Data Structures

CSCI 2270-202: REC 04

Sanskar Katiyar

Logistics

Office Hours at ECAE 128 (Aerospace Lobby)

Tuesday: 12:15 pm – 2:15 pm

Thursday: 5:00 pm – 6:00 pm

Friday: 1:30 pm – 3:30 pm

Recitation Materials *(Notes, Slides, Code, etc.)*

sanskarkatiyar.github.io/CSCI2270

Logistics

To submit an assignment, remember to hit:

"*Finish attempt*" button, and

"*Submit all and finish*" button

Hardcoding (policy listed in syllabus, under Autograder)

First offence: 0

Repeated offence: Honor Code sanctions

Recitation Outline

1. LL: Recap
2. LL: Traversal
3. LL: Insertion
4. LL: Deletion
5. Exercise

Recap: Linked List

Linked List

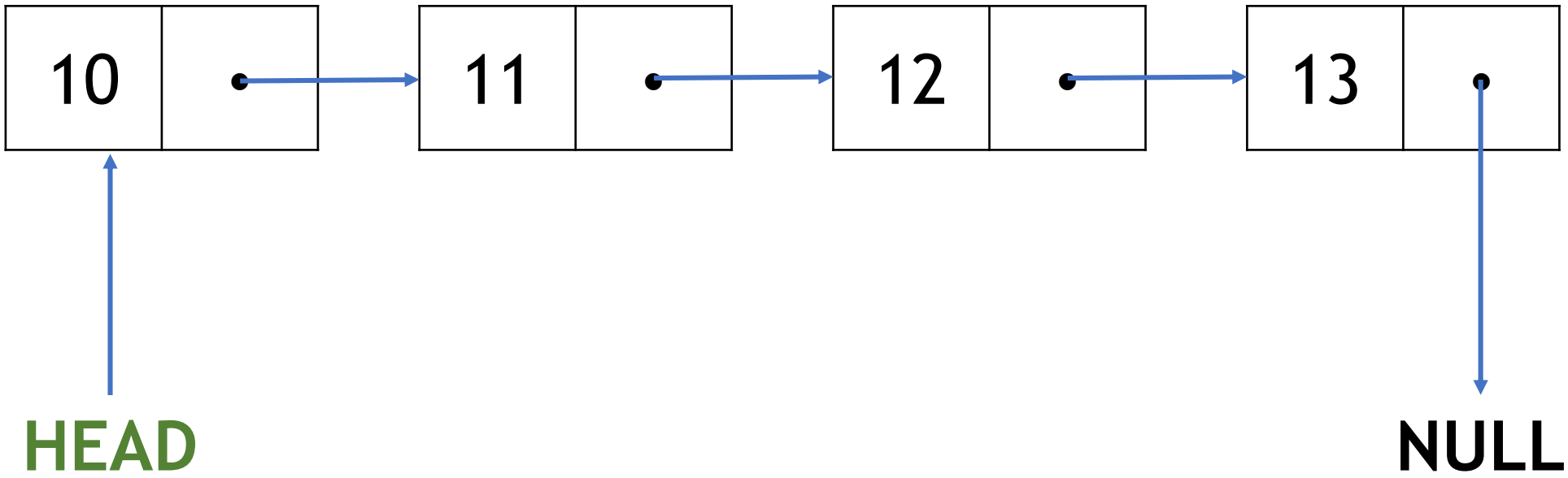
List is composed of ***nodes*** (and ***links***)

Need to track only ***head*** of the list

Last node's link is to **NULL**

Dynamically allocated on heap; But unlike arrays, ***not contiguous*** in memory

Linked List

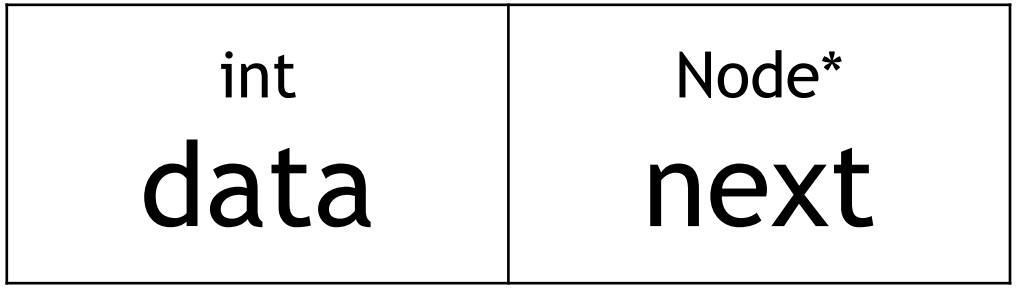


Linked List

Each ***Node*** consists of some data, not necessarily just one variable, or one type

Often the ***Link*** is included as a data member of the node

Recall: **Structures and Classes** allow us to define custom data types and objects



Linked List

```
struct Node
{
    int data;
    Node *next;
};
```

```
class Node
{
public:
    int data;
    Node *next;
};
```

Linked List

```
struct Country
{
   string name;
   string message;
   int numberMessages;
   Country *next;
};
```

Data members of a node

Link to next node

Tips to solve LL problems

Visualize!

Draw the list and track the changes in links.

Keep track of node(s)

Do not lose the HEAD node!

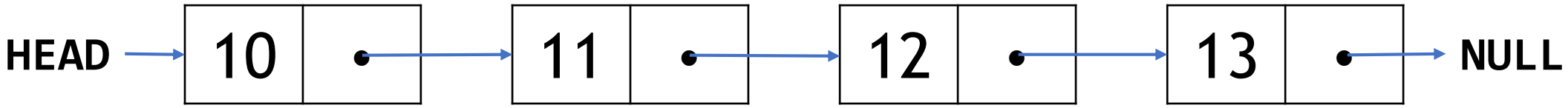
Take care of node pointers important for rewiring.

Temporary pointers are handy.

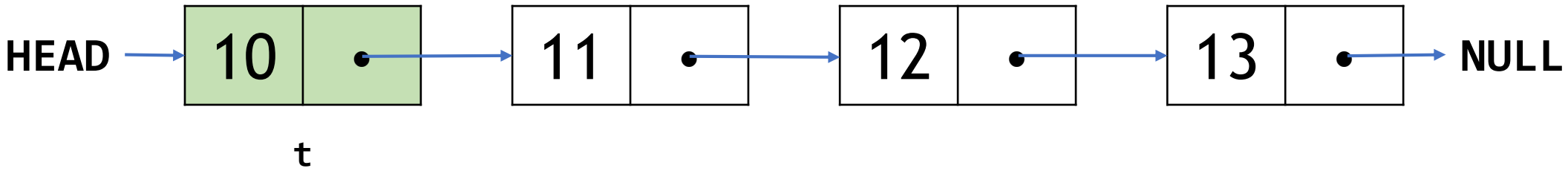
Traversal: Linked List

Traversal: Example

```
struct Node
{
    int data;
    Node* next;
};
```



Traversal: Example

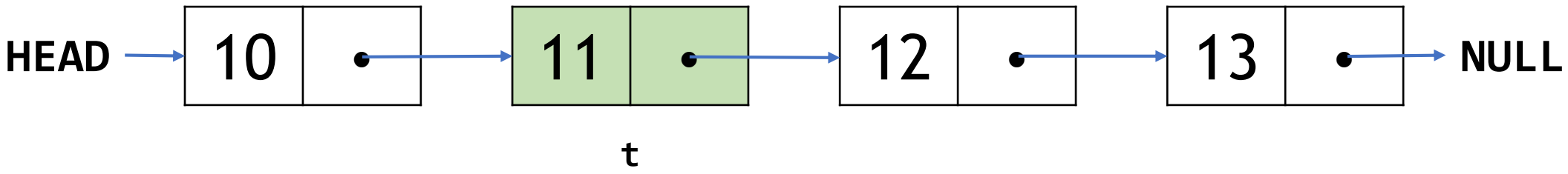



```
Node *t = HEAD;
```

```
cout << t->data;
```

10

Traversal: Example

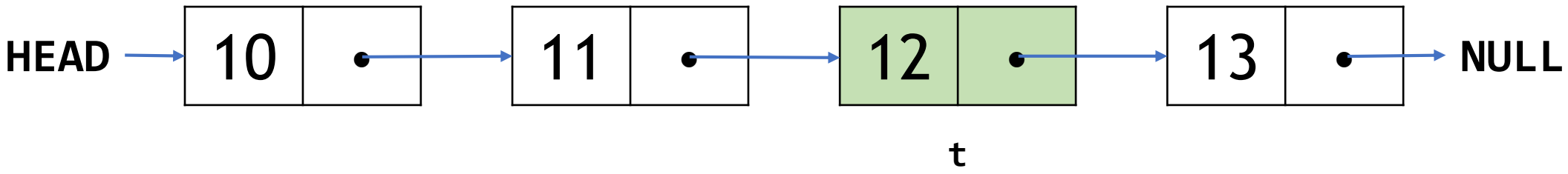



```
Node *t = HEAD;
t = t->next;
```

```
cout << t->data;
```

11

Traversal: Example

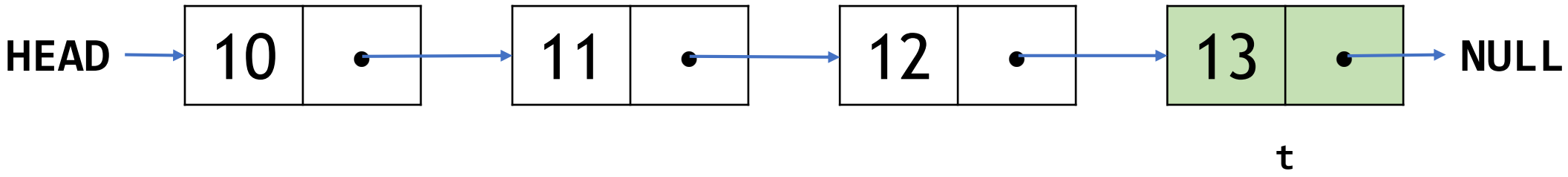



```
Node *t = HEAD;
t = t->next;
t = t->next;
```

```
cout << t->data;
```

12

Traversal: Example



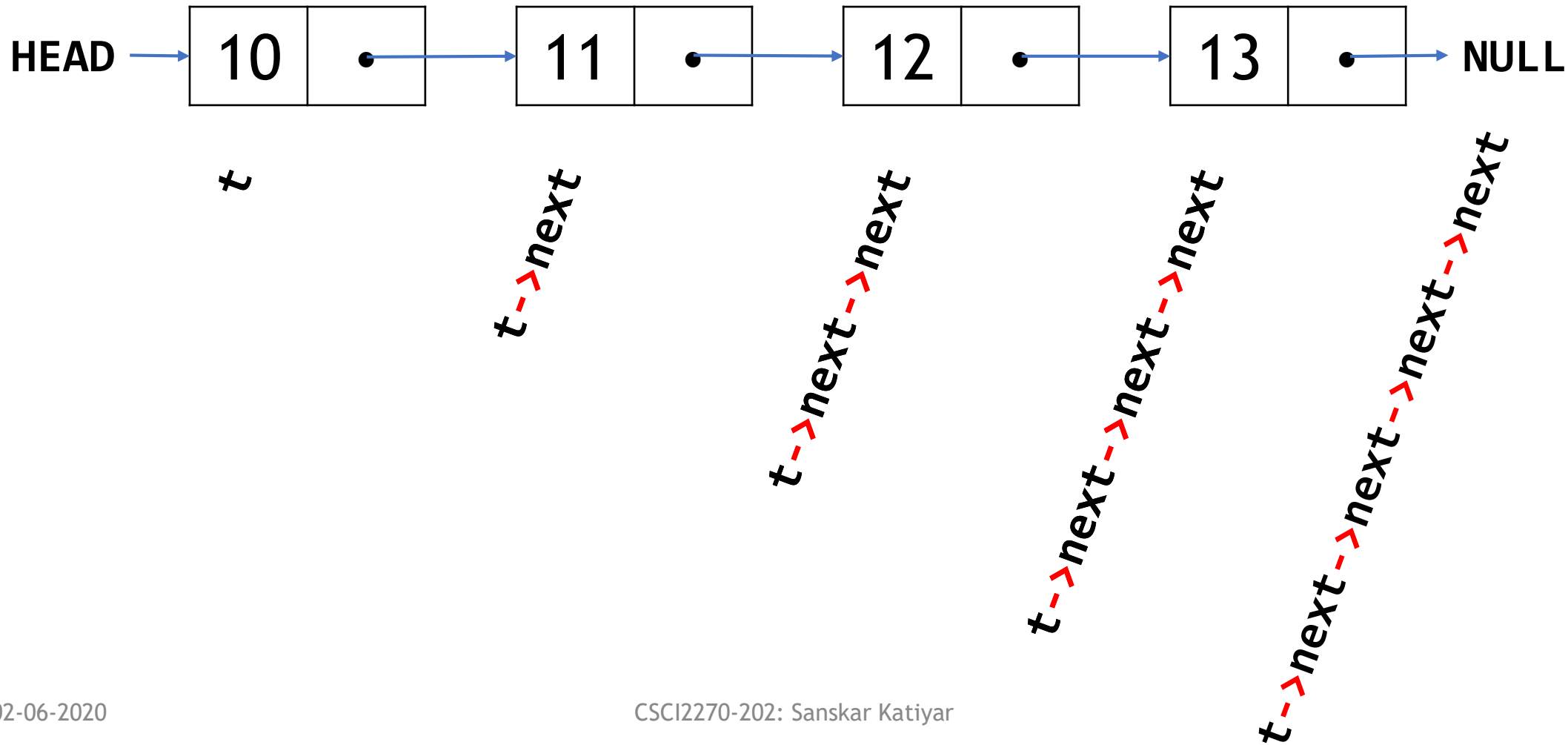

```
Node *t = HEAD;
t = t->next;
t = t->next;
t = t->next;
```

```
cout << t->data;
```

13

Traversal: Example

```
Node* t = HEAD;
```



Traversal: Implementation

Code

```
Node *t = HEAD;


while(t != NULL)
{
 cout << t->data << endl;
 t = t->next;
}
```

Output

```
10
11
12
13
```

Traversal: Roundup

Check the initial, final and update conditions for traversal

Can you modify the traversal algorithm for search?

Code File(s): `traversal.cpp, search.cpp`

Traversal, Search: Function(s)

Recitation 4 Exercise -> LinkedList.cpp

Traversal: `printList()`

Simple traversal and printing

Search: `searchList(int key)`

Modify traversal for search

Insertion: Linked List

Insertion: Steps, Scenarios

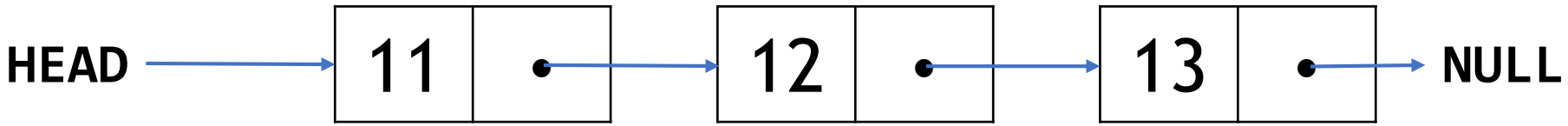
1. Create the node *(you want to insert)*

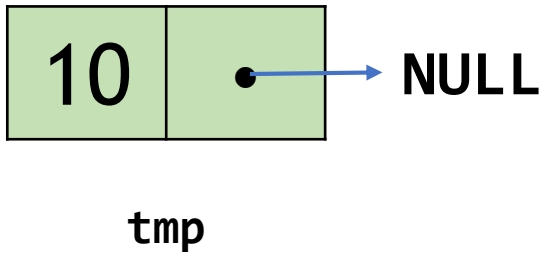
2. Insert the node at:

    Case 1: HEAD

    Case 2: Any other valid place

Insertion: at Head

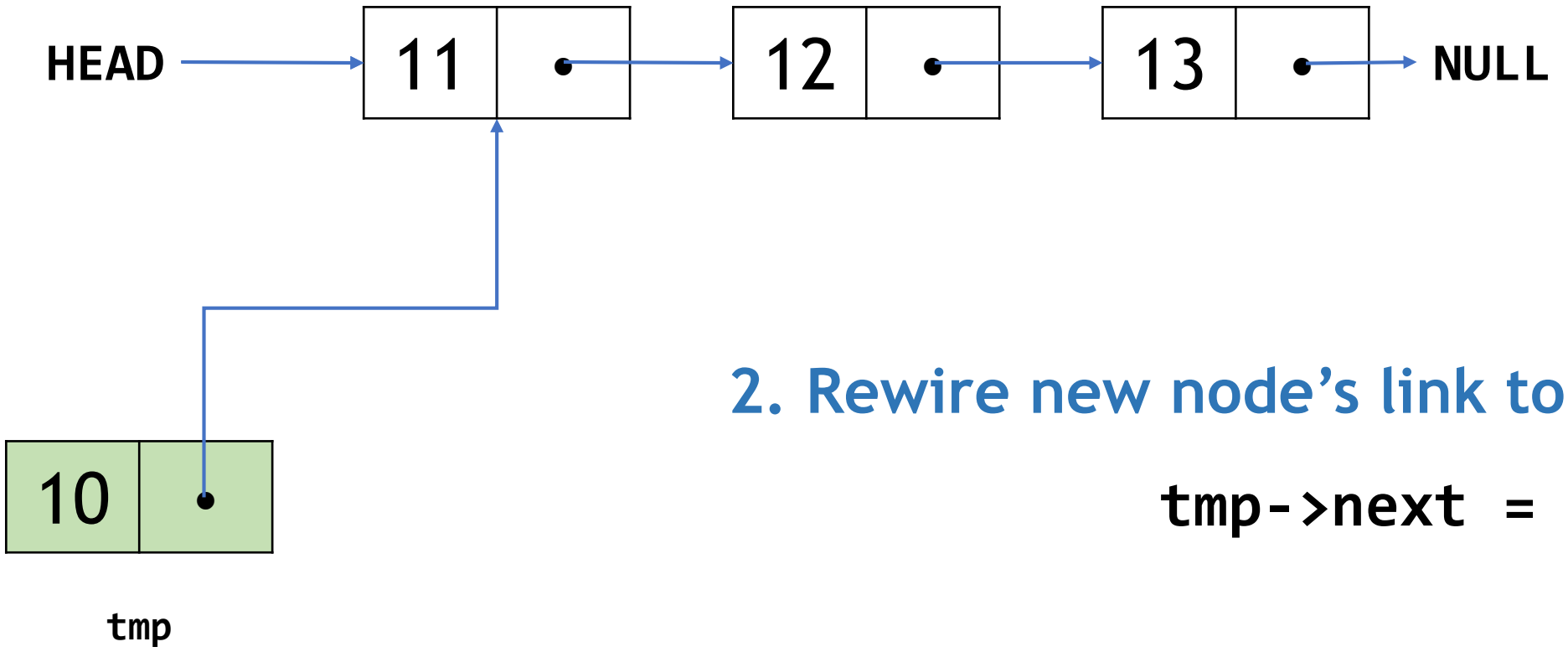




1. Create new node (to be inserted)

```
Node *tmp = new Node({10, NULL});
```

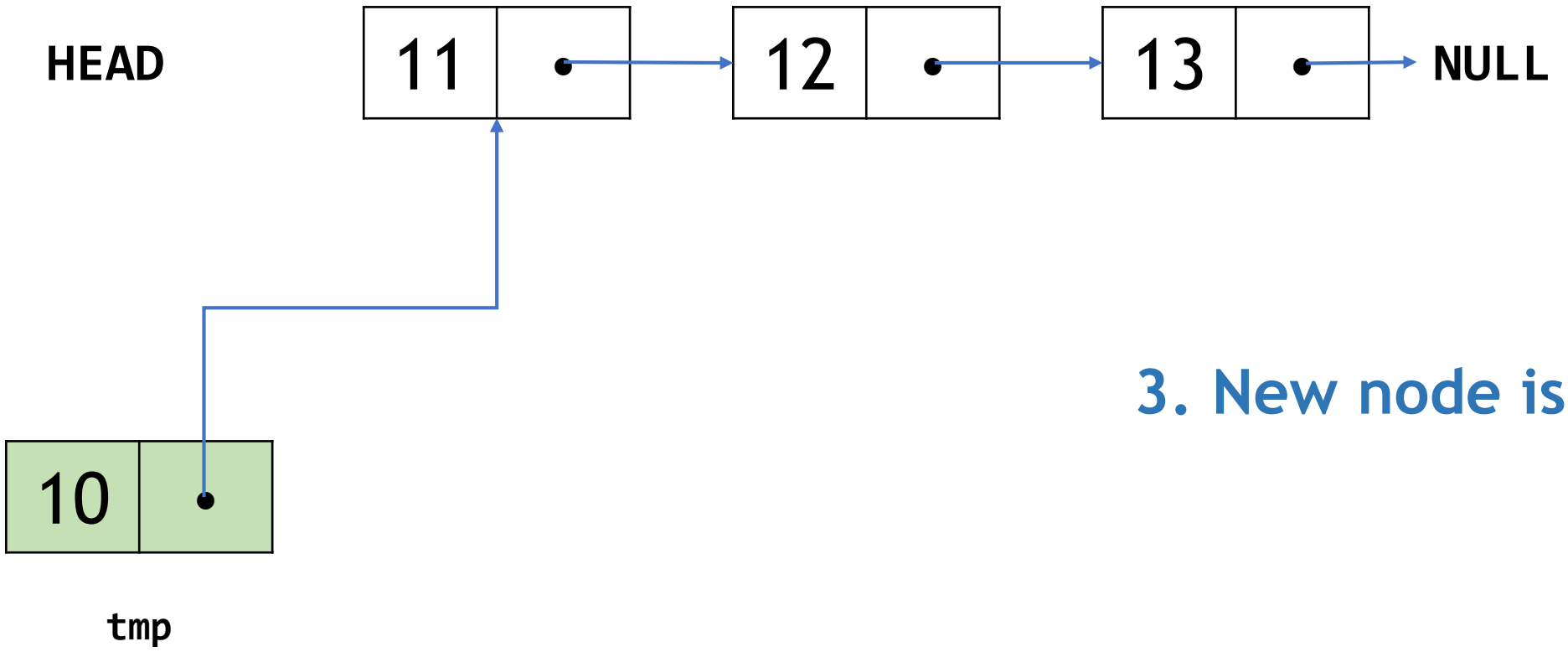
Insertion: at Head



2. Rewire new node's link to HEAD

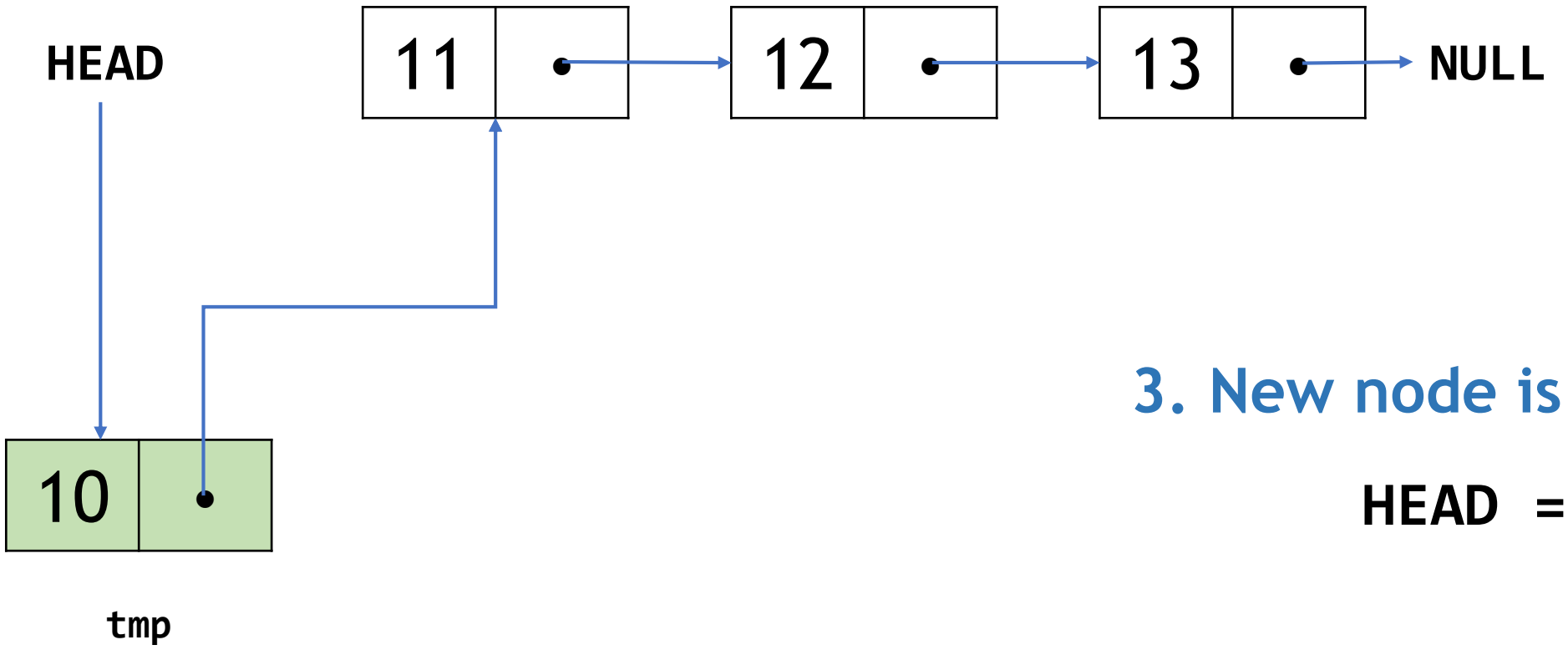
```
tmp->next = HEAD;
```

Insertion: at Head



3. New node is HEAD

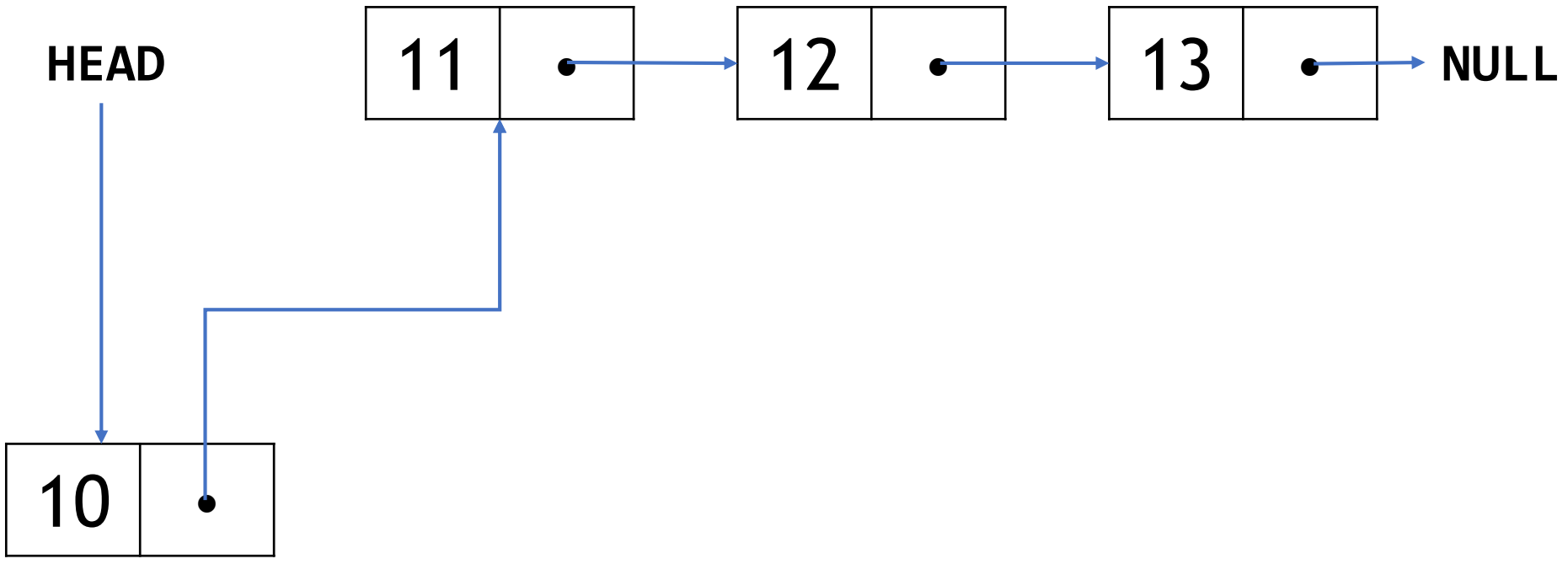
Insertion: at Head



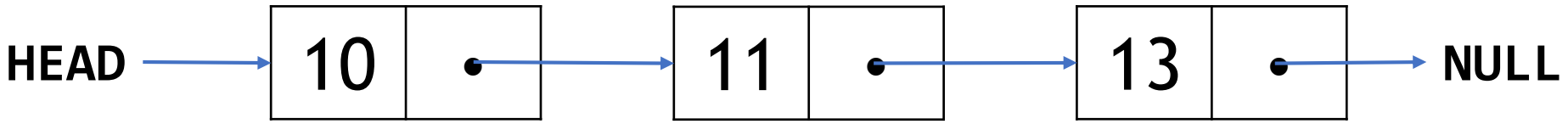
3. New node is HEAD

```
HEAD = tmp;
```

Insertion: at Head

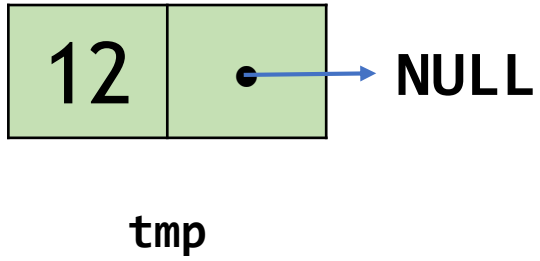


Insertion: at any other valid position

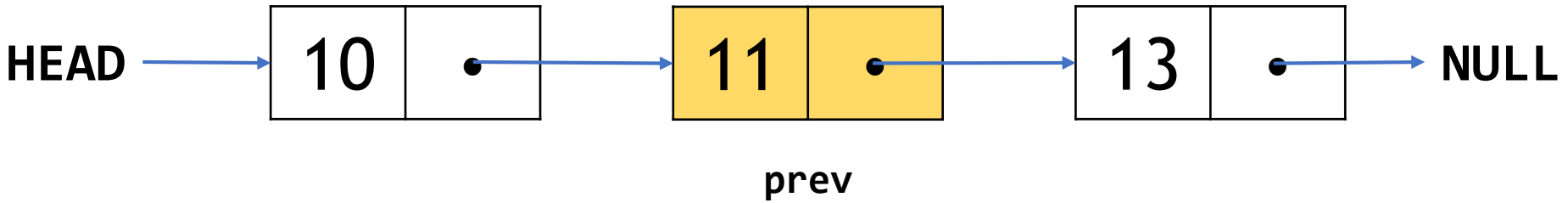


1. Create new node (to be inserted)

```
Node *tmp = new Node({12, NULL});
```

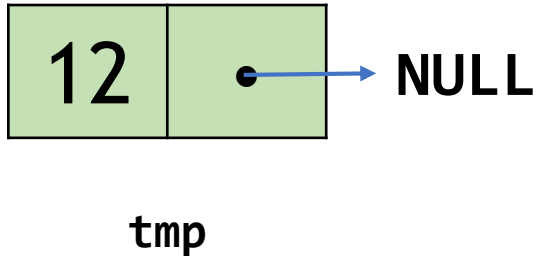


Insertion: at any other valid position

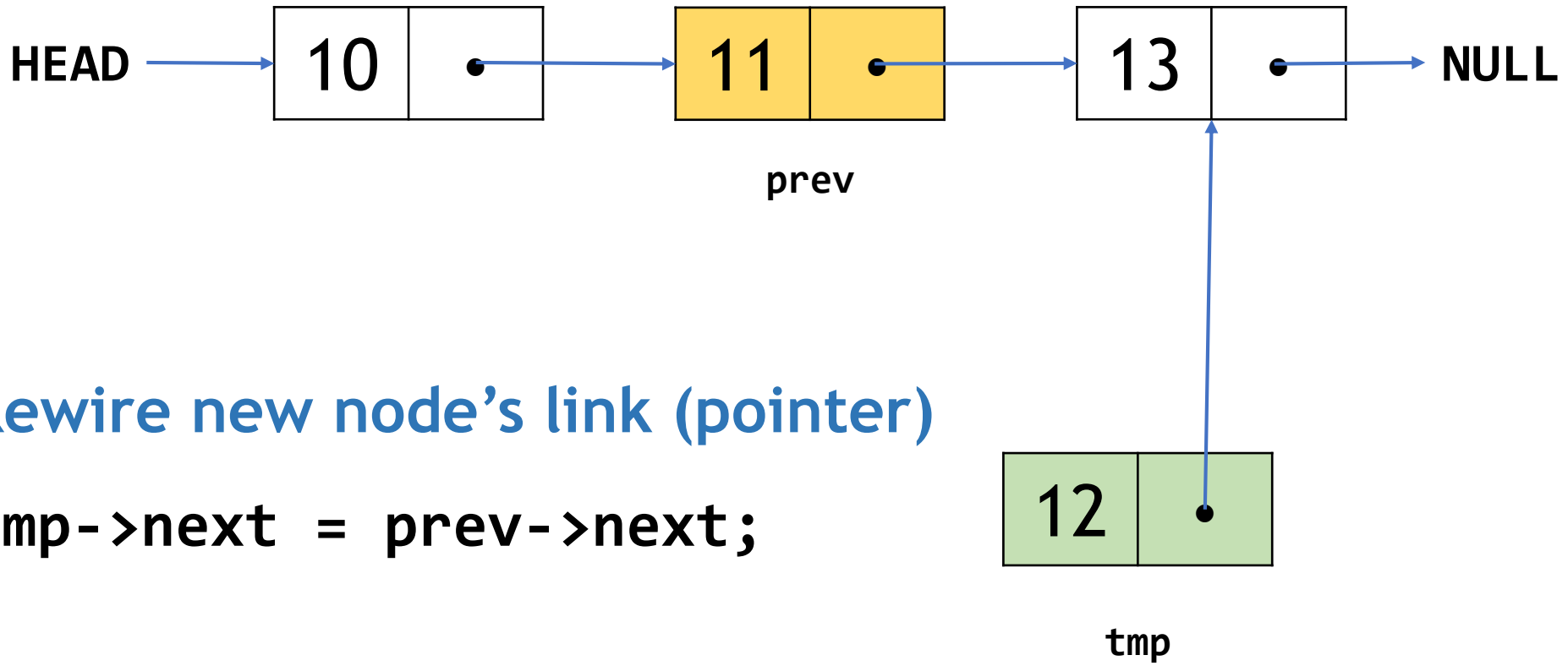


2. Find (pointer to) node to insert after

```
Node *prev = HEAD->next; // e.g.
```



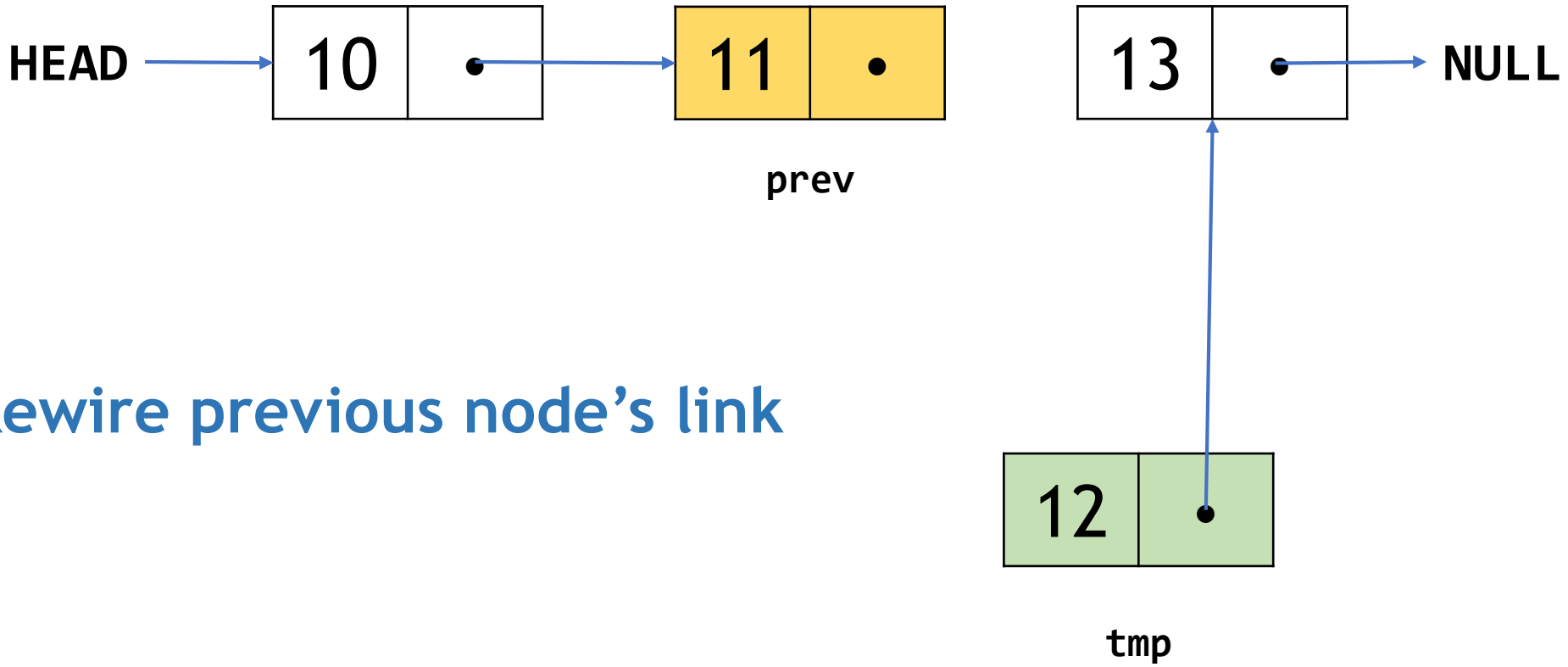
Insertion: at any other valid position



3. Rewire new node's link (pointer)

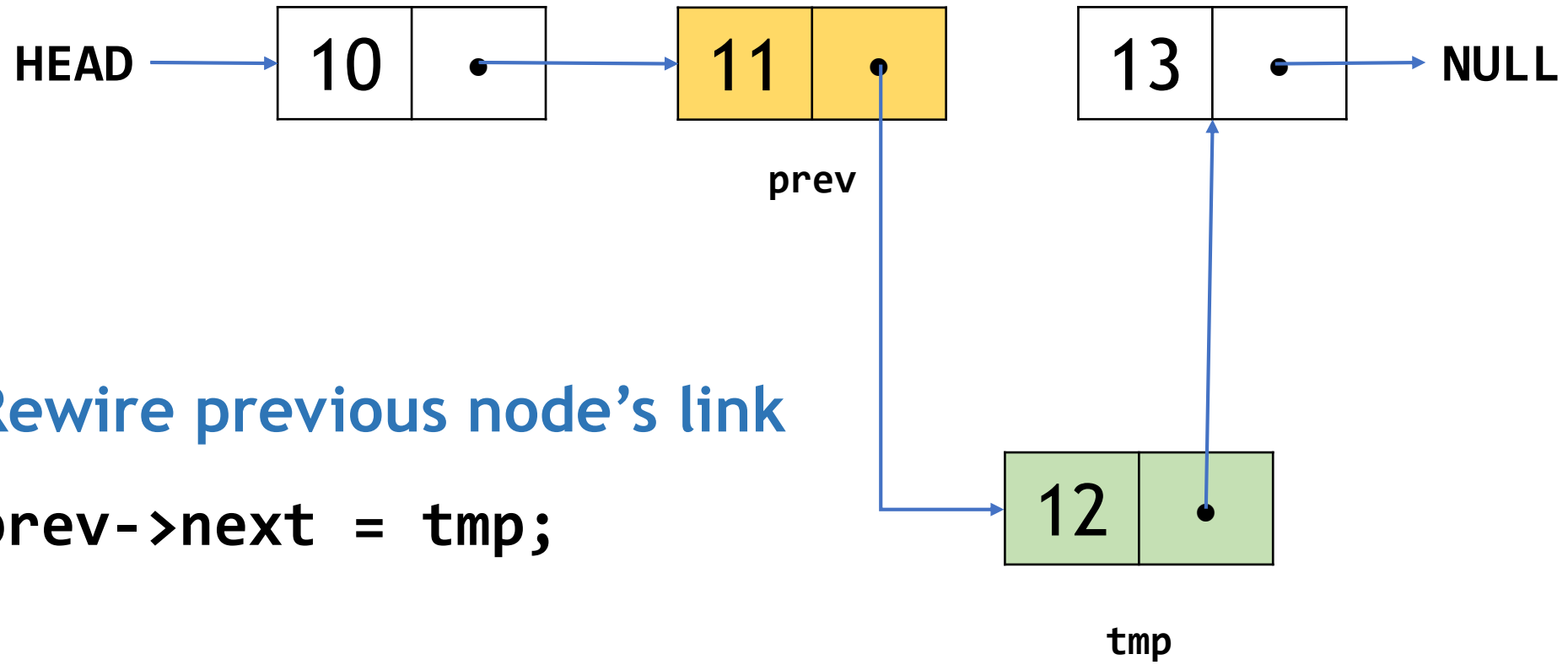
```
tmp->next = prev->next;
```

Insertion: at any other valid position



4. Rewire previous node's link

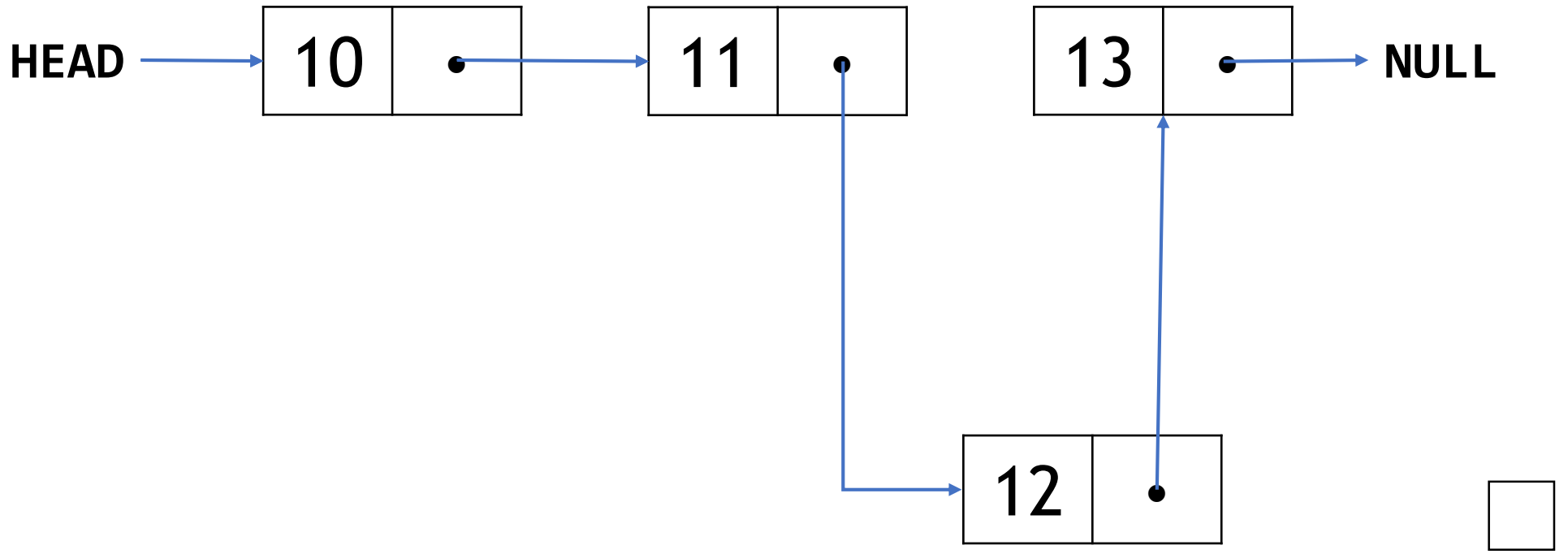
Insertion: at any other valid position



4. Rewire previous node's link

```
prev->next = tmp;
```

Insertion: at any other valid position



Insertion: at end

Same as the general case, *except Step 2*.

How to find the last node?

```
Node * t = HEAD;

while (t->next != NULL) {
    t = t->next;
}
```

Insertion: Roundup

Take care of all the cases. Don’t lose the HEAD pointer.

Code File(s): `insertion.cpp`

How to write Insertion as a function?

Recitation 4 Exercise -> LinkedList.cpp

```
insert(Node *prev, int newKey);
```

Deletion: Linked List

Deletion: Steps, Scenarios

1. Find & Create temp pointer to node, to be deleted

2. Simultaneously maintain the (pointer to) **previous** node *(If reqd)*

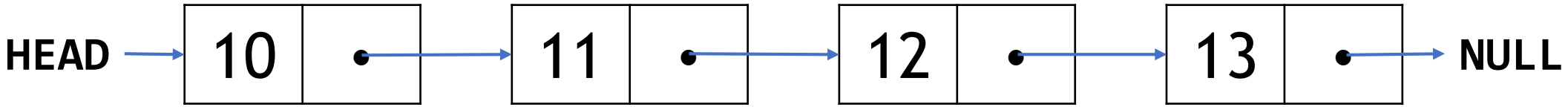
3. Delete the:

    Case 1: HEAD node

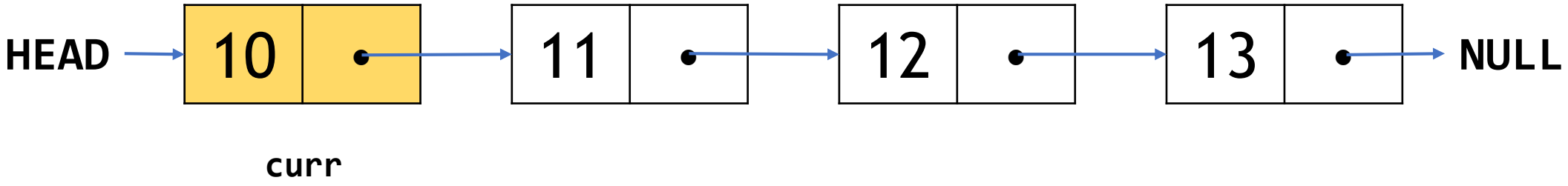
    Case 2: Any other valid node

    Case 3: Entire List

Deletion: HEAD node



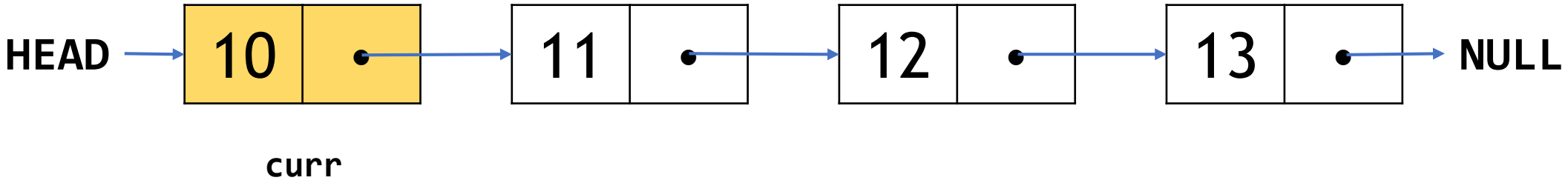
Deletion: HEAD node



1. Find & Create temp (pointer to) node, to be deleted

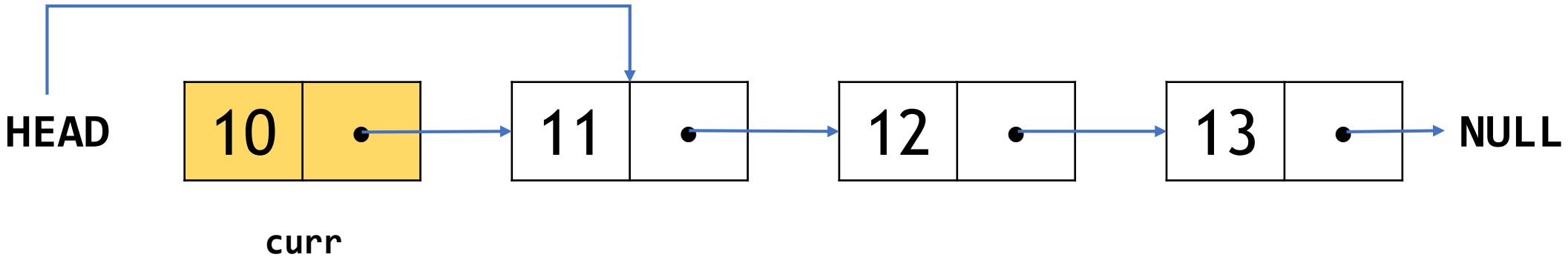
```
Node* curr = HEAD;
```

Deletion: HEAD node



2. Modify HEAD pointer to next node

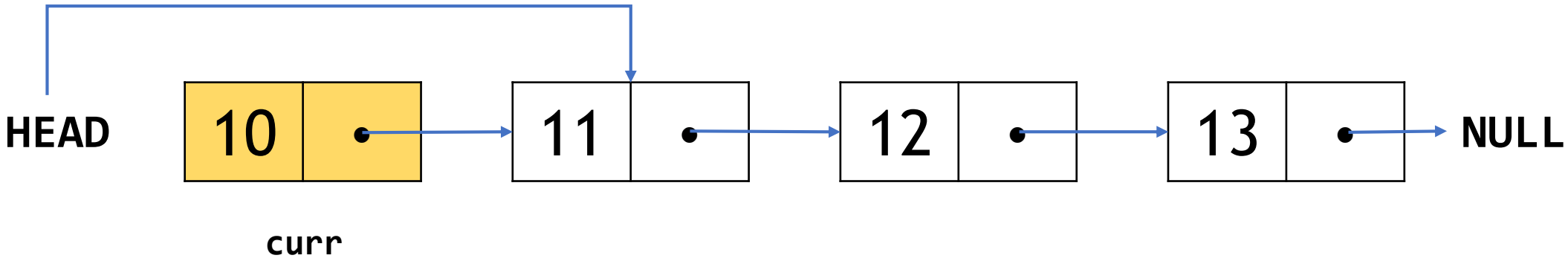
Deletion: HEAD node



2. Modify HEAD pointer to next node

```
HEAD = HEAD->next;
```

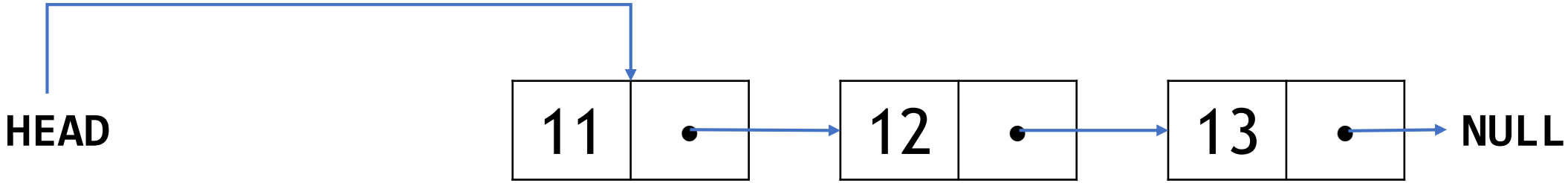
Deletion: HEAD node

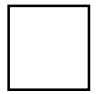


3. Free the node to be deleted

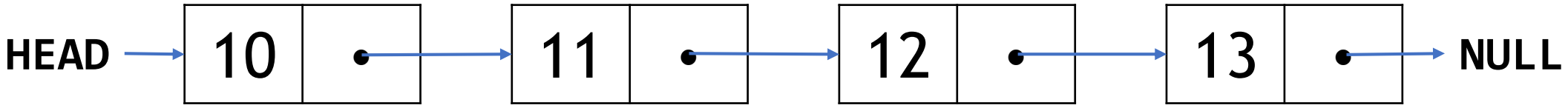
```
delete curr;
```

Deletion: HEAD node

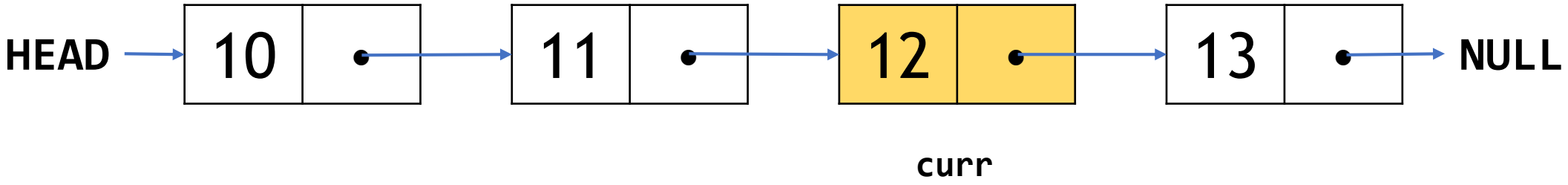




Deletion: any other valid node



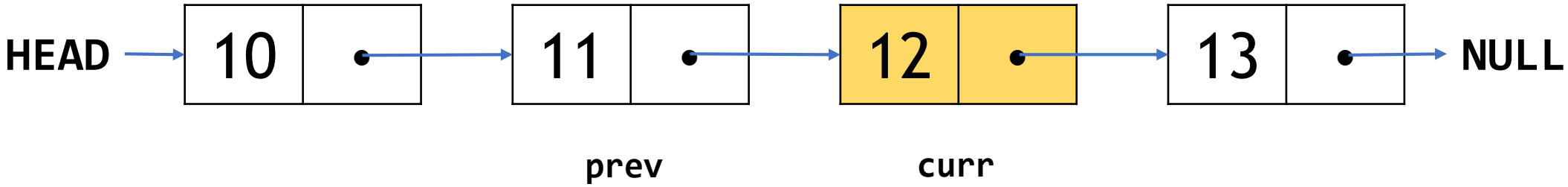
Deletion: any other valid node



1. Find & Create temp (pointer to) node, to be deleted

```
Node* curr = HEAD->next->next; // e.g.
```

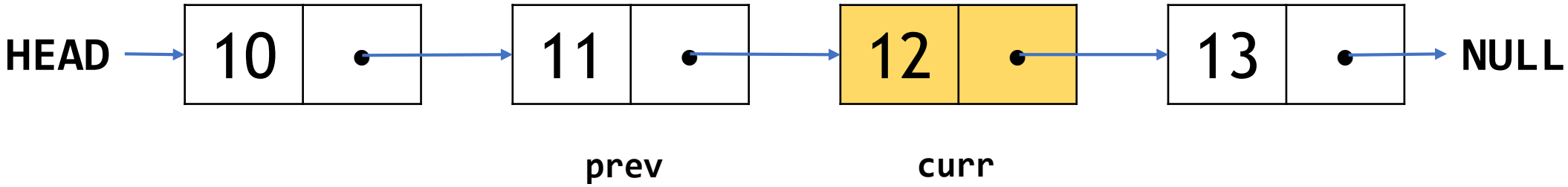
Deletion: any other valid node



2. Simultaneously, maintain (pointer to) previous node

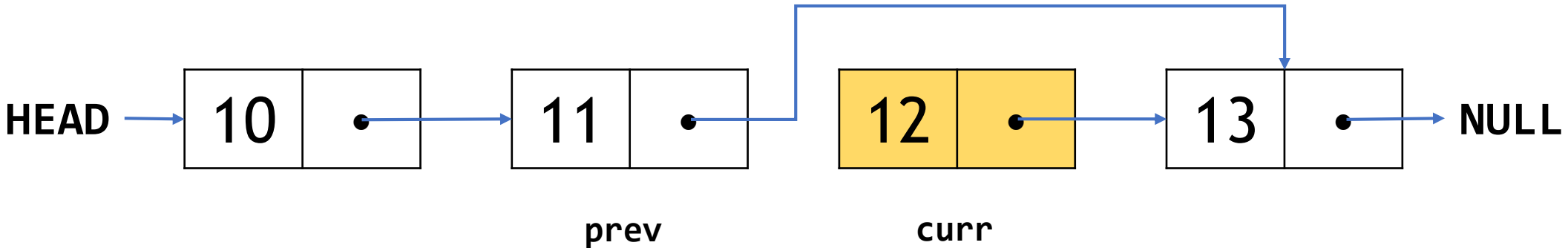
```
Node* prev = HEAD->next; // e.g.
```

Deletion: any other valid node



3. Rewire previous node's link

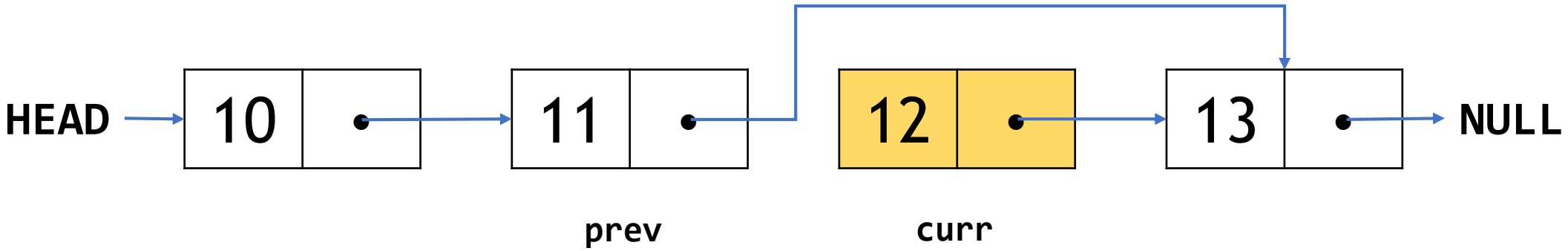
Deletion: any other valid node



3. Rewire previous node's link

```
prev->next = curr->next;
```

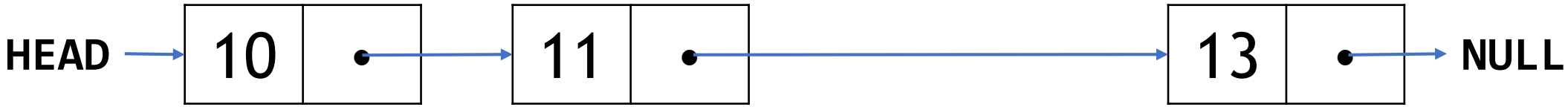
Deletion: any other valid node

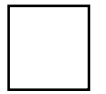


4. Free the node to be deleted

```
delete curr;
```

Deletion: any other valid node





Deletion: `curr, prev` pointers

How to maintain `curr, prev` **pointers simultaneously?**

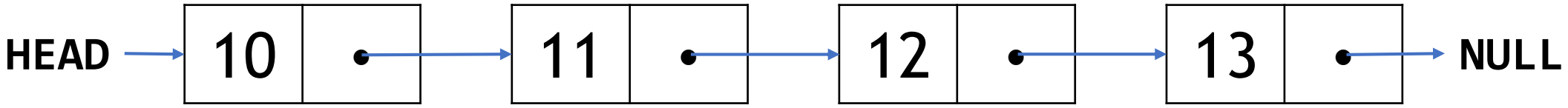
```
Node *curr = HEAD;
Node *prev = NULL;

while (curr && curr->data != someValue) {
    prev = curr;
    curr = curr->next;
}
```

Deletion: Entire List

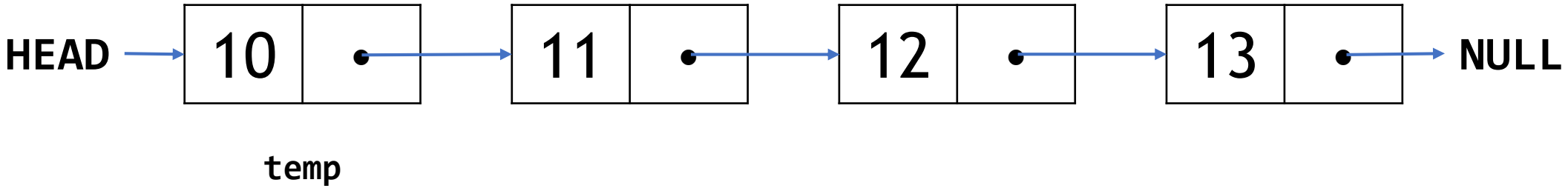
A simple approach: Delete the head node until NULL

```
Node* temp = head;

while(head) {
     temp = head;
     head = head->next;
     delete temp;
}

temp = NULL;
```

Deletion: Entire List



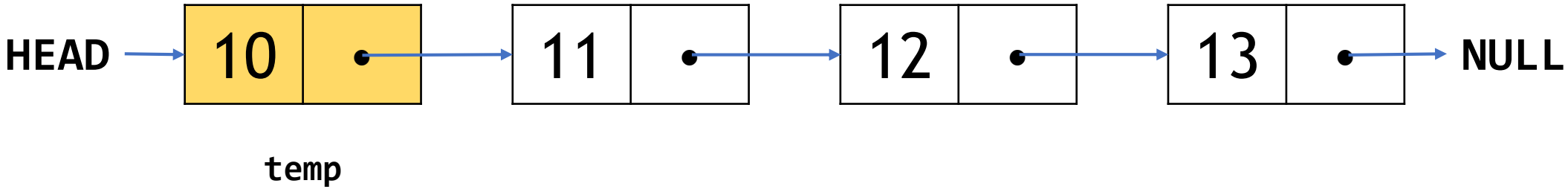
Deletion: Entire List



1. Initialize temp

```
Node* temp = HEAD;
```

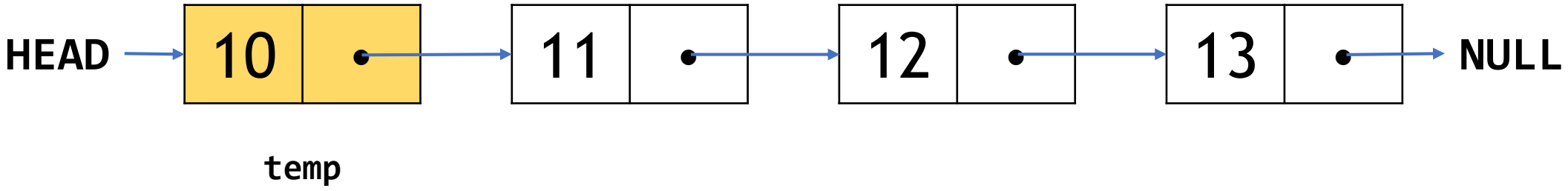
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

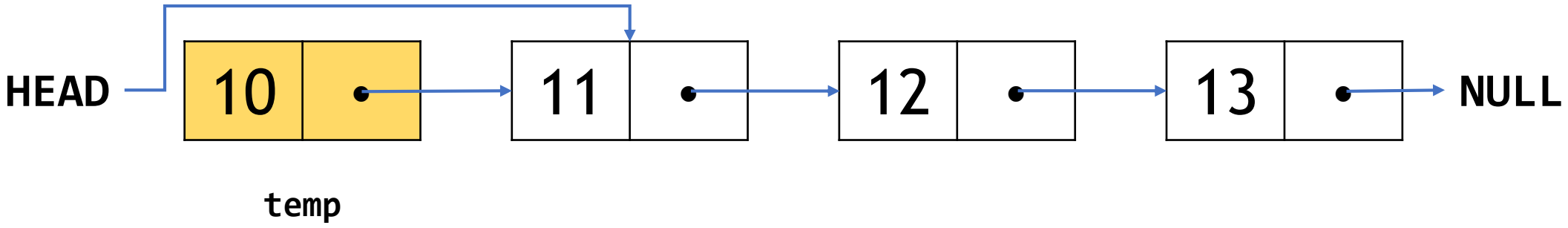
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

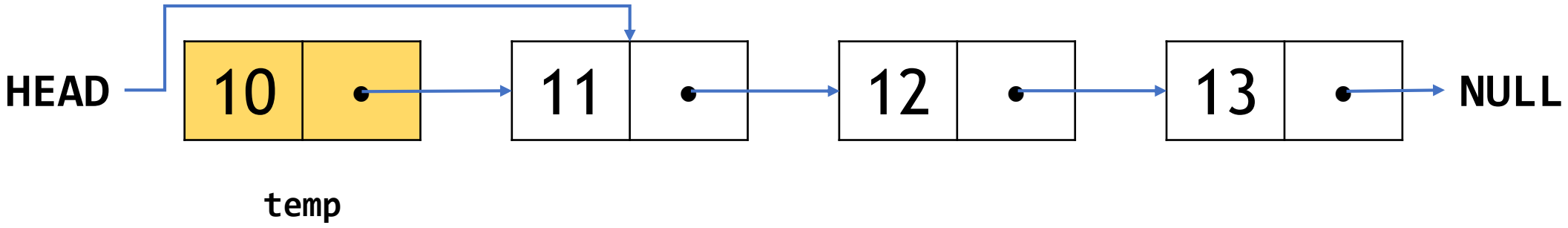
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

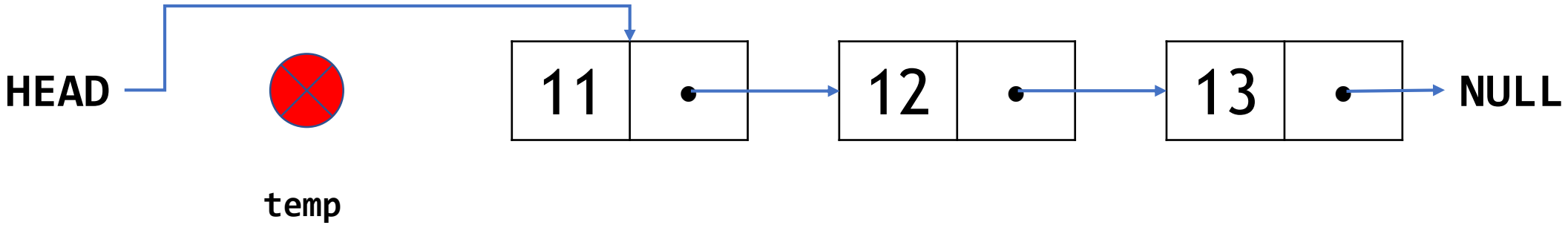
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

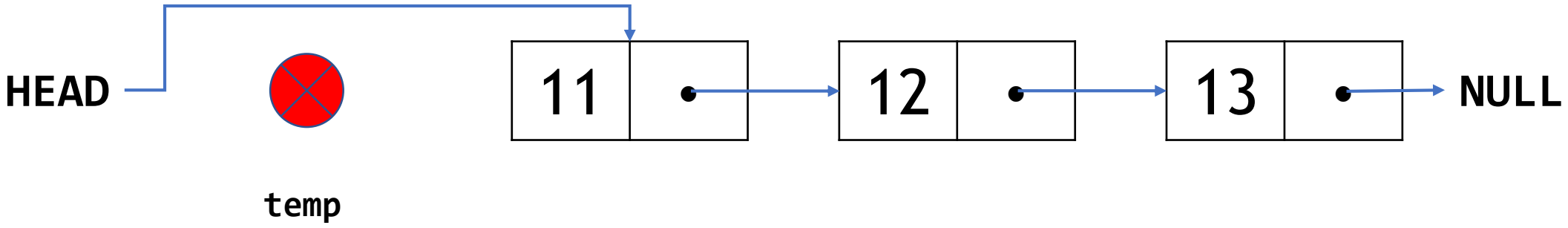
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

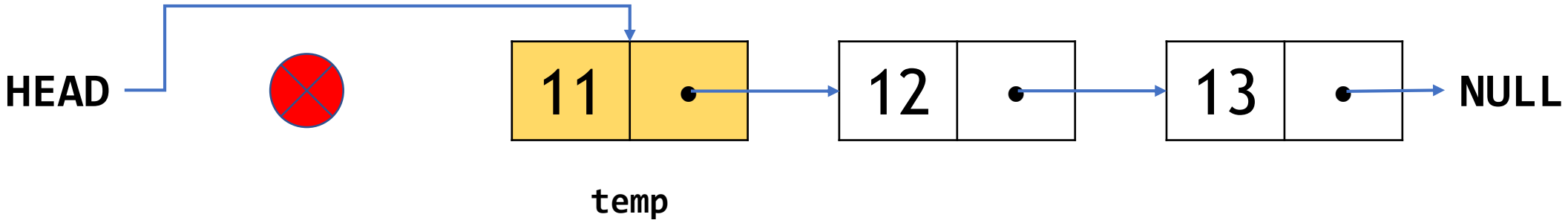
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

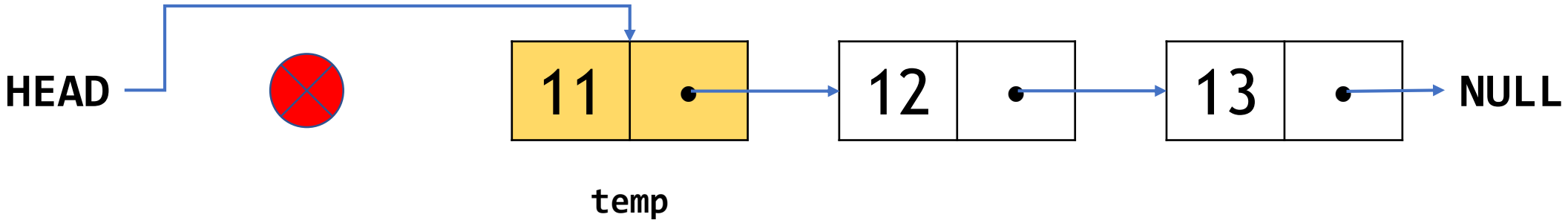
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

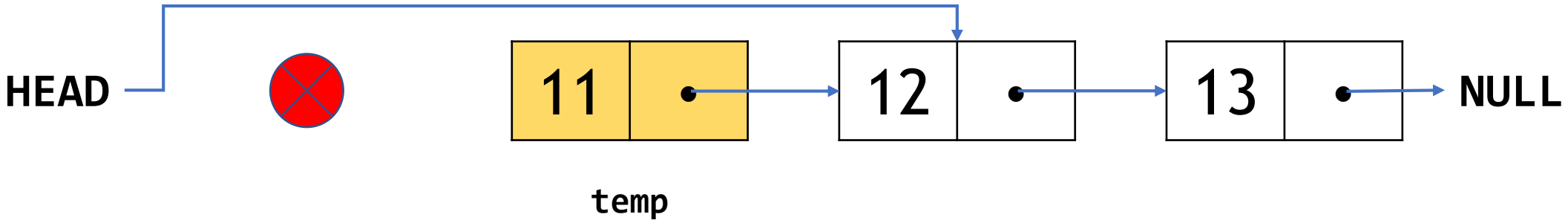
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

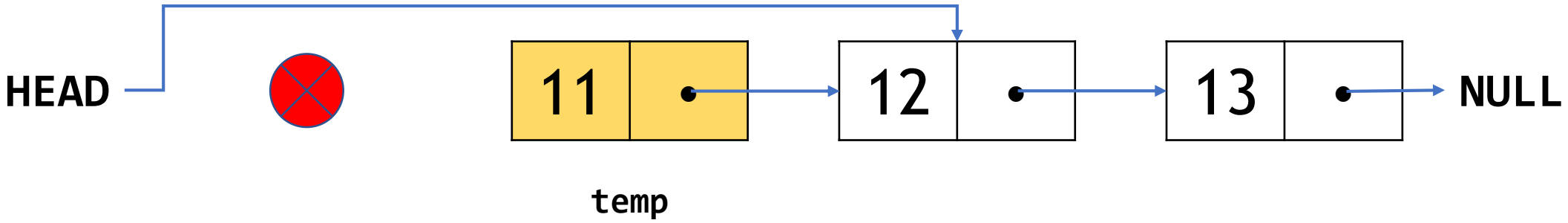
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

Deletion: Entire List



2. while (head != NULL) do

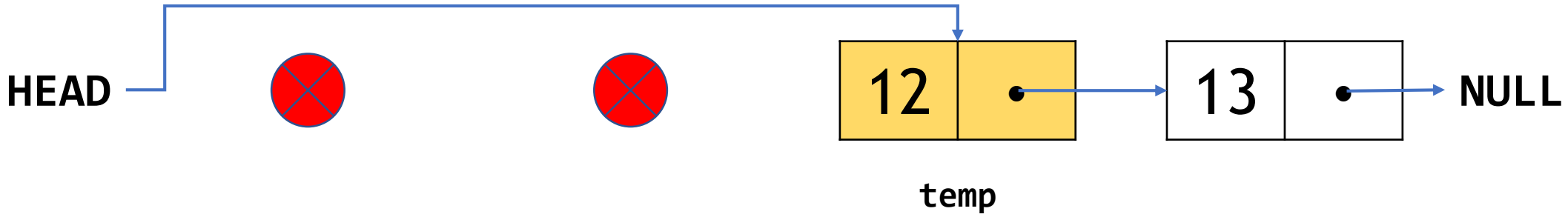
```
temp = head;
head = head->next;
delete temp;
```

Deletion: Entire List

2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

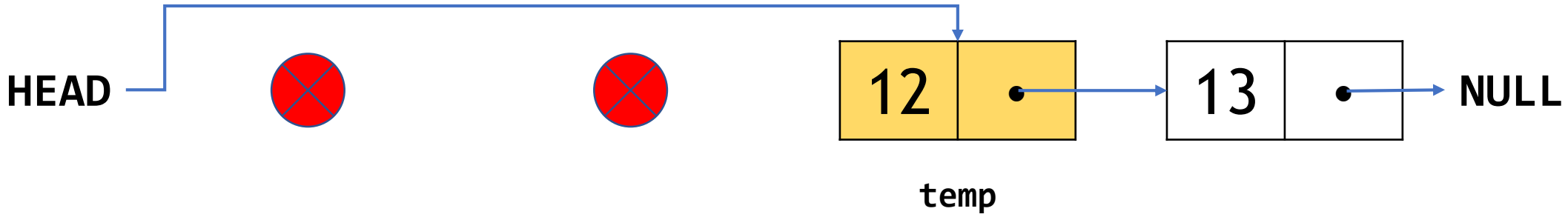
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

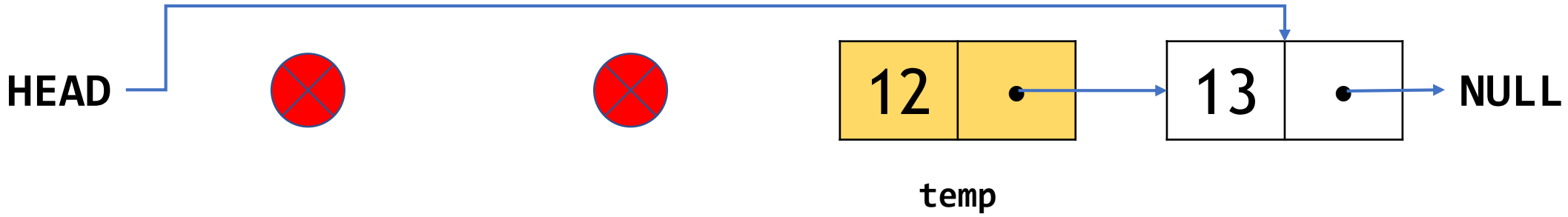
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

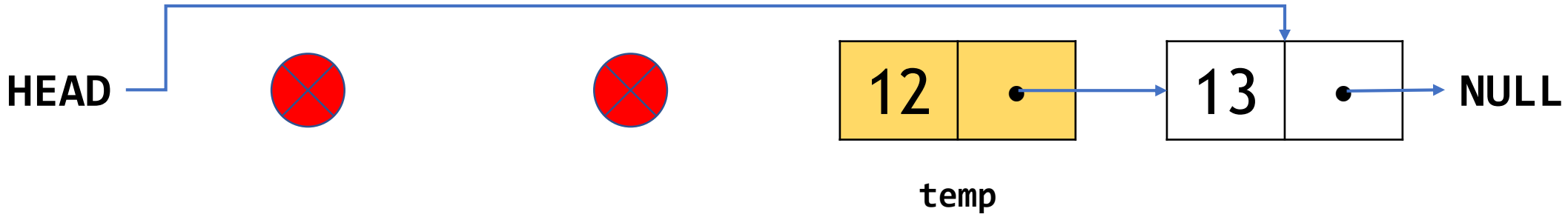
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

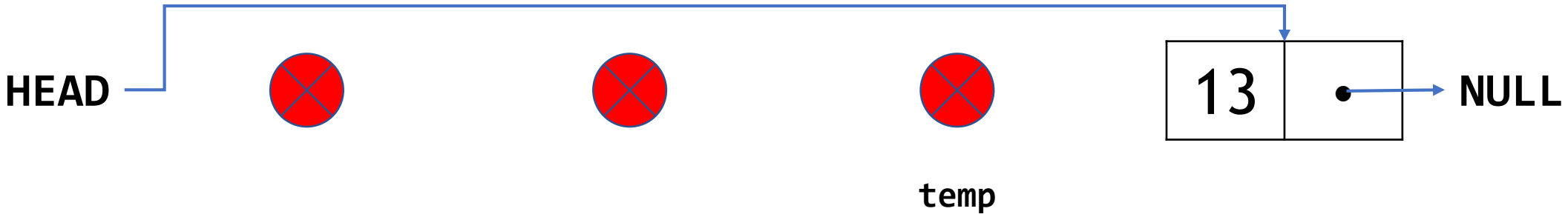
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

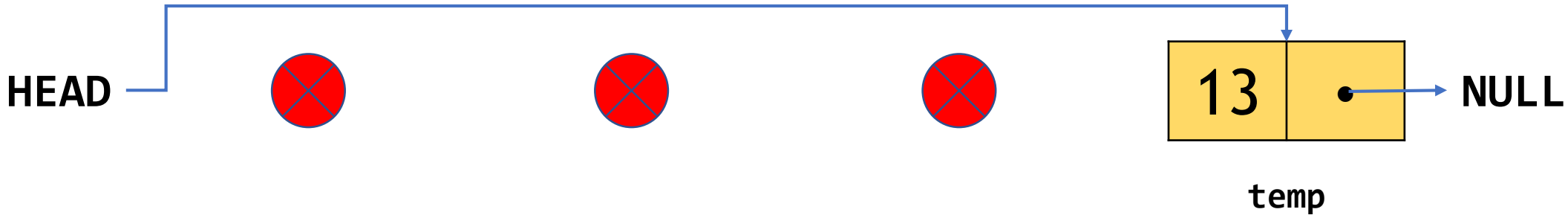
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

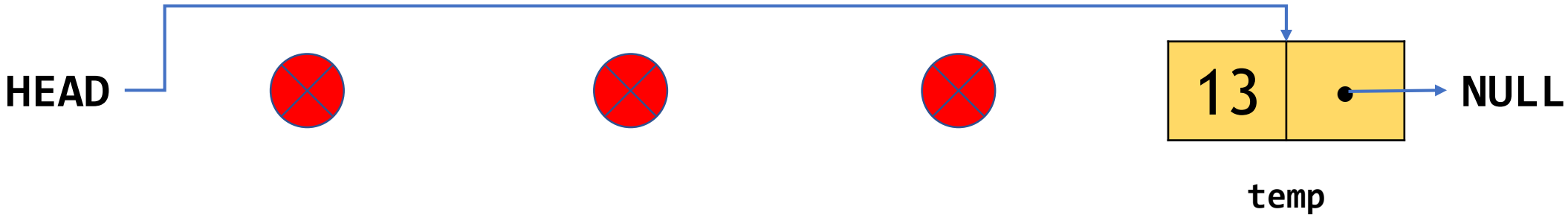
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

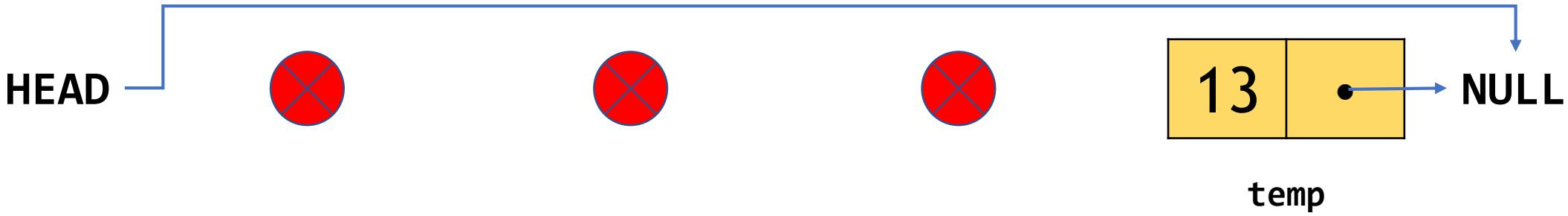
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

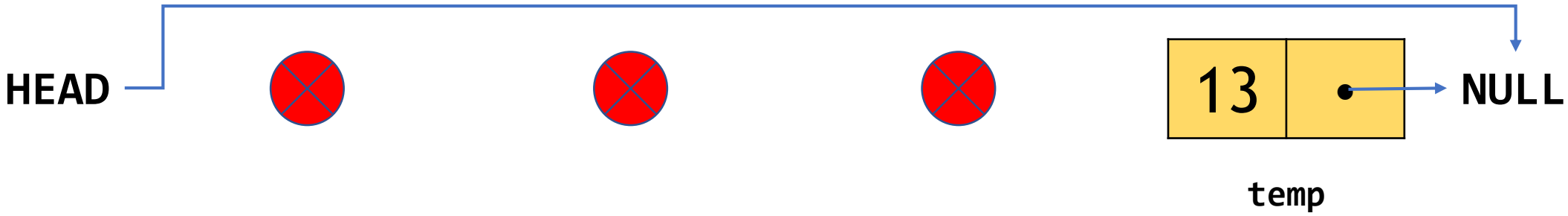
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

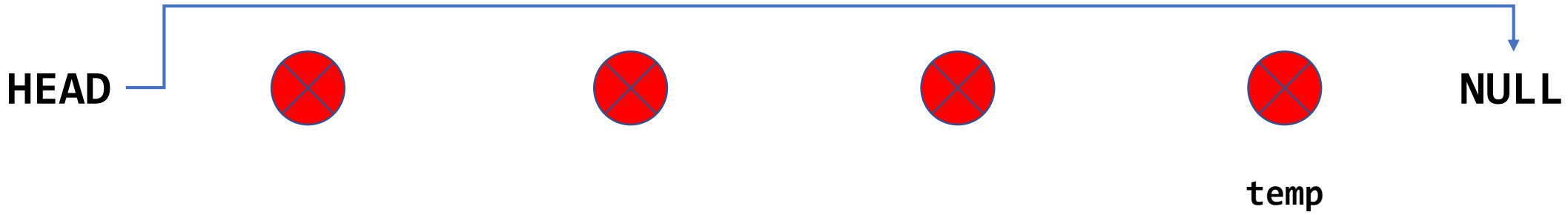
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

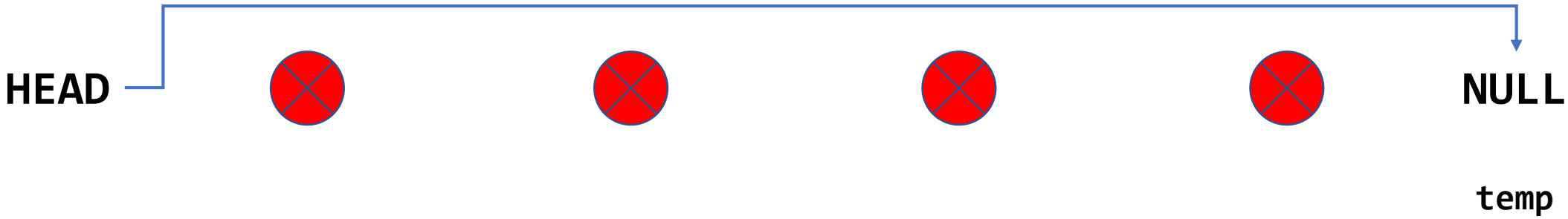
Deletion: Entire List



2. while (head != NULL) do

```
temp = head;
head = head->next;
delete temp;
```

Deletion: Entire List



3. Set temp to NULL

```
temp = NULL;
```

Deletion: Entire List

Deletion: Roundup

Take care of all the cases.

Order of rewiring, deletion is important.

Code File(s): `deletion_node.cpp, deletion_list.cpp`

Exercise

Linked List

Exercise: Silver

Q: ***Given a position in the linked list, delete the node at that position.***

Complete:

```
bool deleteAtIndex(int index);
```

How to?

Review: Deletion in a Linked List

Exercise: Gold

Q: ***Swap the first and last nodes in a linked list***

Complete:

```
bool swapFirstAndLast();
```

How to?

Draw it out!

Head, postHead, Tail, preTail node pointers, **rewire the links!**

Edge cases!