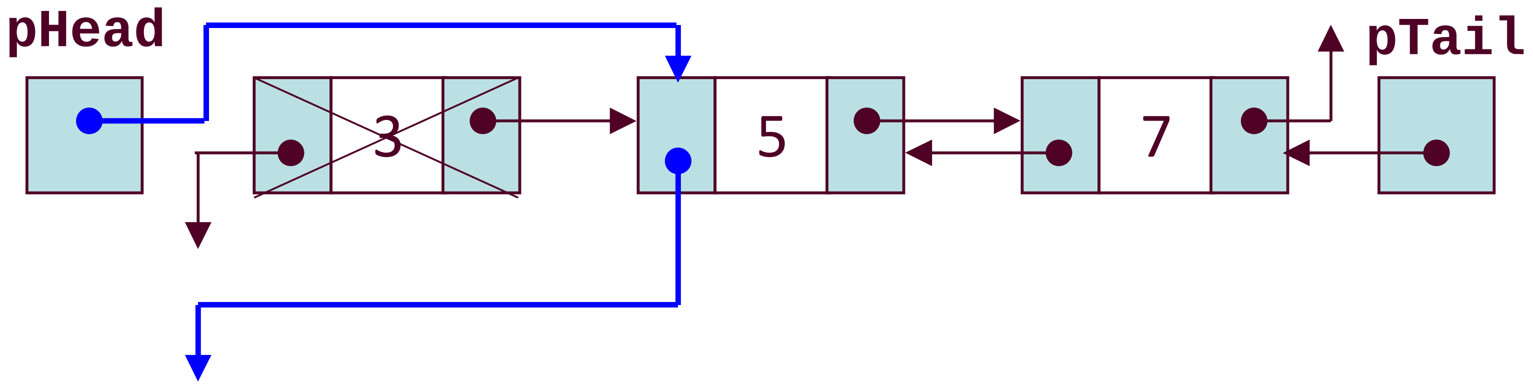
Deleting the First Node

- Delete the first node from the following list:



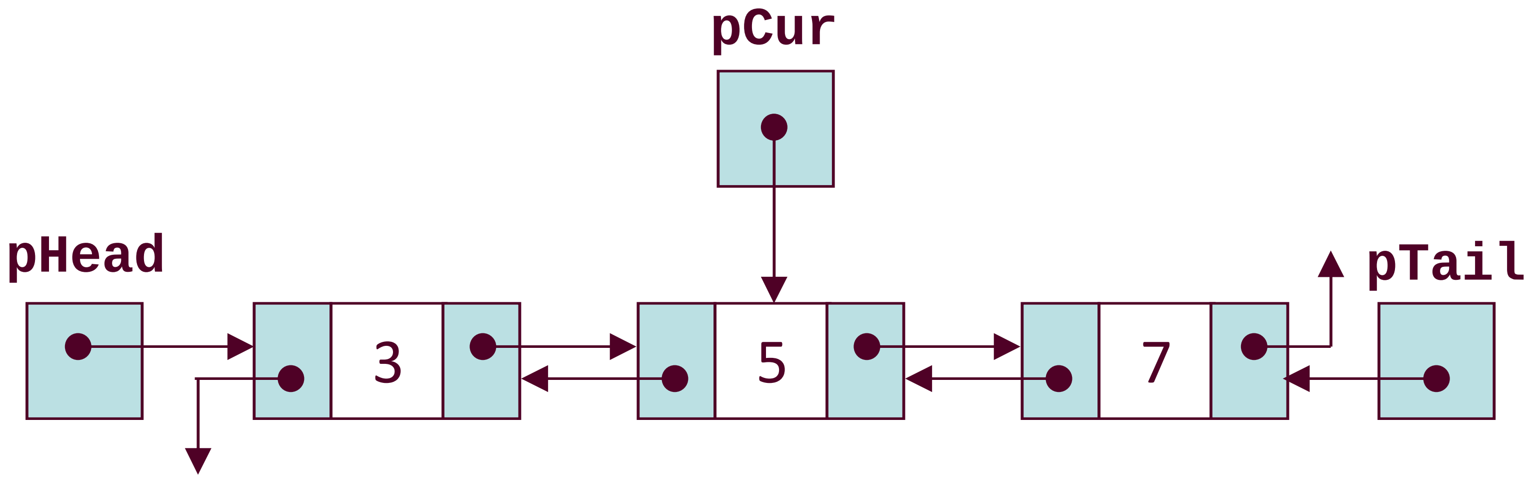
Deleting the First Node

- Adjust pointers and deallocate the space occupied by the deleted node



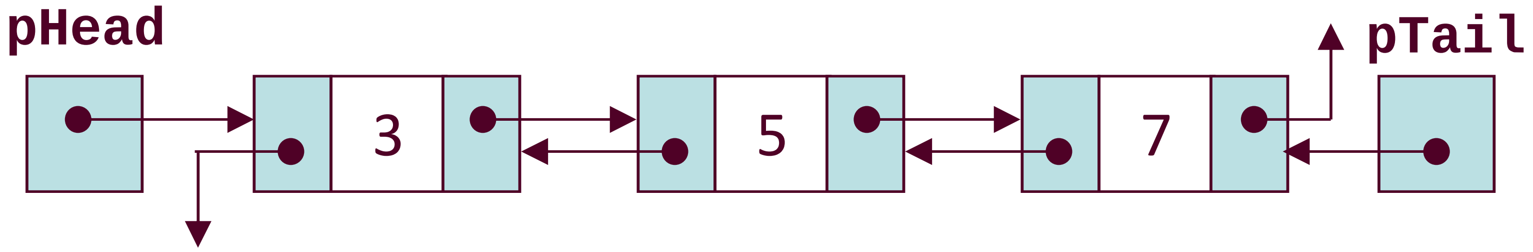
Deleting a Target Node

- Delete the node pointed to by **`pCur`**:



Deleting the Last Node

- Delete the last node from this list:



Traversing a Two-way Linked List

- Same as the traversal in a one-way list, but
  - can start at the head
  - can start at the tail