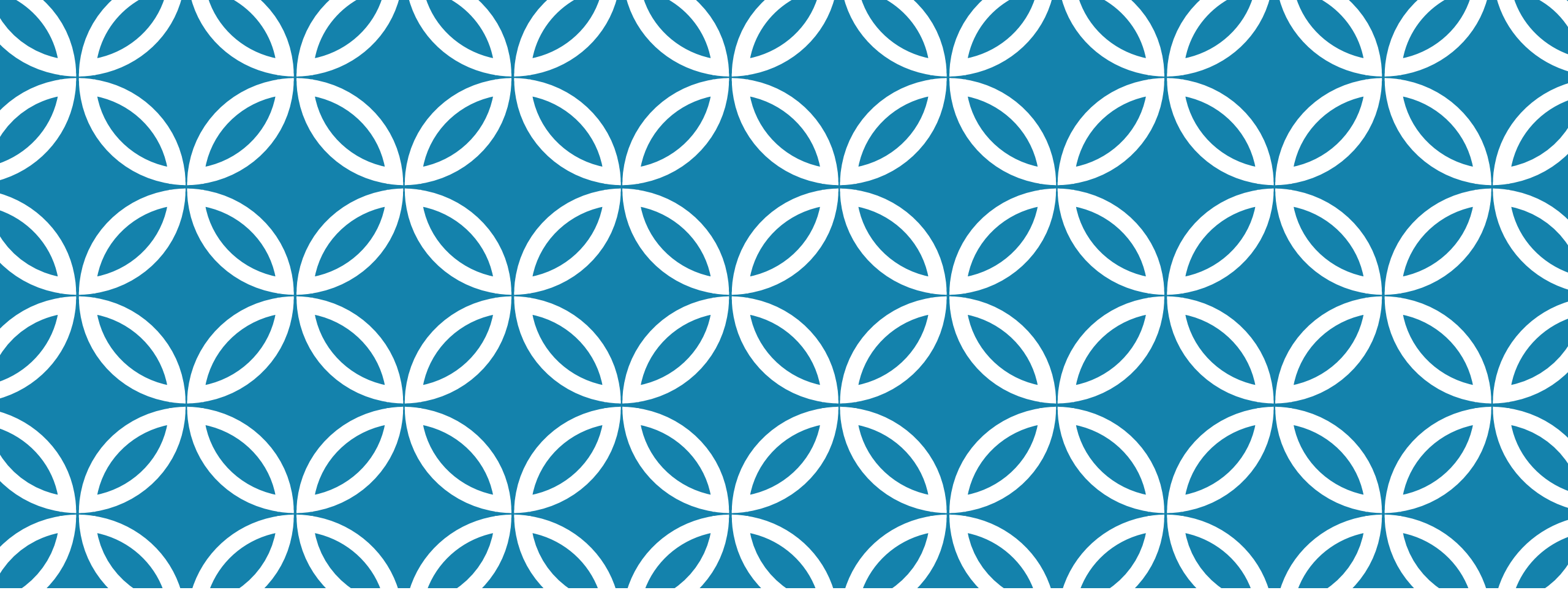


COMP 2611, Data Structures

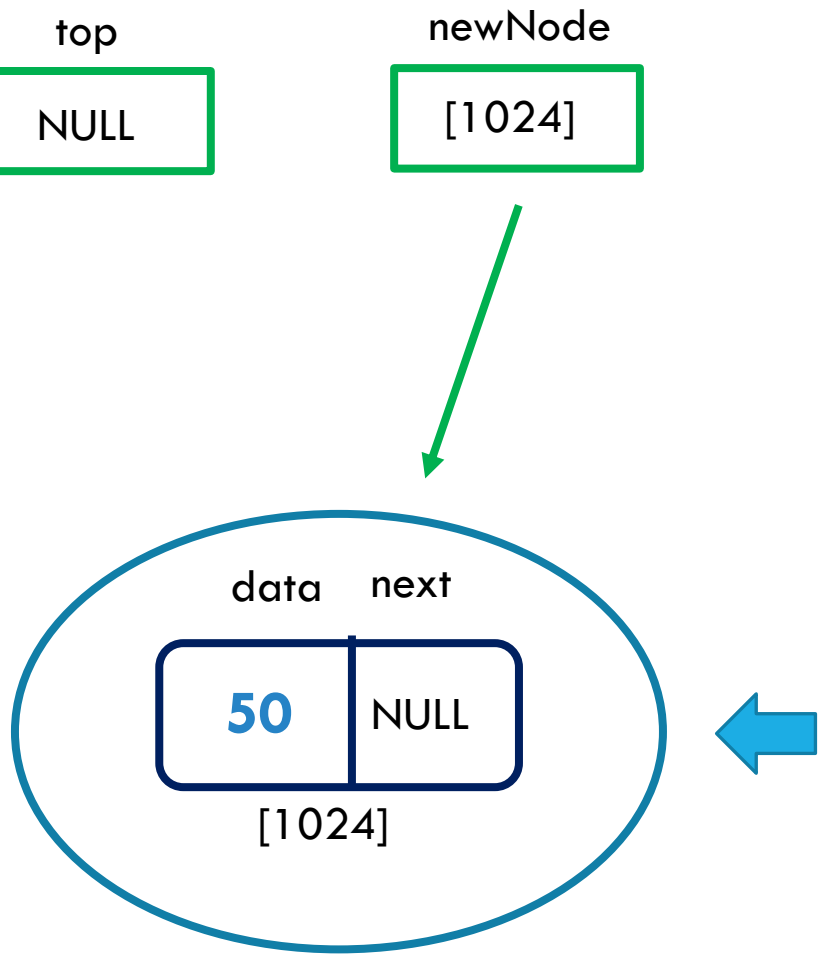
LECTURE 2: REVIEW OF LINKED LISTS (CONTINUED)

INSERTING NODES IN A LINKED LIST

A node can be inserted:

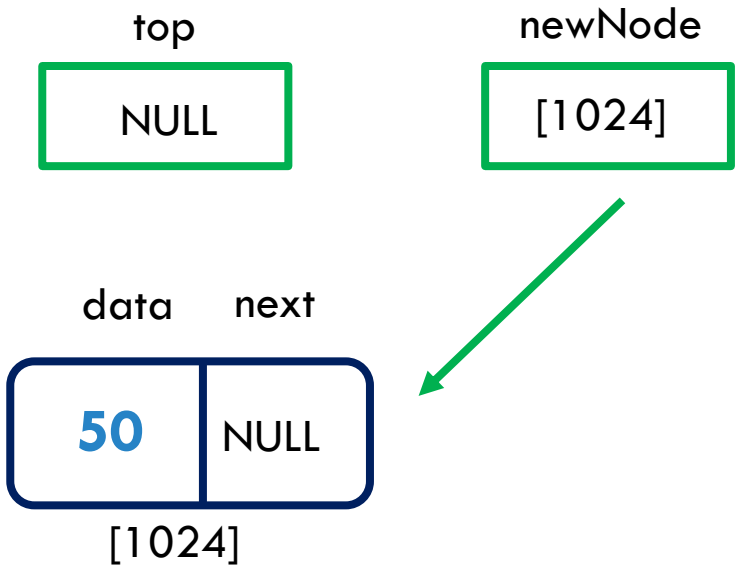
- At the top of the linked list
- At the end of the linked list
- Somewhere between the top and the end of the linked list

We will now look at a few examples where nodes are inserted at the top of a linked list.

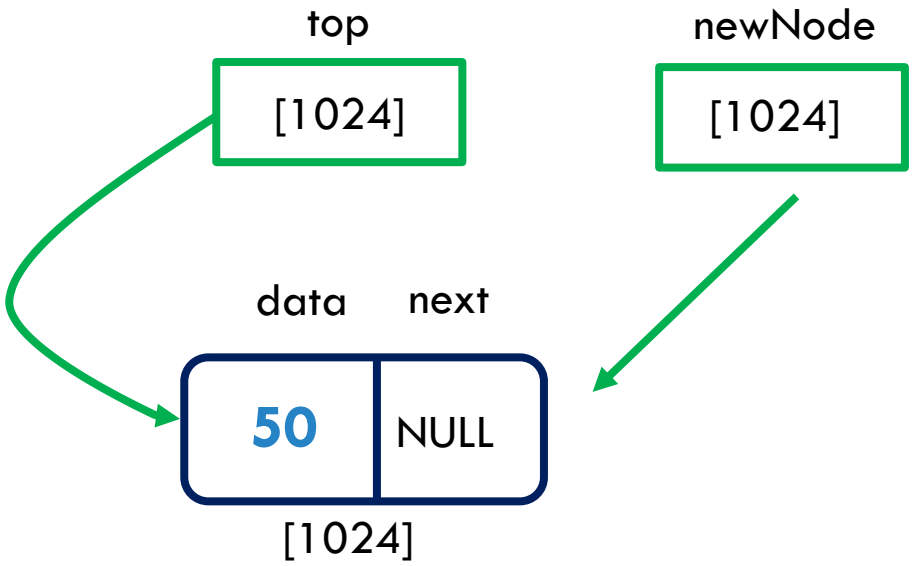



```
Node * top;
Node * newNode;

top = NULL;
```

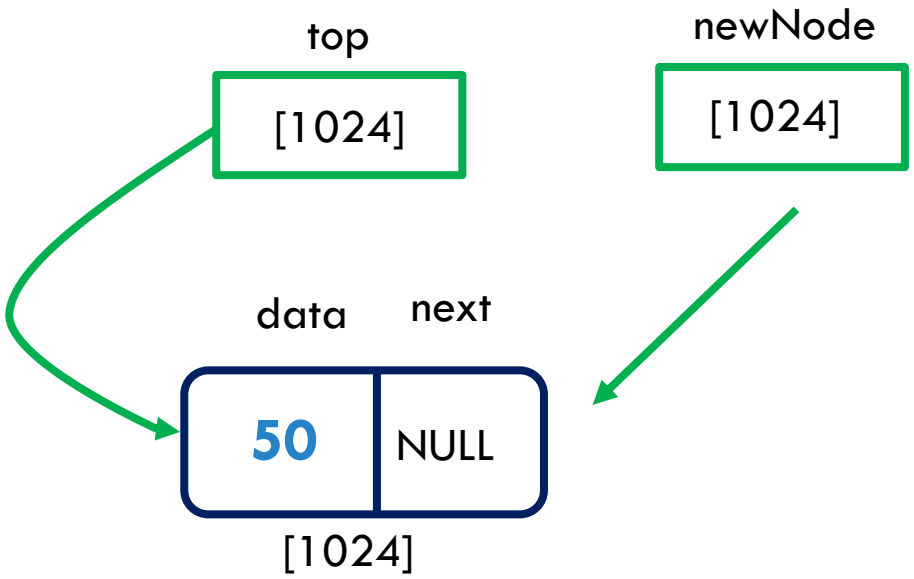
```
newNode = createNode (50);
```

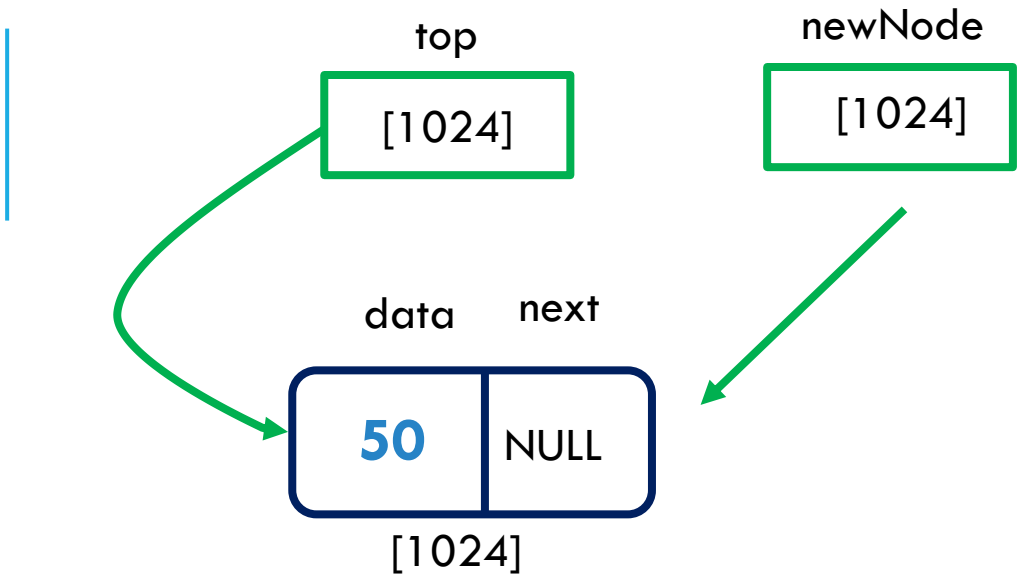


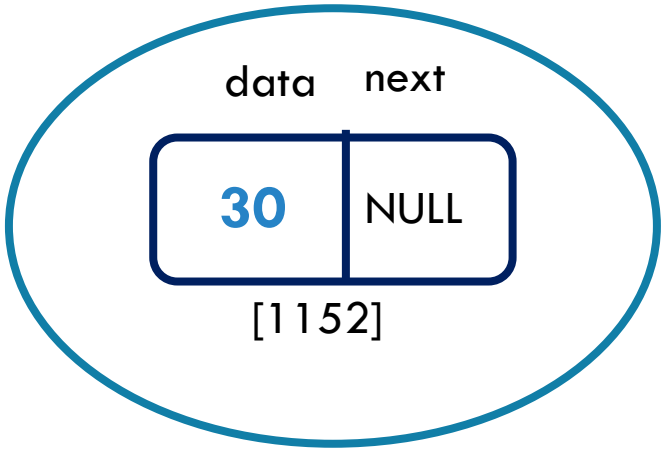

```
top = newNode;
```



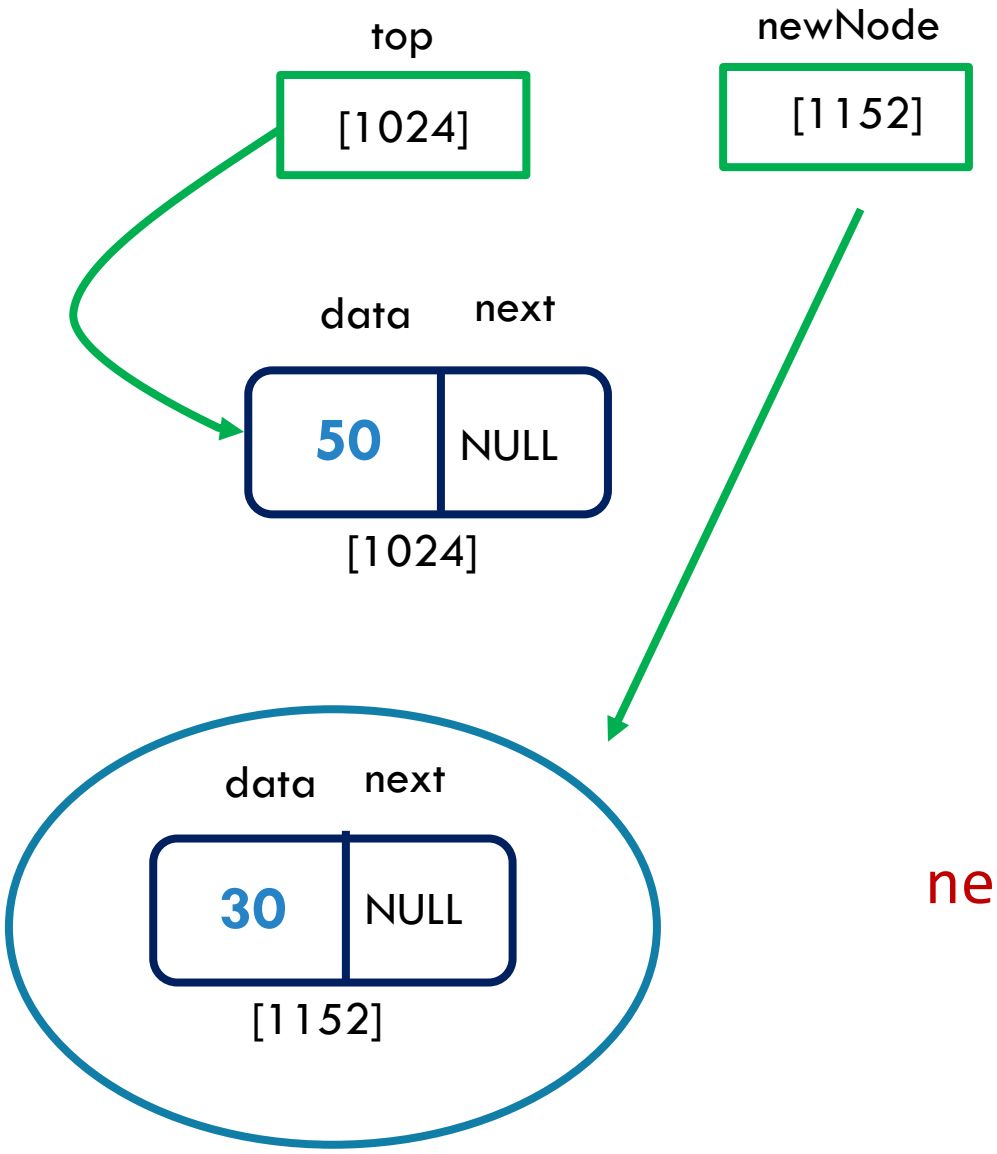

```
top = newNode;
```


top
[1024]
newNode
[1024]
data
next
50
NULL
[1024]

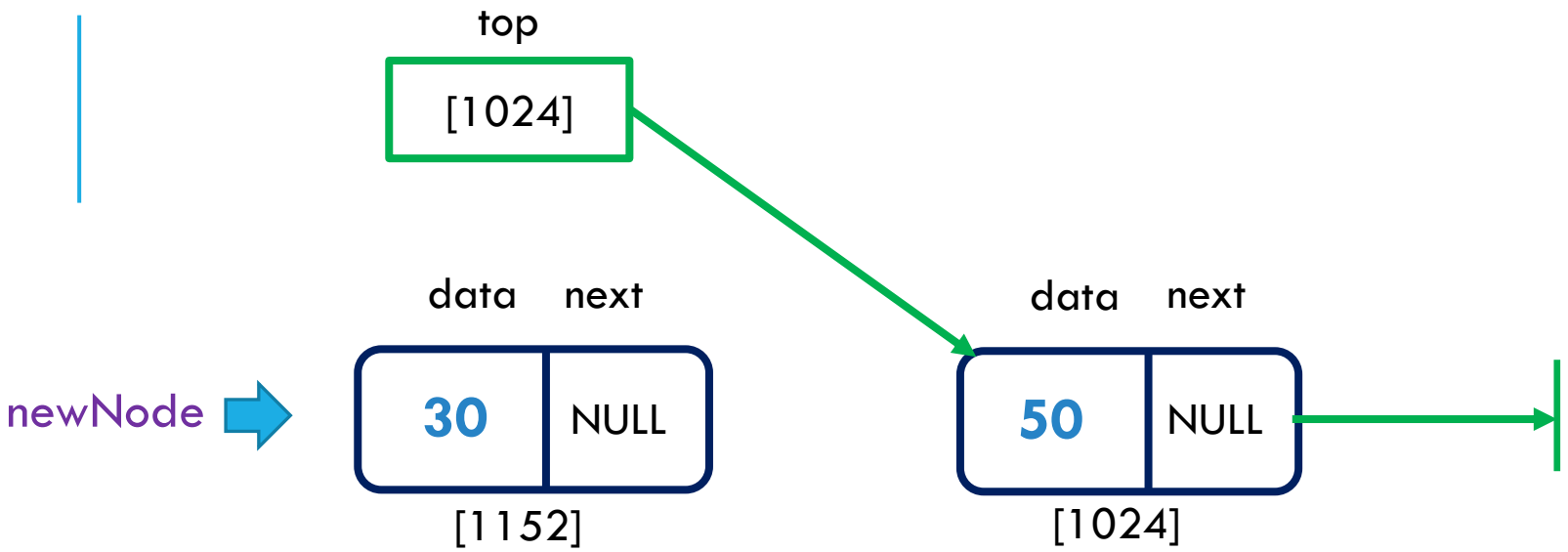





```
newNode = createNode(30);
```



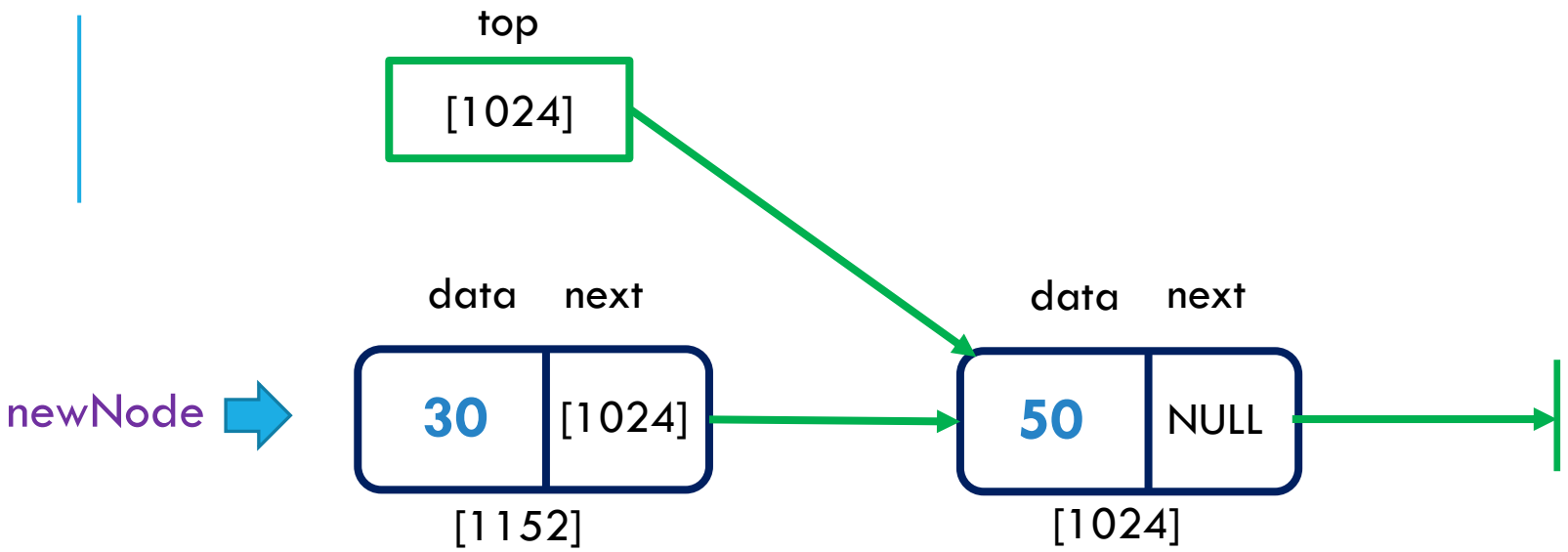

```
newNode = createNode(30);
```



- For *newNode* to point to the node with 50, put the address of the node with 50 in *newNode→next* (i.e, [1024]).

```
newNode->next = top;
```

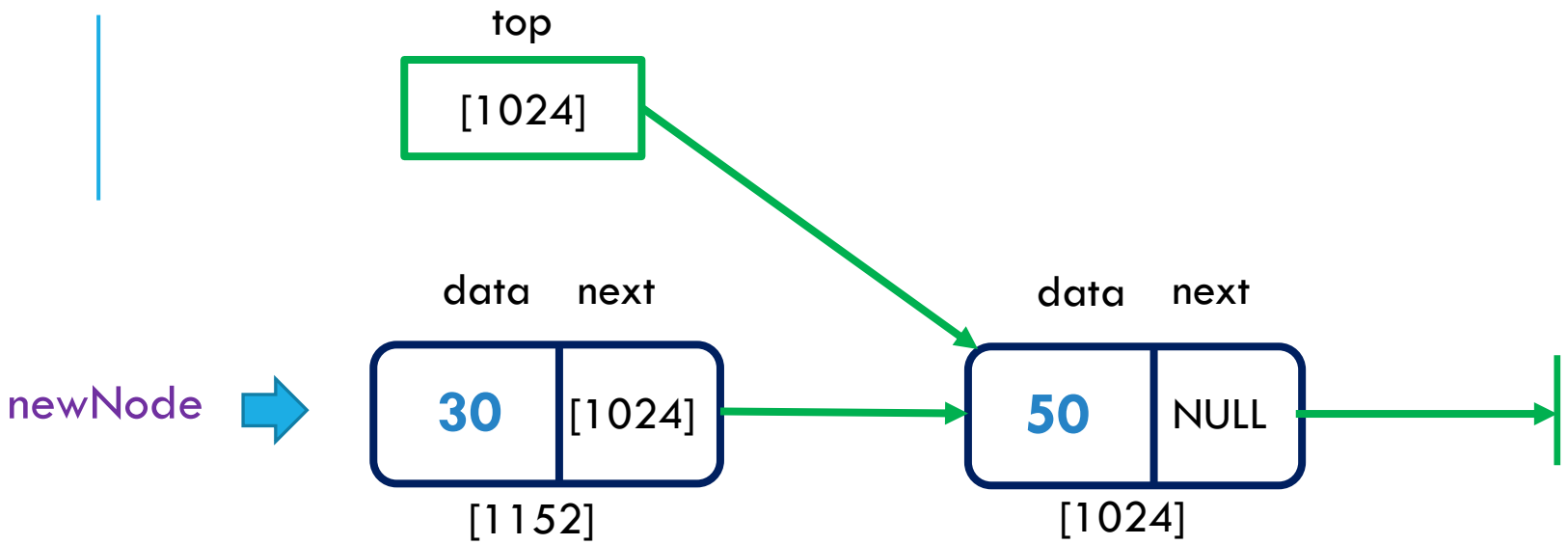
- How to get the address of the node with the value 50?



- For *newNode* to point to the node with 50, put the address of the node with 50 in *newNode→next* (i.e, [1024]).

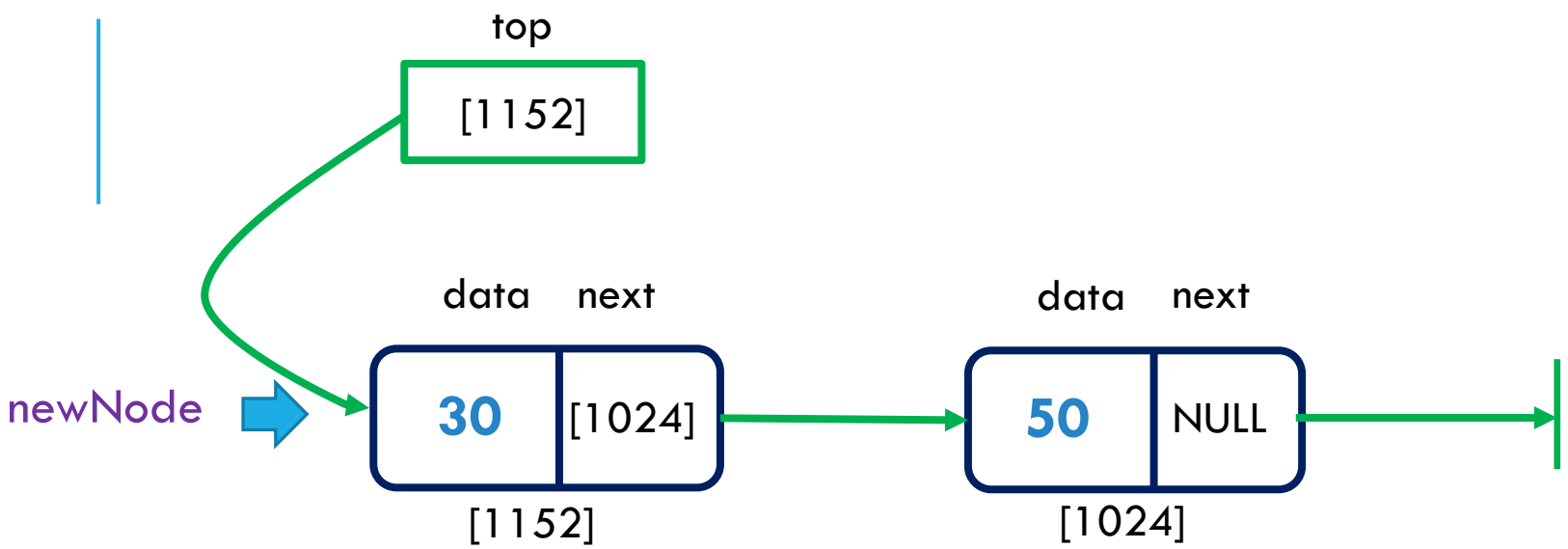
```
newNode->next = top;
```

- How to get the address of the node with the value 50?



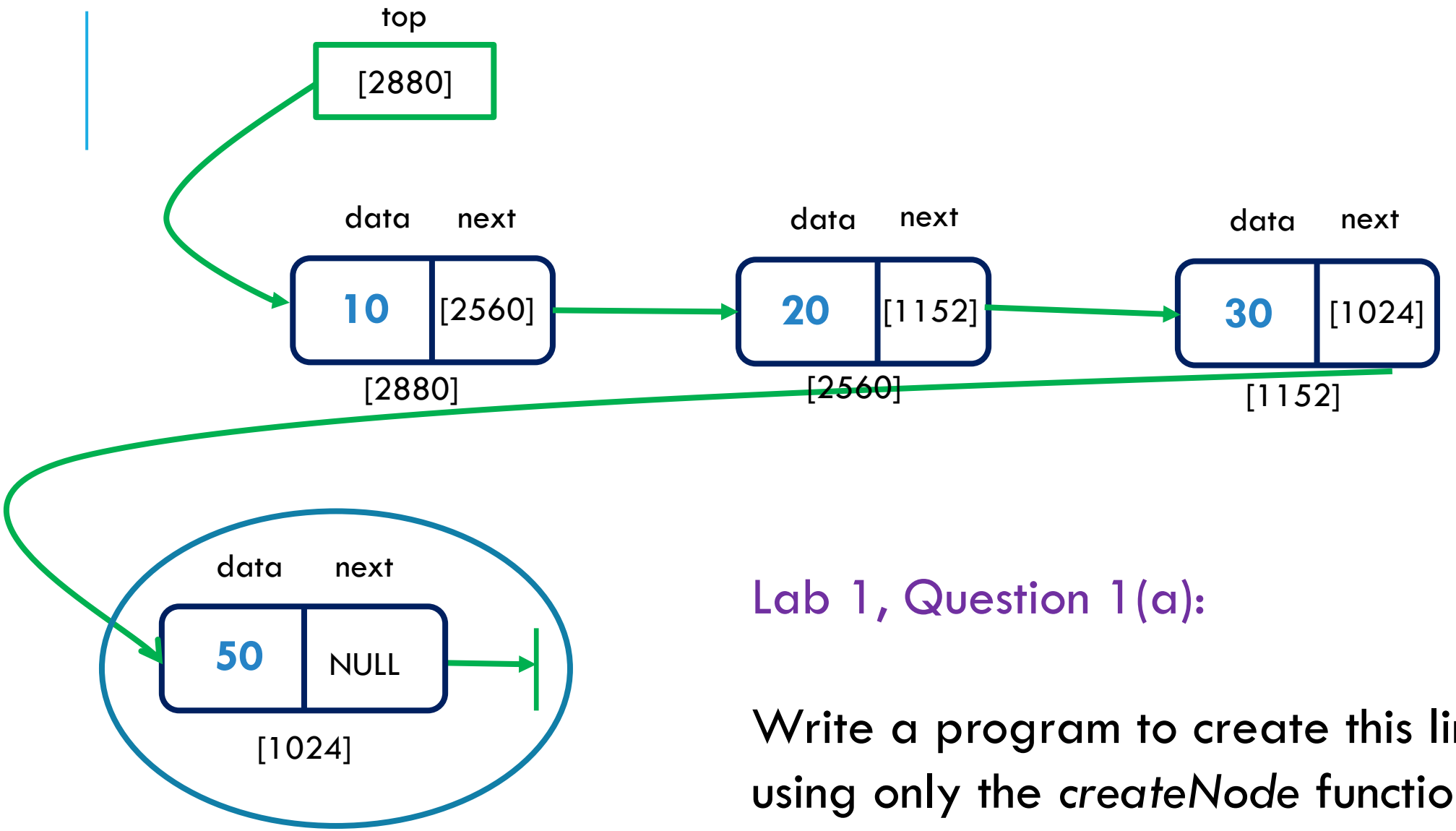
- How to get *top* to point to the newly created node?

```
top = newNode;
```



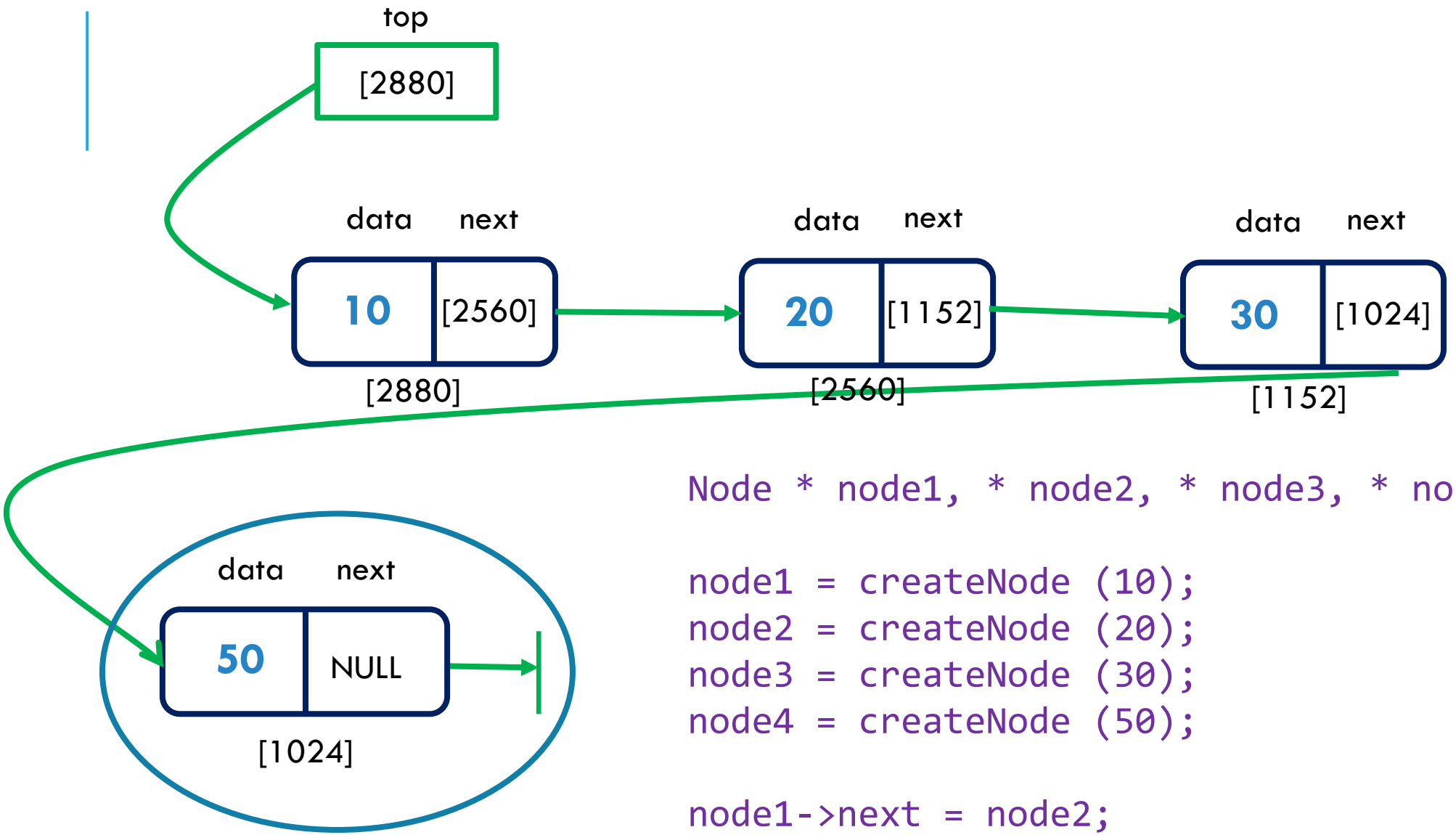
- How to get *top* to point to the newly created node?

```
top = newNode;
```



Lab 1, Question 1(a):

Write a program to create this linked list using only the *createNode* function.

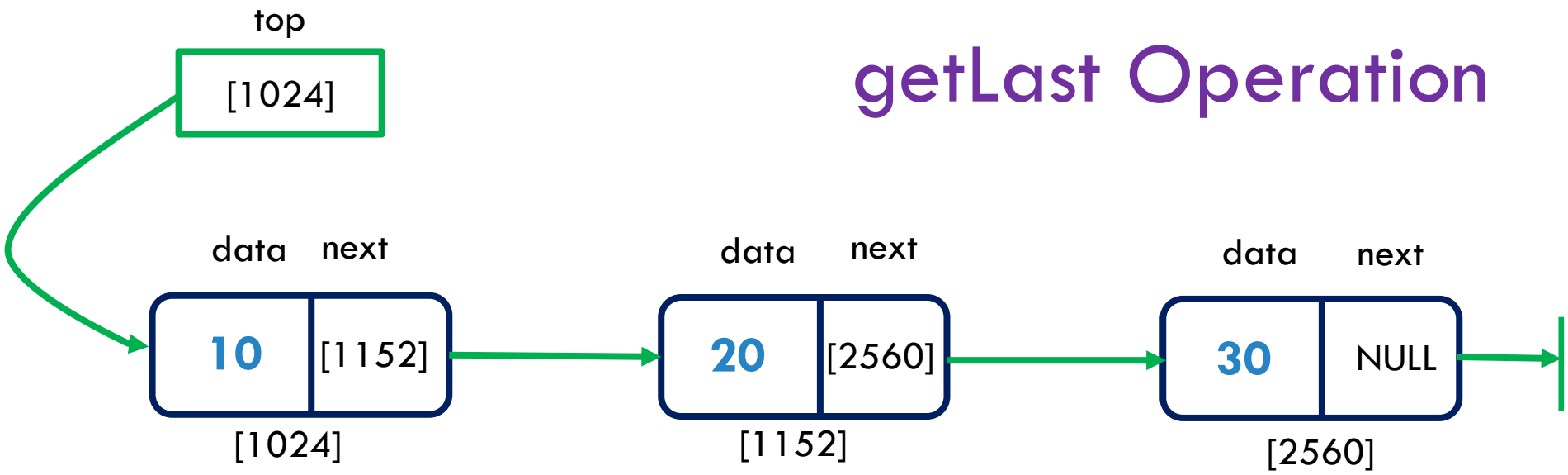



```
Node * node1, * node2, * node3, * node4;

node1 = createNode (10);
node2 = createNode (20);
node3 = createNode (30);
node4 = createNode (50);

node1->next = node2;
node2->next = node3;
```

LINKED LIST OPERATIONS

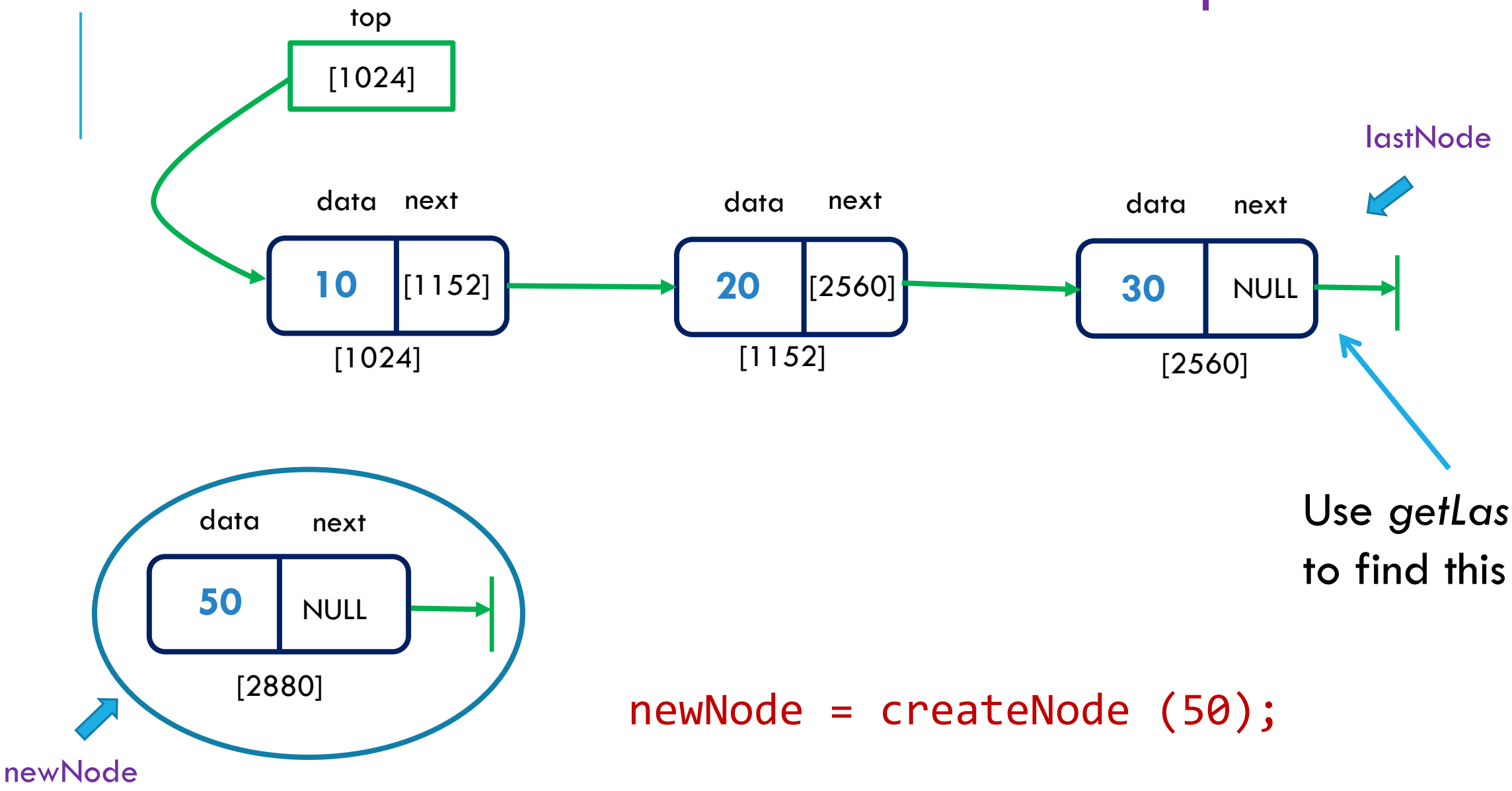
```
Node * insertAtHead (Node * top, int n)
Node * insertAtTail (Node * top, int n)
Node * insertSorted (Node * top, int n)
Node * getLast (Node * top)
int size (Node * top)
bool contains (Node * top, int key)
Node * deleteAtHead (Node * top)
bool isEmpty (Node * top)
void printList (Node * top)
```

getLast Operation



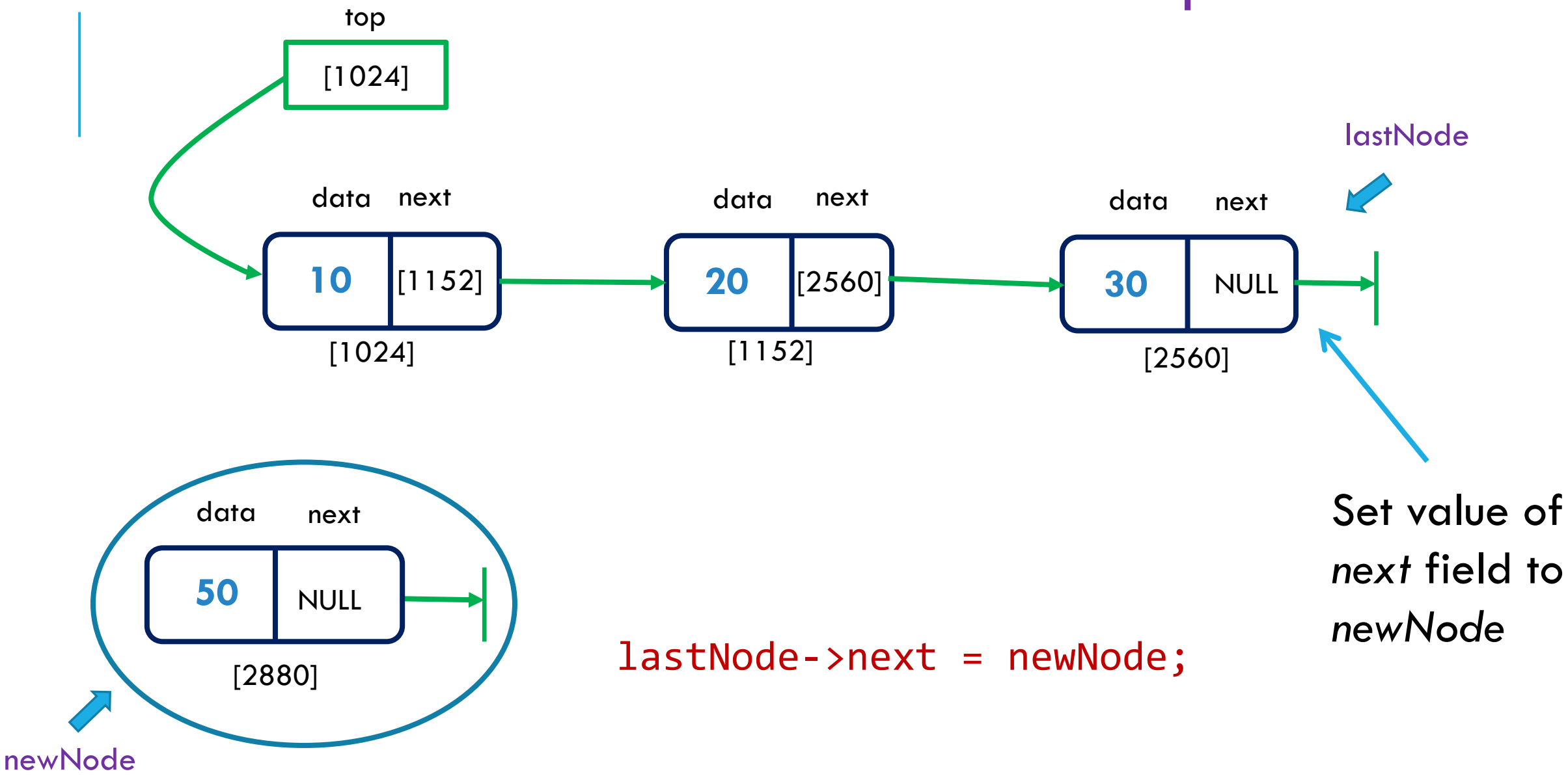

```
Node * getLast (Node * top) {
    Node * last;
    last = top;
    while (last != NULL) {
          last = last->next;
    }
    return last;
}
```

insertAtTail Operation

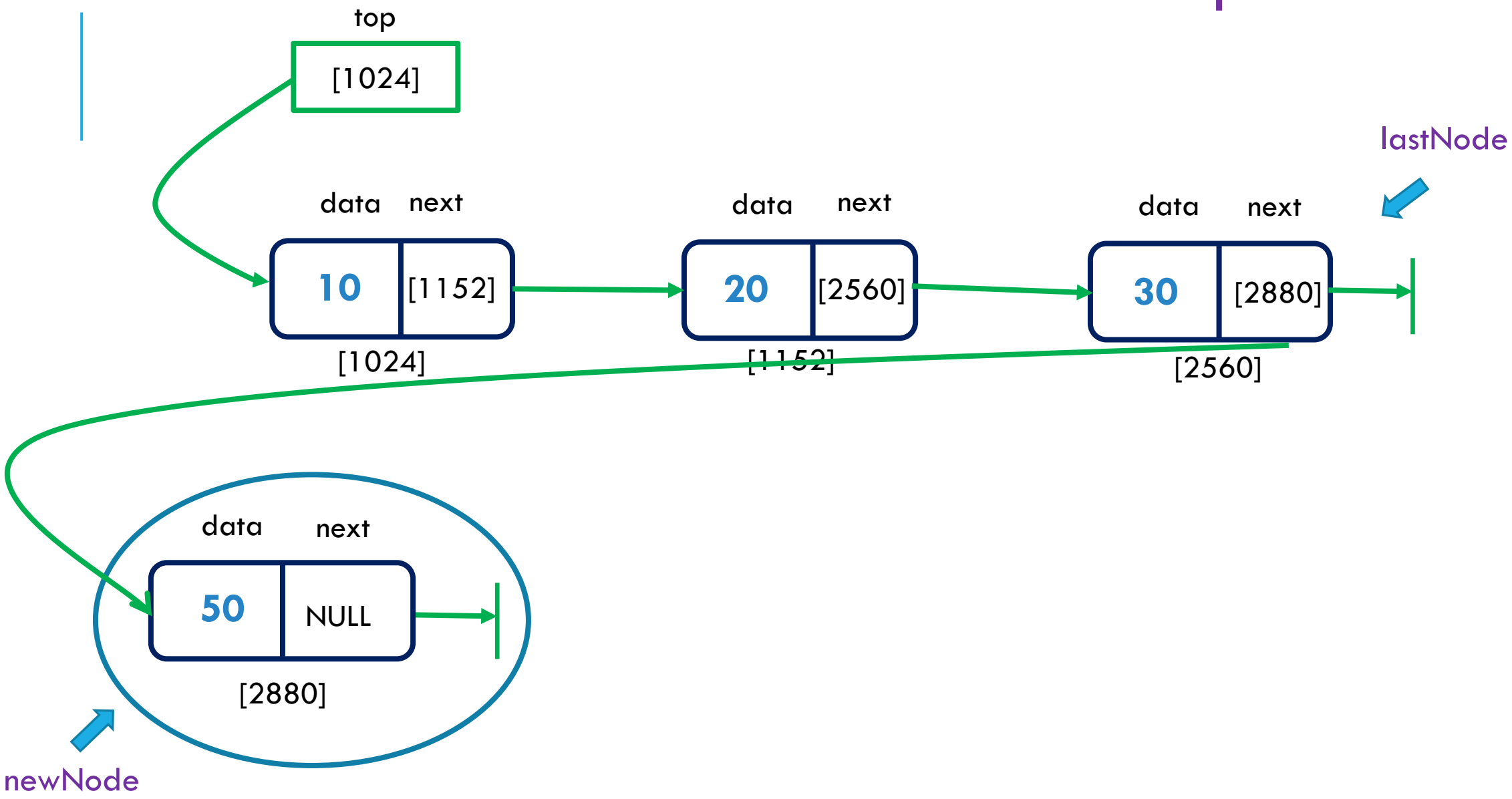



```
newNode = createNode (50);
```

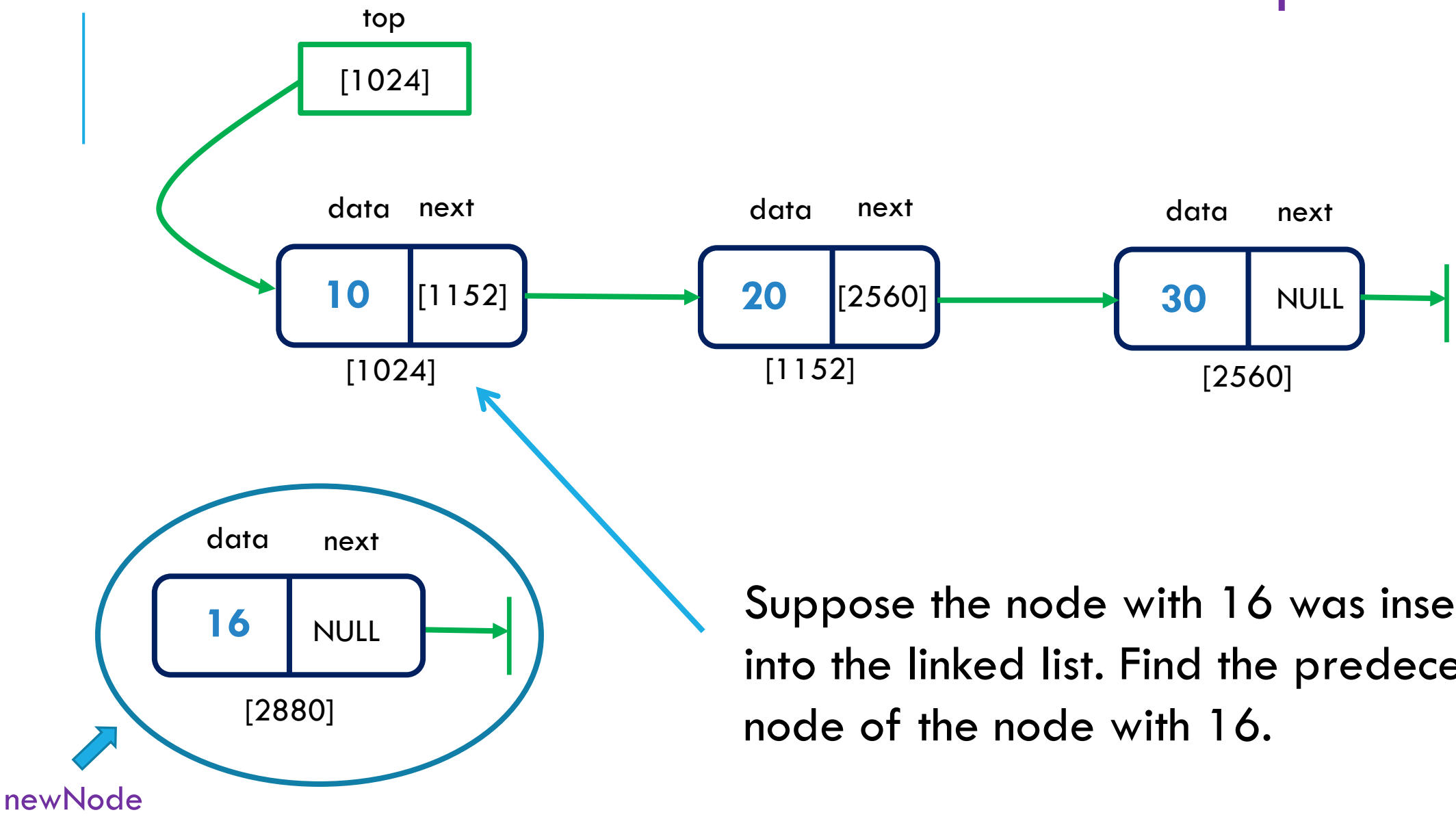
insertAtTail Operation



insertAtTail Operation

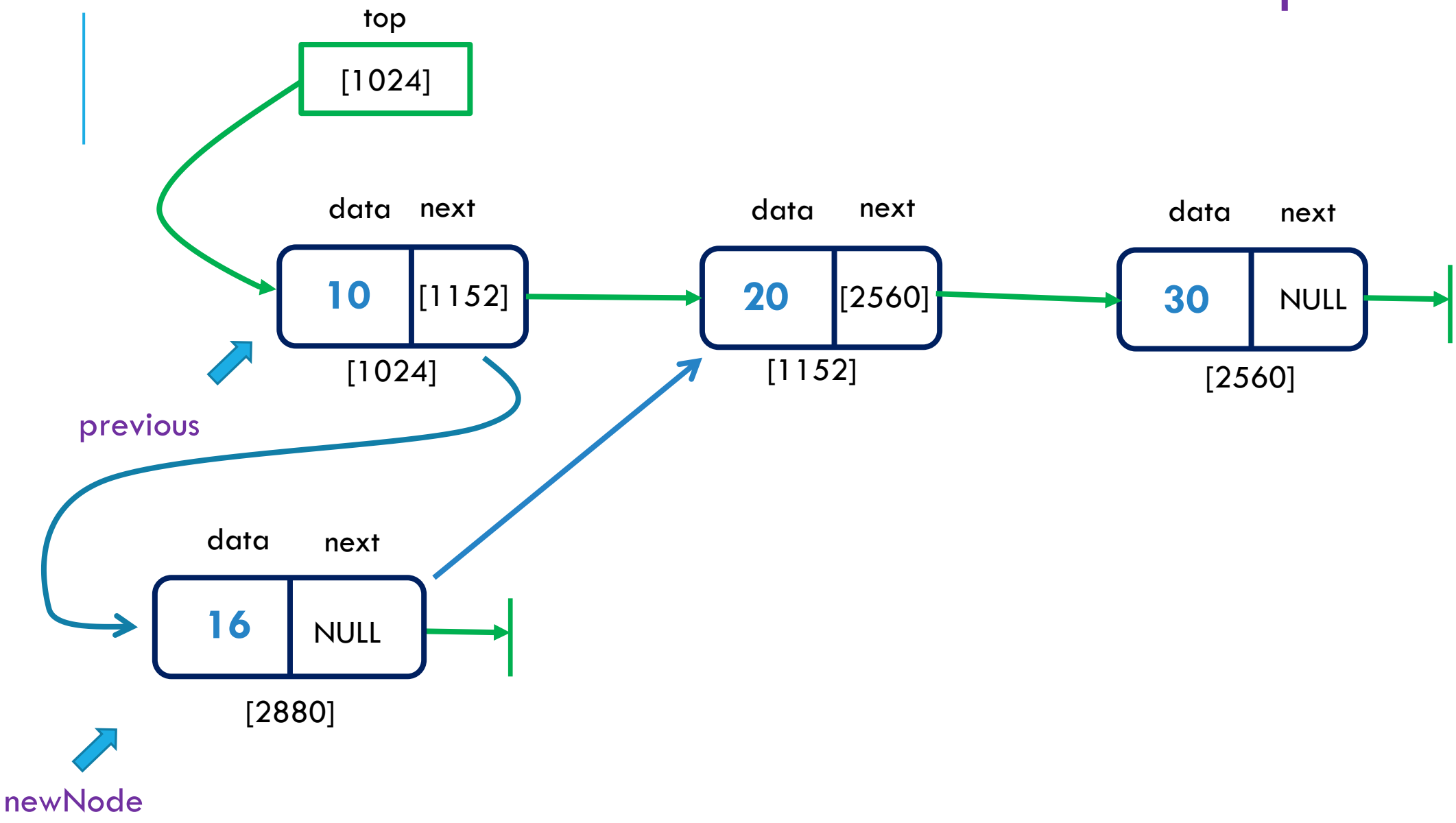


insertSorted Operation

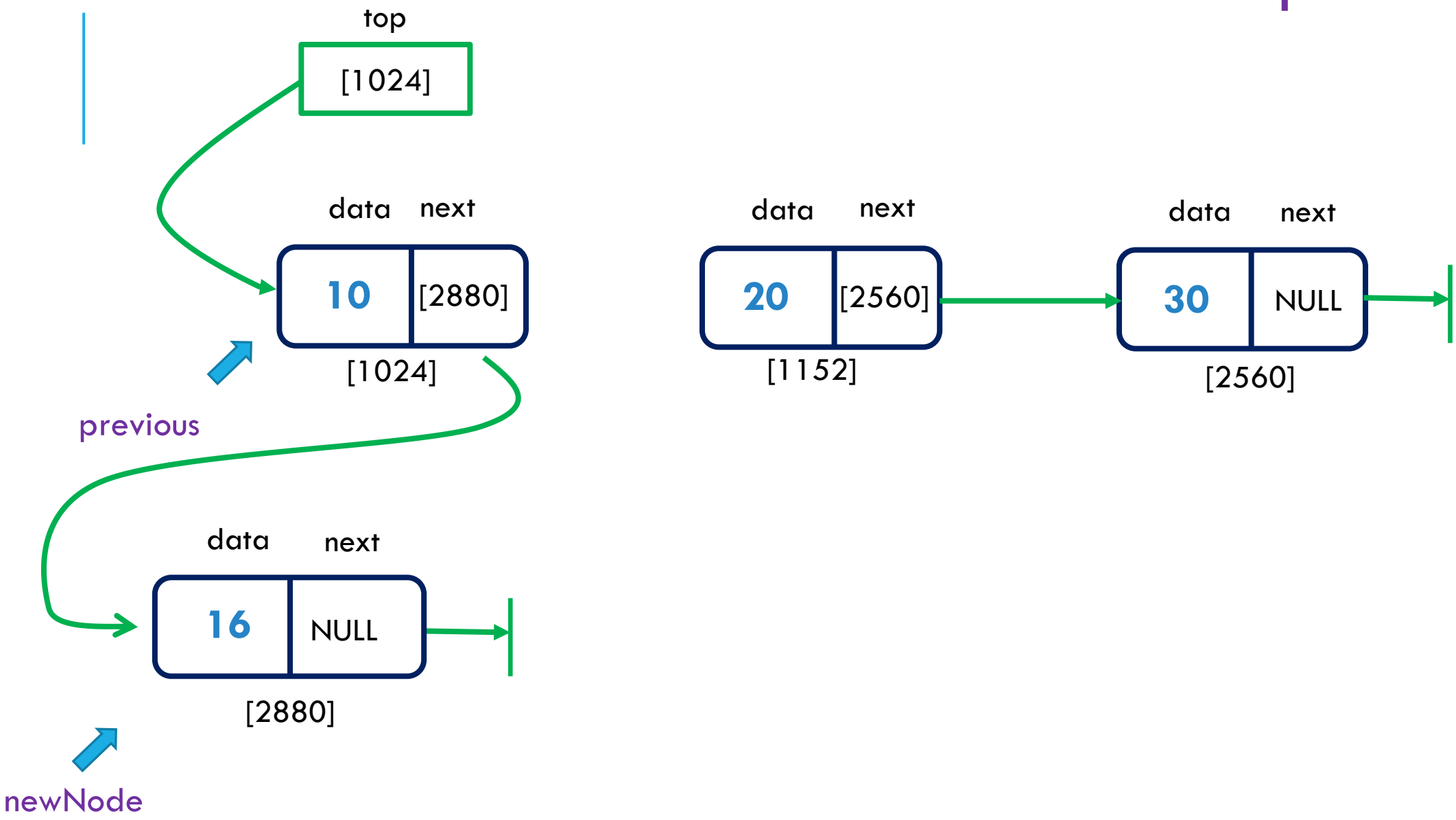


Suppose the node with 16 was inserted into the linked list. Find the predecessor node of the node with 16.

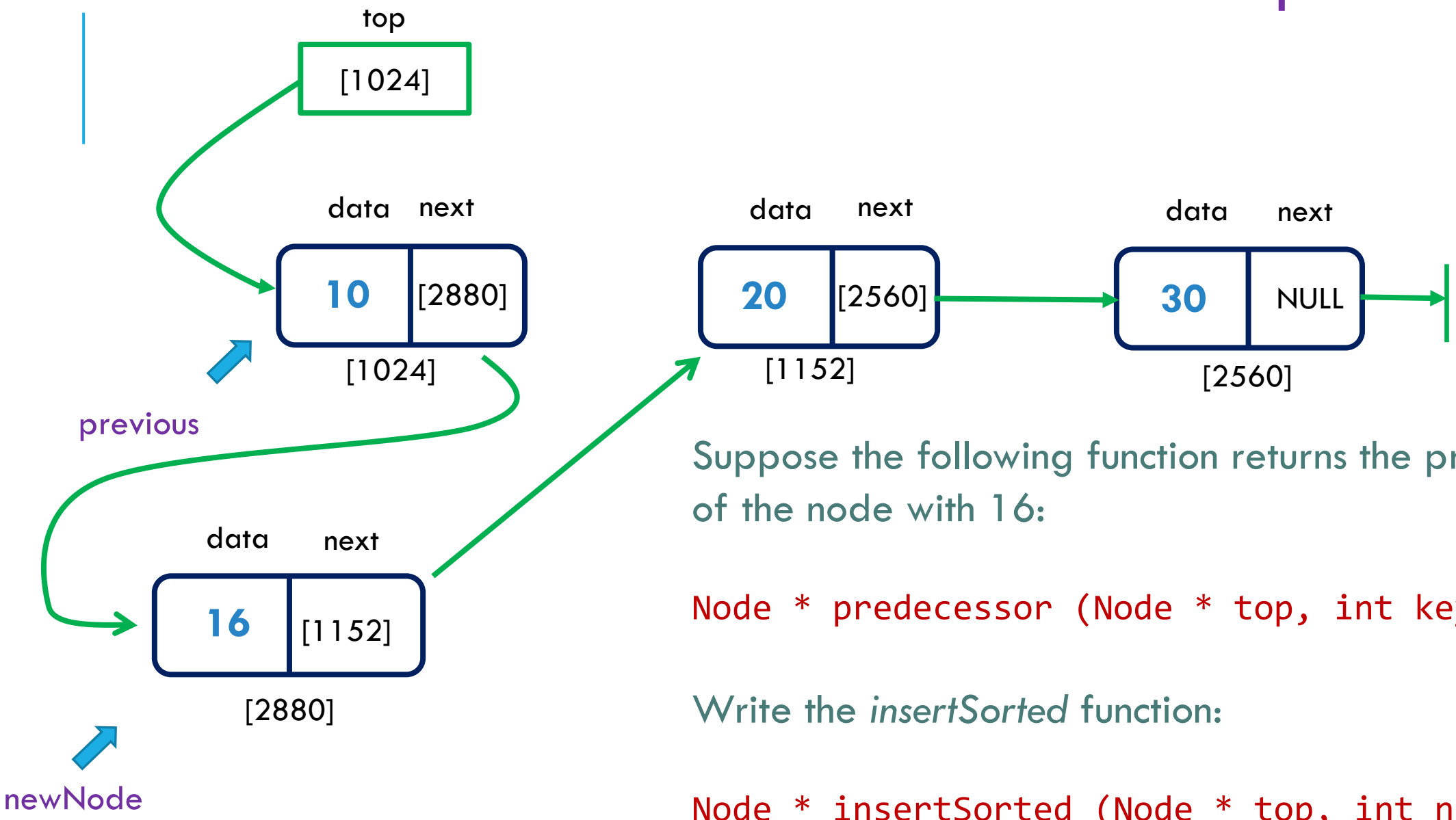
insertSorted Operation



insertSorted Operation



insertSorted Operation



Suppose the following function returns the predecessor of the node with 16:

```
Node * predecessor (Node * top, int key);
```

Write the *insertSorted* function:

```
Node * insertSorted (Node * top, int n);
```

WRITING THE INSERTSORTED FUNCTION

```
Node * newNode;
Node * previous;
previous = predecessor (top, 16);
newNode = createNode (16);
```

There are two cases to consider:

1. previous is NULL
2. previous is not NULL

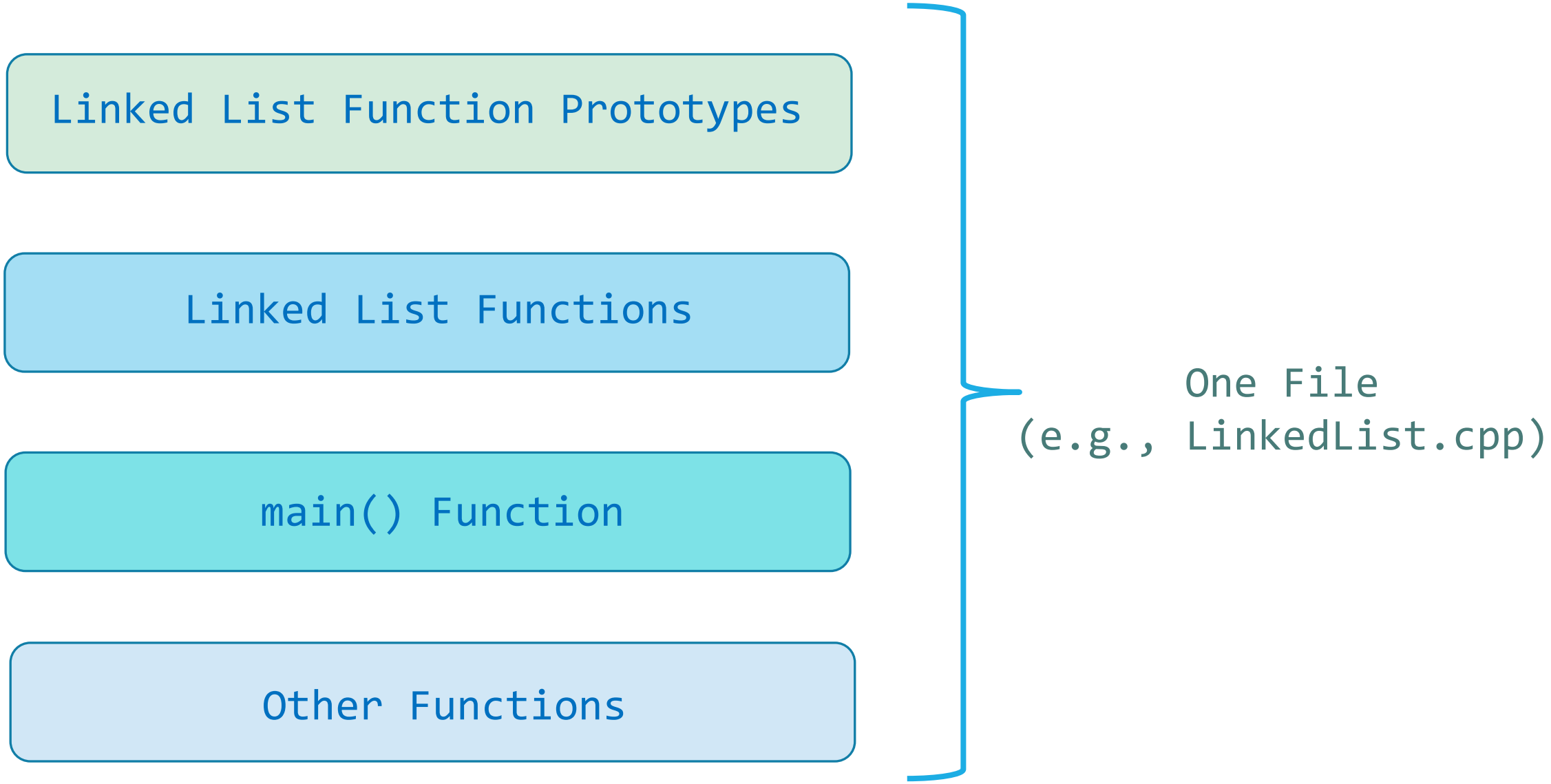
Case 1:

```
newNode->next = top;
return newNode; // new top
```

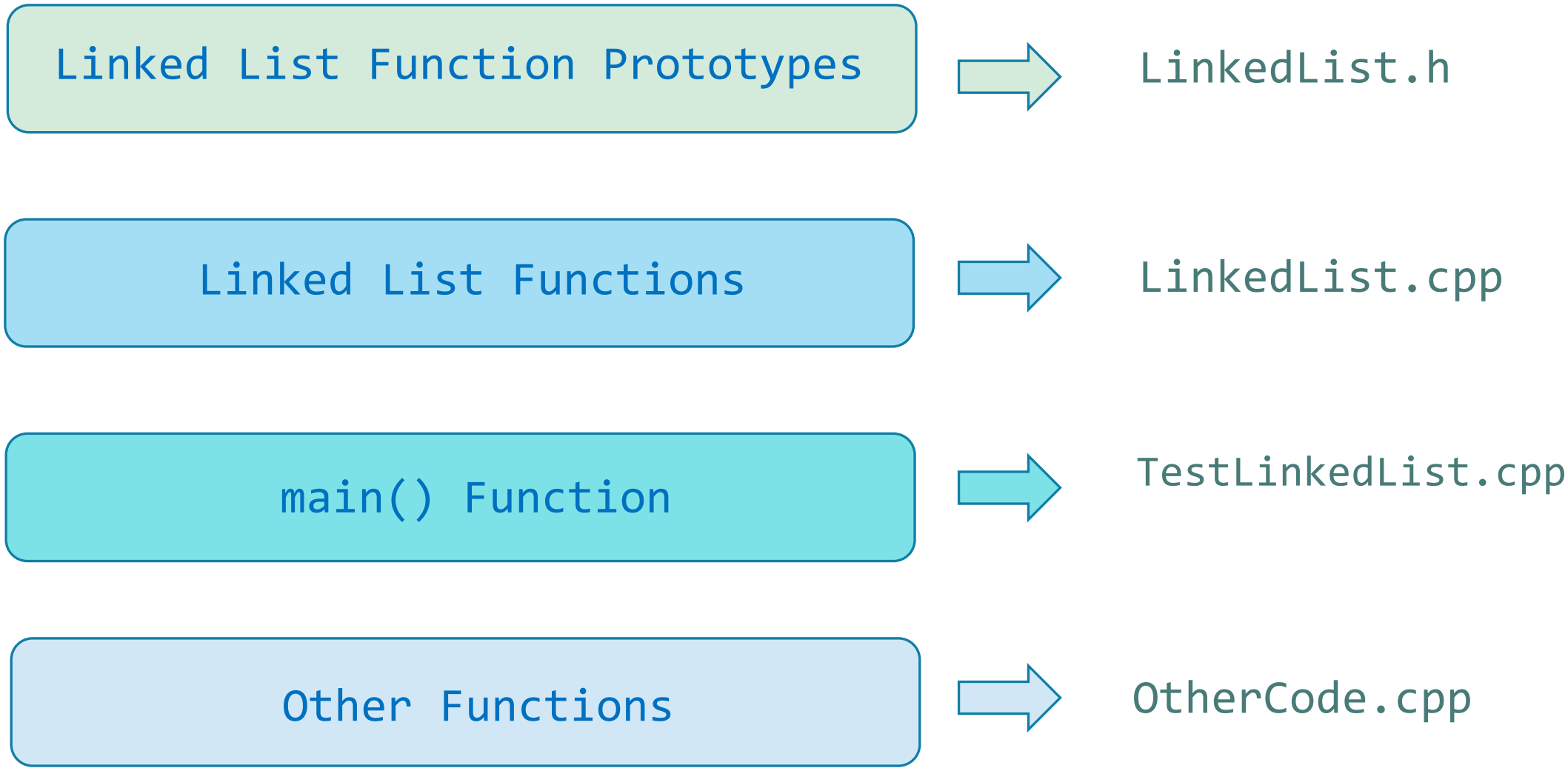
Case 2:

```
newNode->next = previous->next;
previous->next = newNode;
return top; // top unchanged
```

ORGANIZING CODE



ORGANIZING CODE



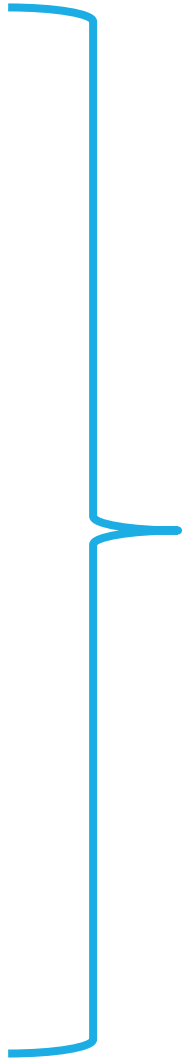
CREATING A PROJECT

LinkedList.h

LinkedList.cpp

TestLinkedList.cpp

OtherCode.cpp



A Project
(e.g., LinkedLists.dev)

Folder: LinkedList-Enhanced