Lists

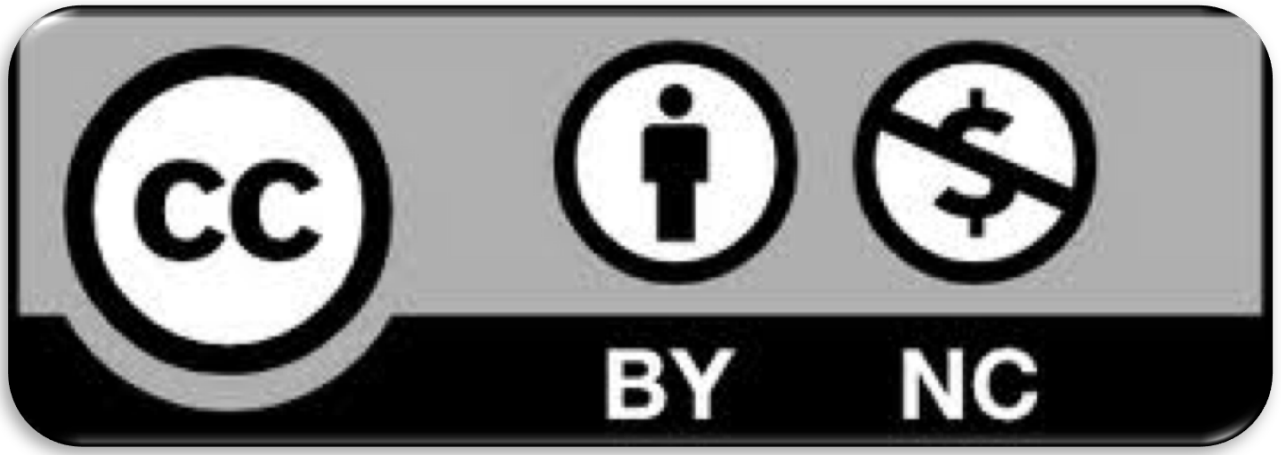
Alessandro SAVINO
Politecnico di Torino (Italy)

alessandro.savino@polito.it

www.testgroup.polito.it

License Information

This work is licensed under the Creative Commons BY-NC License



To view a copy of the license, visit:
http://creativecommons.org/licenses/by-nc/3.0/legalcode

Disclaimer

- **We disclaim any warranties or representations as to the accuracy or completeness of this material.**
- **Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.**
- **Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.**

Goal

– **This lecture aims at presenting Lists as ADT, focusing first on some typical operations and then on possible implementation.**

Prerequisites

- **Lectures:**
  - **11_8.1 and 11_8.2**

Further readings

- **Students interested in a deeper look at the covered topics can refer, for instance, to the books listed at the end of the lecture.**

Outline

- **Lists**
- **List operations**
- **List implementation**
- **Sentinels**
- **Doubly and multiple linked lists**
- **STL Lists**

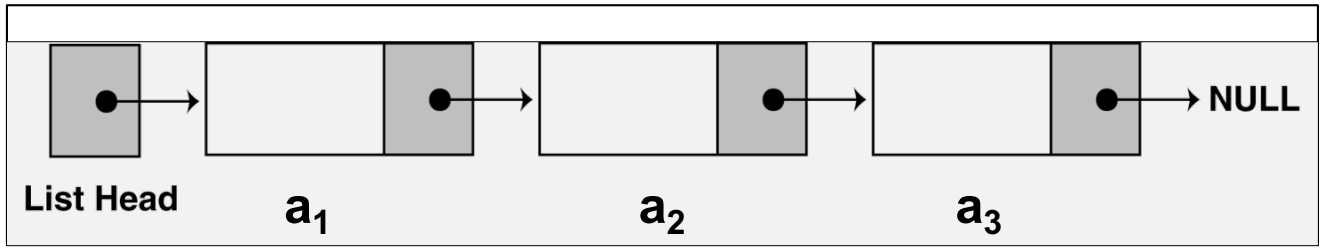
Outline

- Lists
- List operations
- List implementation
- Sentinels
- Doubly and multiple linked lists
- STL Lists

Liste

- **Lists are flexible ADTs**
  - **A list is a series of connected nodes, where each node is a data structure containing specific information**

    . $a_1, a_2, \ldots, a_n \quad n \geq 0$
  - **It is possible to insert, search, and delete every node inside the list**
  - **A list can grow or shrink in size as the program runs.**



Definitions

- If n = 0 *empty list*
- a_1 first element of the list, or *head*
- a_n the last element of the list, or *tail*
- a_i *precedes* a_{i+1} $i : 1 \leq i < n$
- a_i *follows* a_{i-1} $i : 1 < i \leq n$
- Node a_i is placed in the *position i*

Definitions

- **If n = 0 *empty list***
- **a_1 first element of the list, or *head***
- **a_n the last element of the list, or *tail***
- **a_i *precedes* a_{i+1} i : 1 ≤ i < n**
- **a_i *follows* a_{i-1} i : 1 < i ≤ n**
- **Node a_i is placed in the *position i***

Element or node *position* depends on the actual context and structure implementation, i.e., it may represent an index, address, pointer, or iterator

Outline

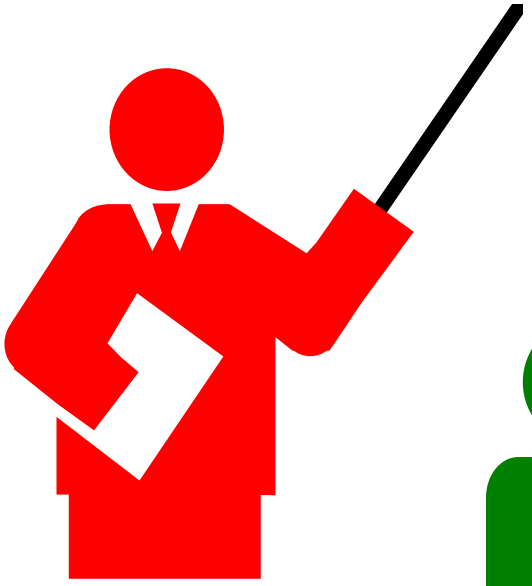
- Lists
- List operations
- List implementation
- Sentinels
- Doubly and multiple linked lists
- STL Lists

insert (x, p, L)

Inserts in the list *L* an element *x* in position *p*.

Returns:

- **OK if everything is ok**
- **ERROR otherwise.**

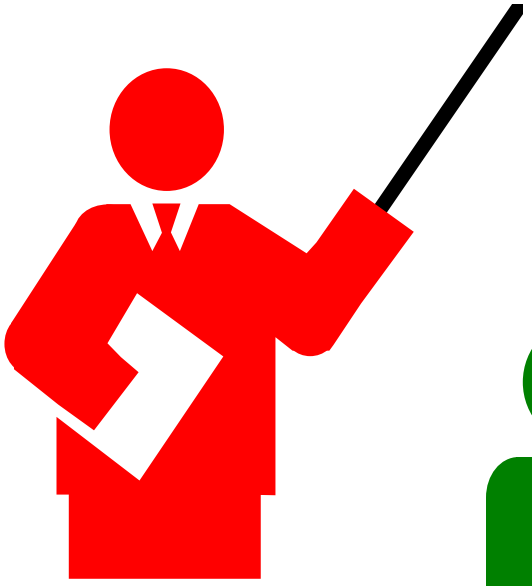


delete (p, L)

Deletes in the list *L* the element in position *p*.

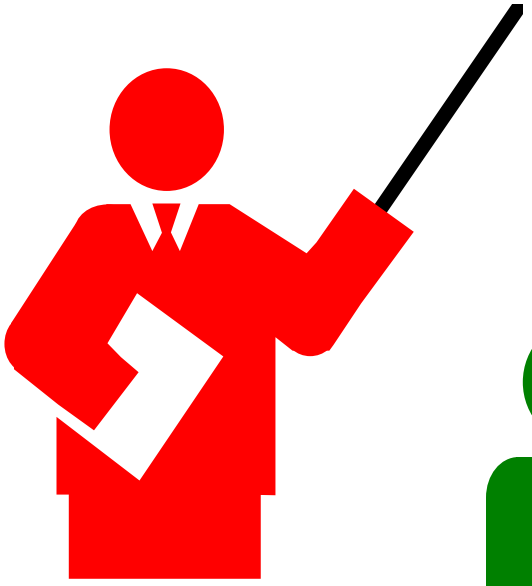
Returns:

- **OK if everything is ok**
- **ERROR otherwise.**



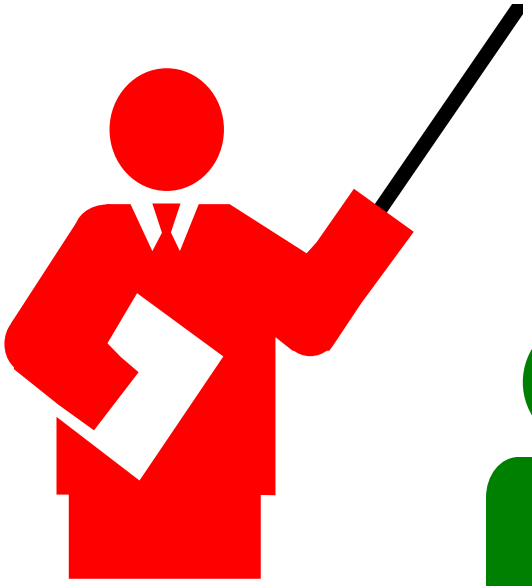
locate (x, p, L)

Returns the position in the list *L* of the first occurrence of *x* after position *p*.
If the element does not exist returns the last element in the list.



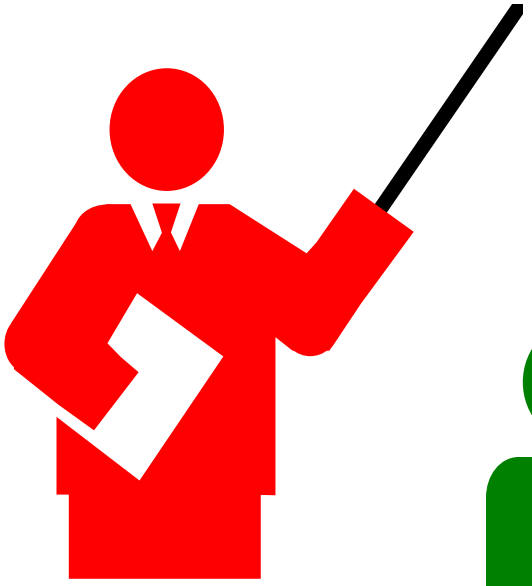
retrieve (p, L)

Returns the element of the list L at position p;
If the element does not exist returns ERROR.



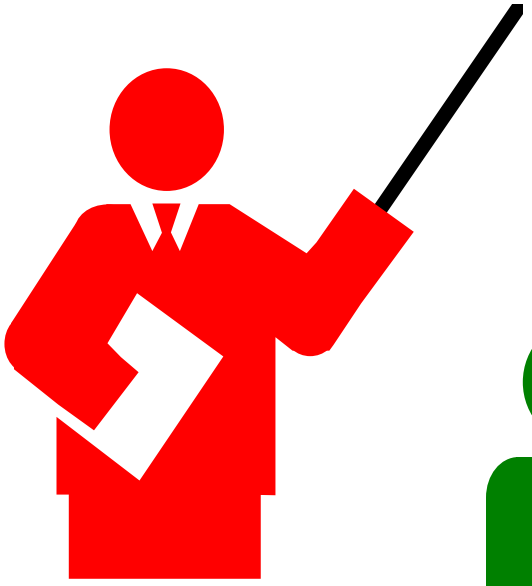
next (p, L)
previous (p, L)

Return the next or previous position of the element in position *p*.
If p ∉ [1,n], ERROR



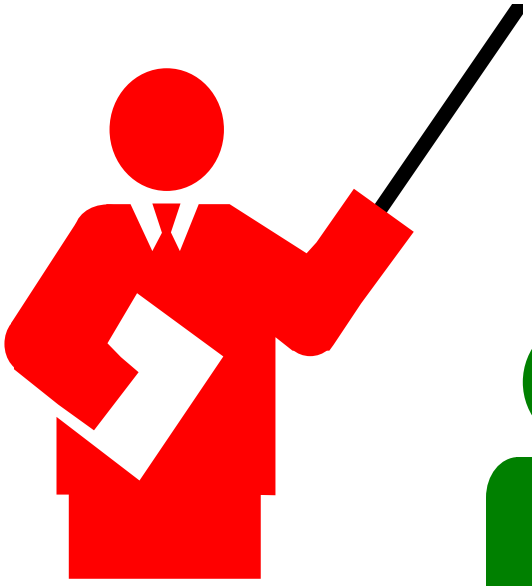
next (p, L)
previous (p, L)

next(n,L) = eol(L)
next(eol(L),L) = ERROR
previous(1,L) = ERROR



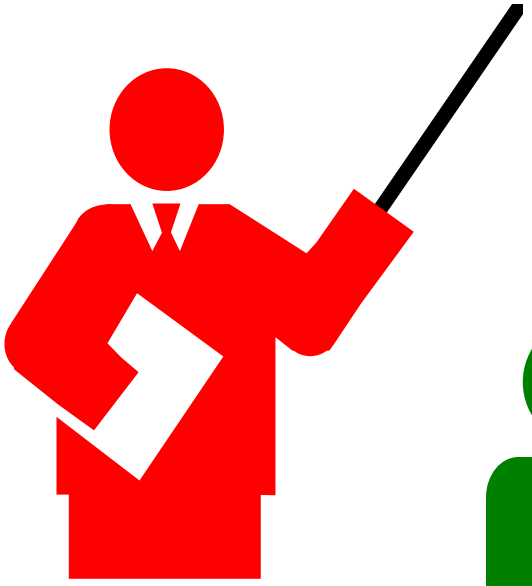
clear(L)

Clear the content of the list.



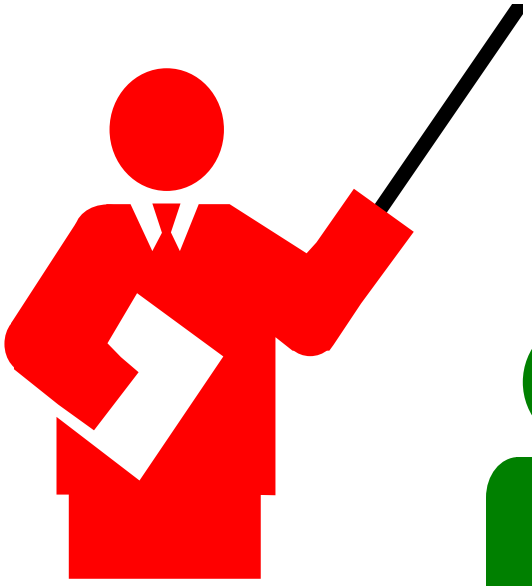
first (L)

Returns the first position in the list.



end(L)

Returns the last position in the list.



Outline

- Lists
- List operations
- List implementation
- Sentinels
- Doubly and multiple linked lists
- STL Lists

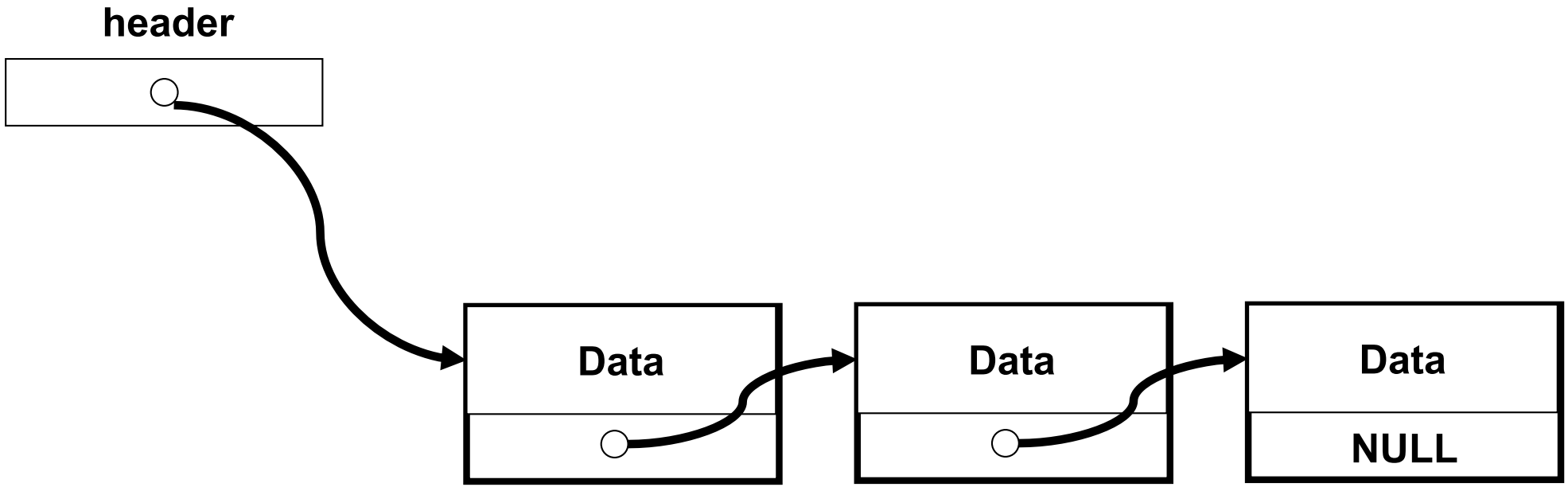
List implementation

There are different implementations of a list:

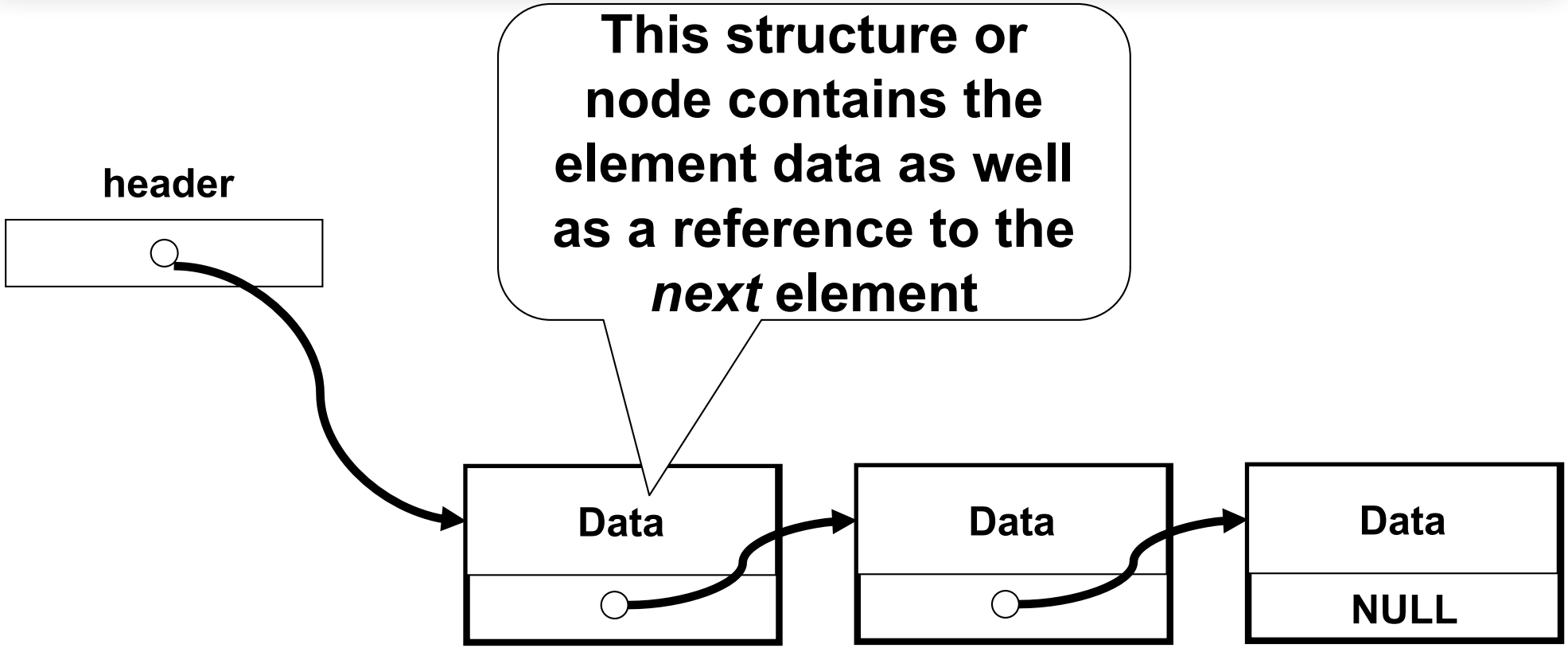
- **Implicit**
  - **Using vectors**
- **Explicit:**
  - **Devoted structures**

- **Implementation may depend on the list operations, list size, programming language.**

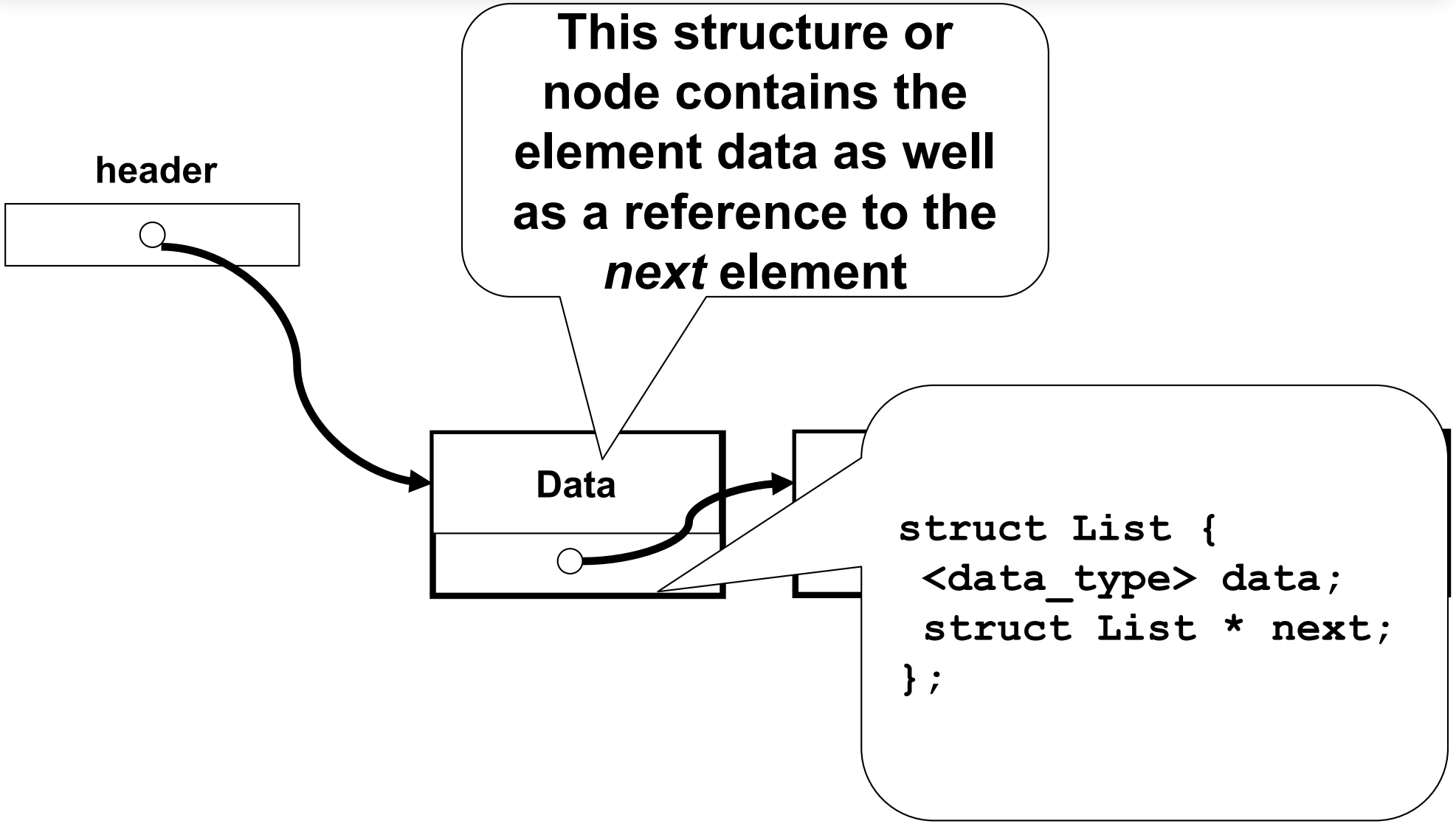
List Implementation



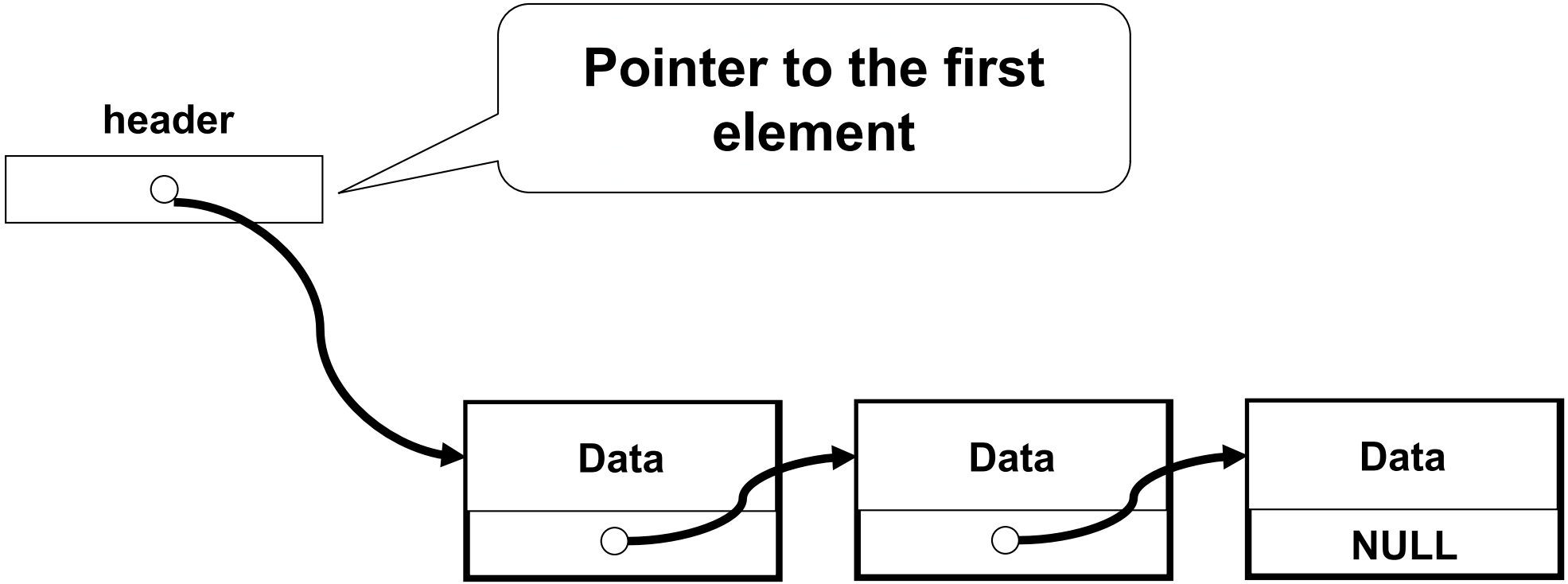
List Implementation



List Implementation



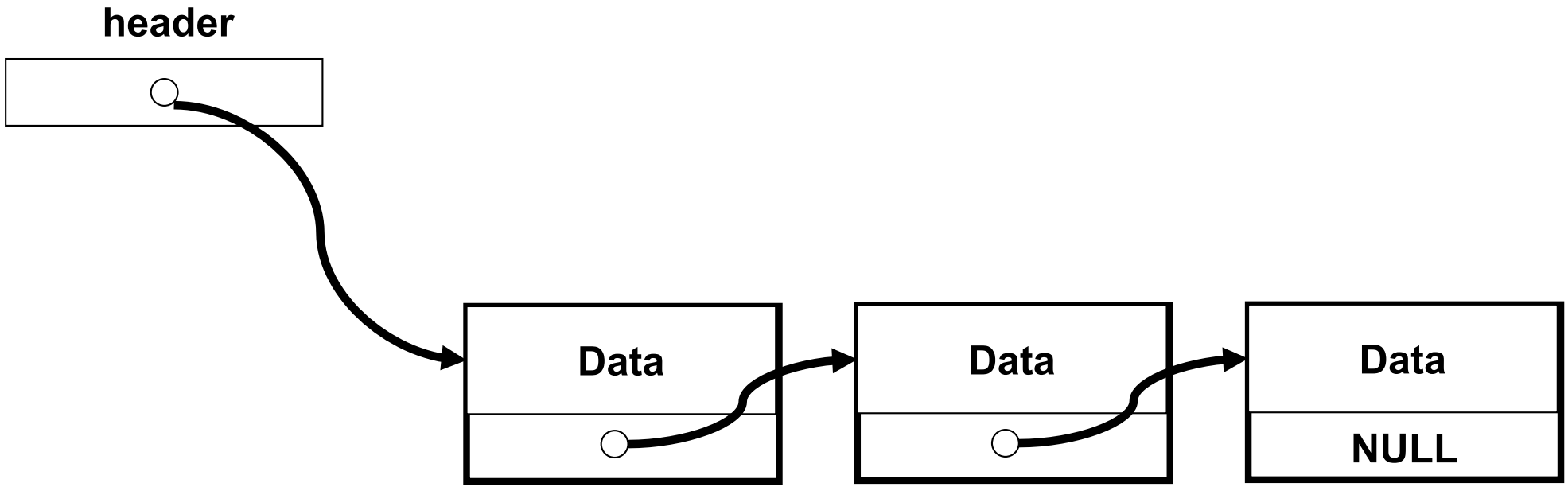
List Implementation



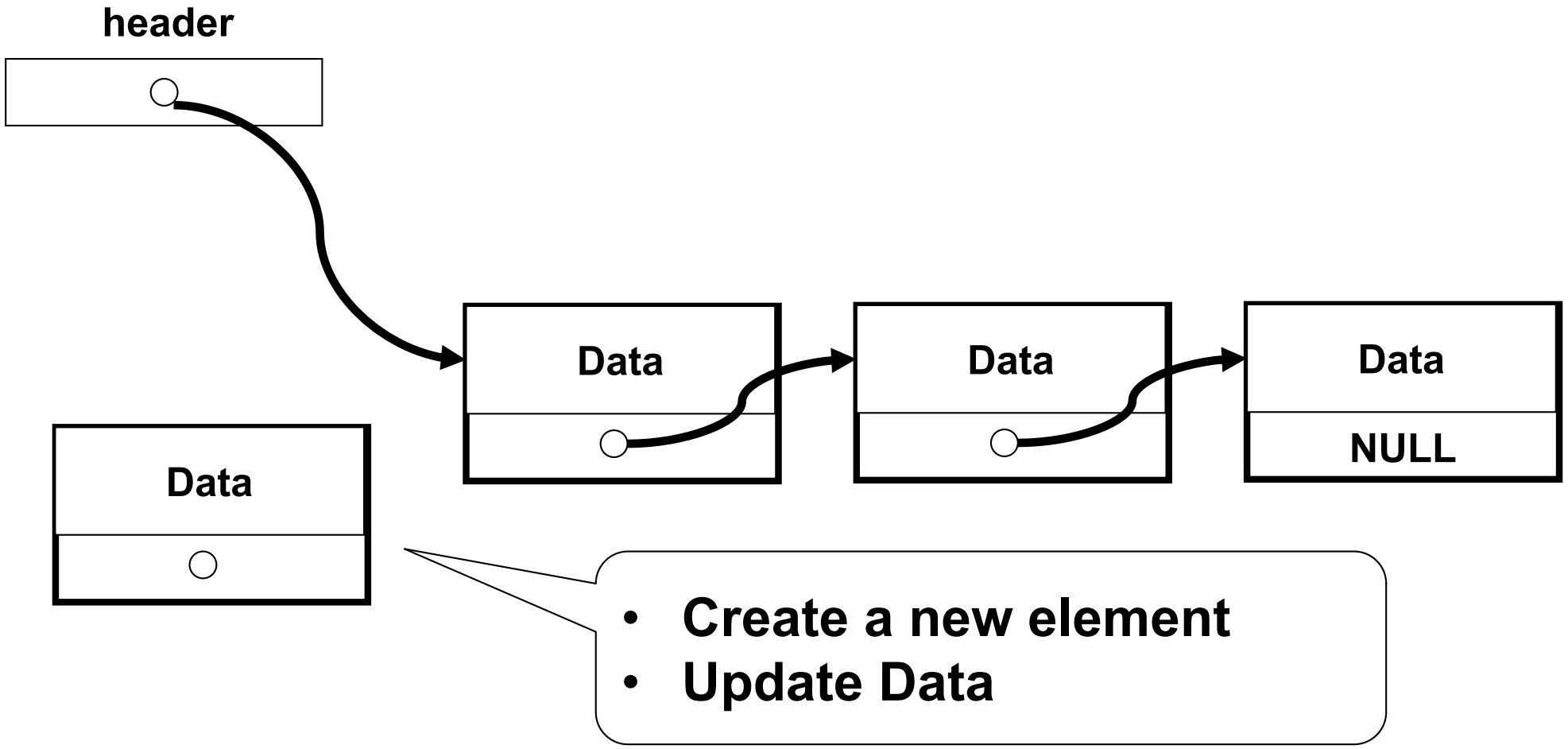
Insert operations

- ***Insert at the head***
- ***Insert inside***
- ***Insert at the end***

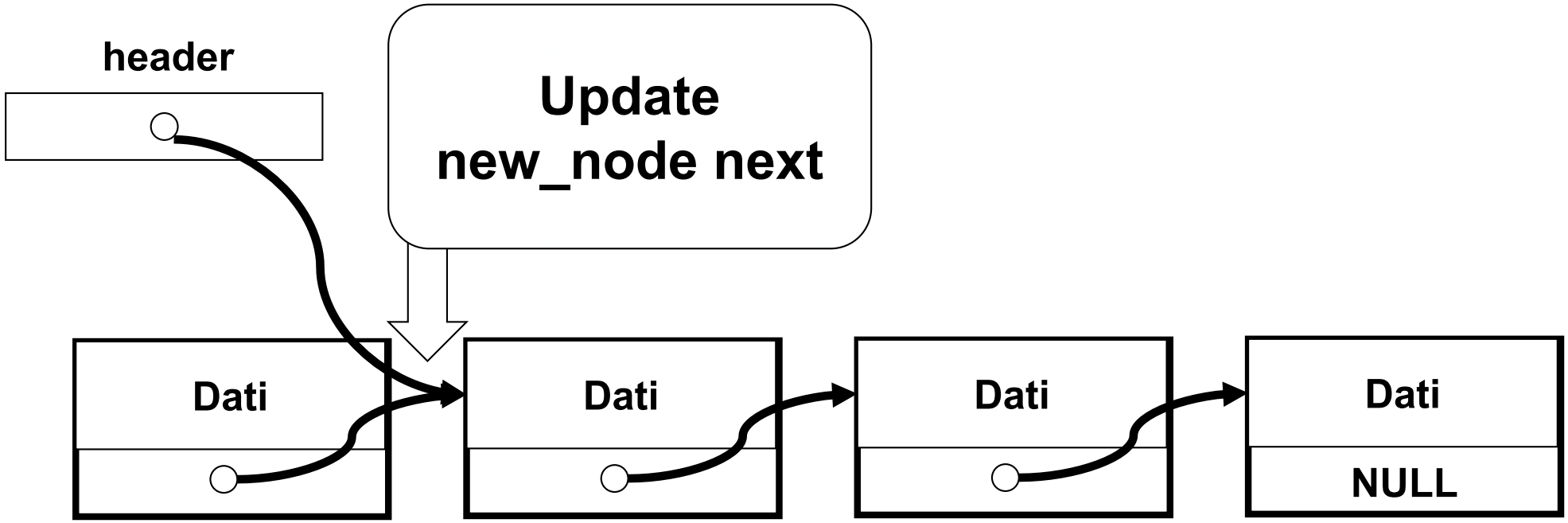
Insert at the head



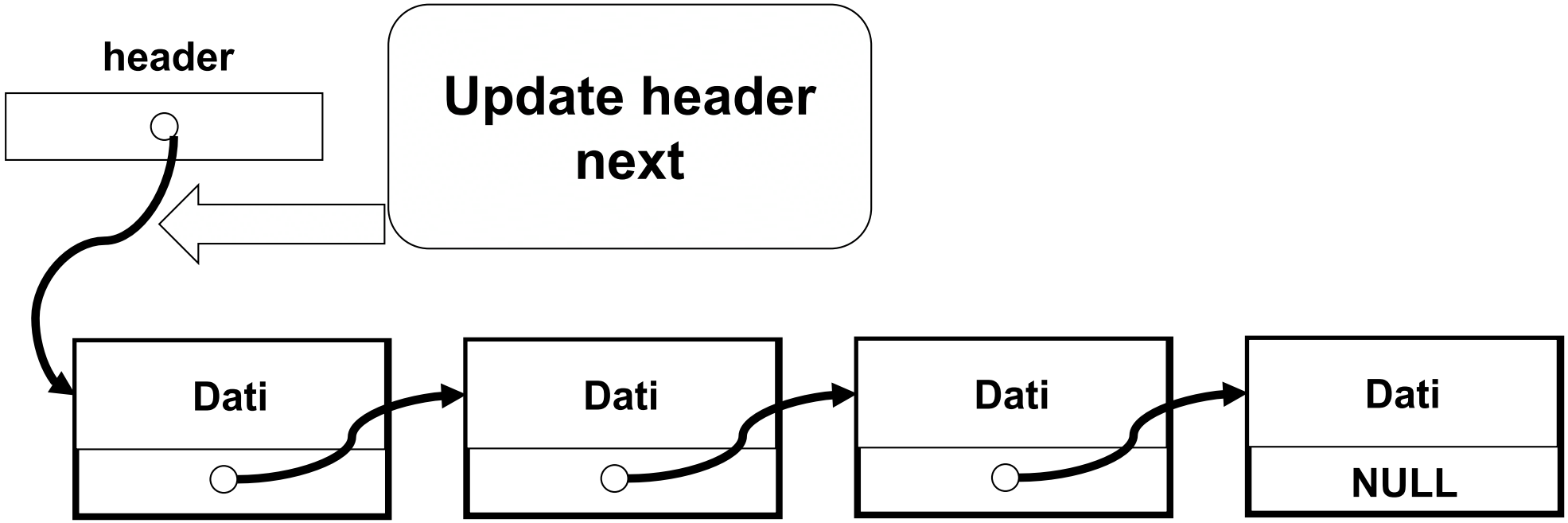
Insert at the head



Insert at the head



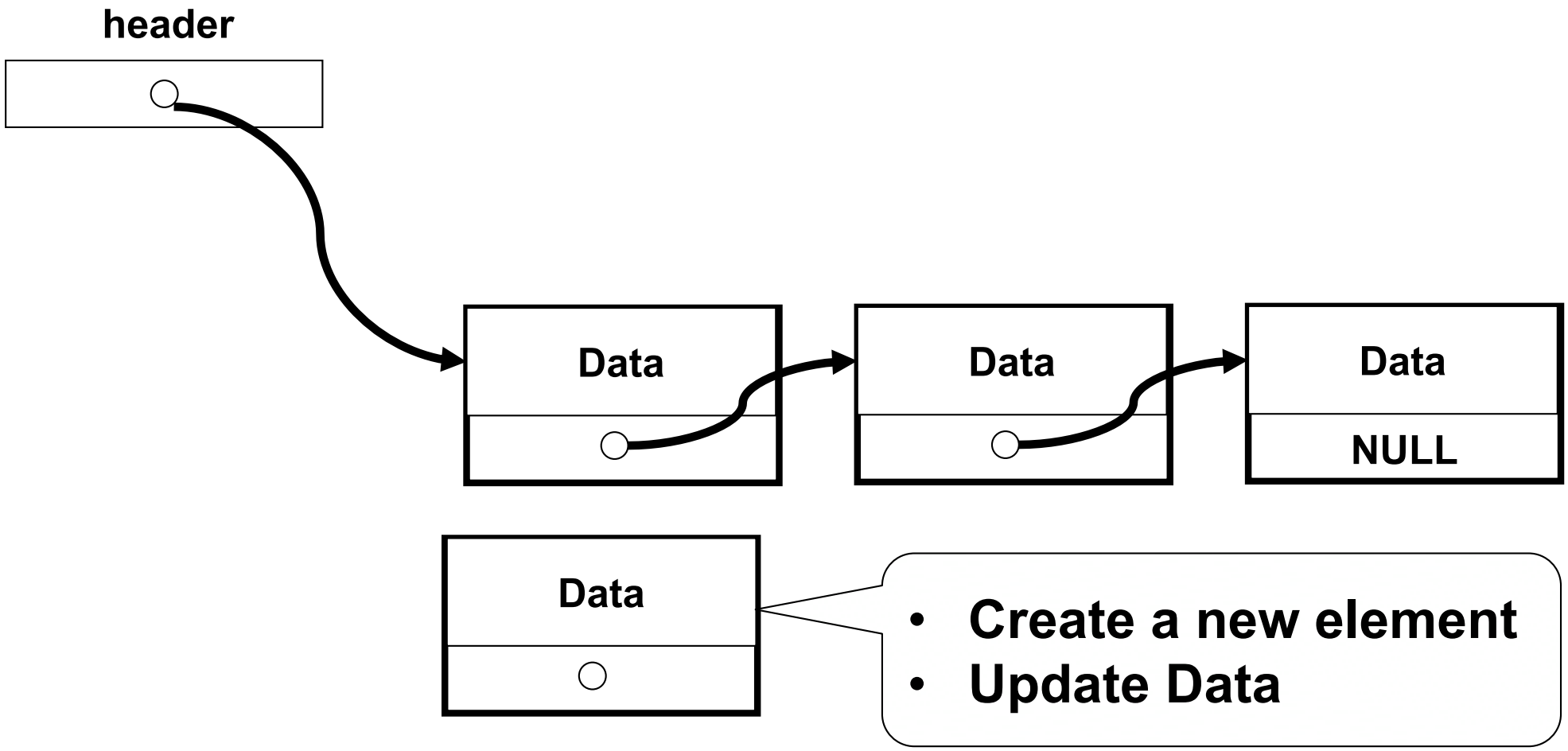
Insert at the head



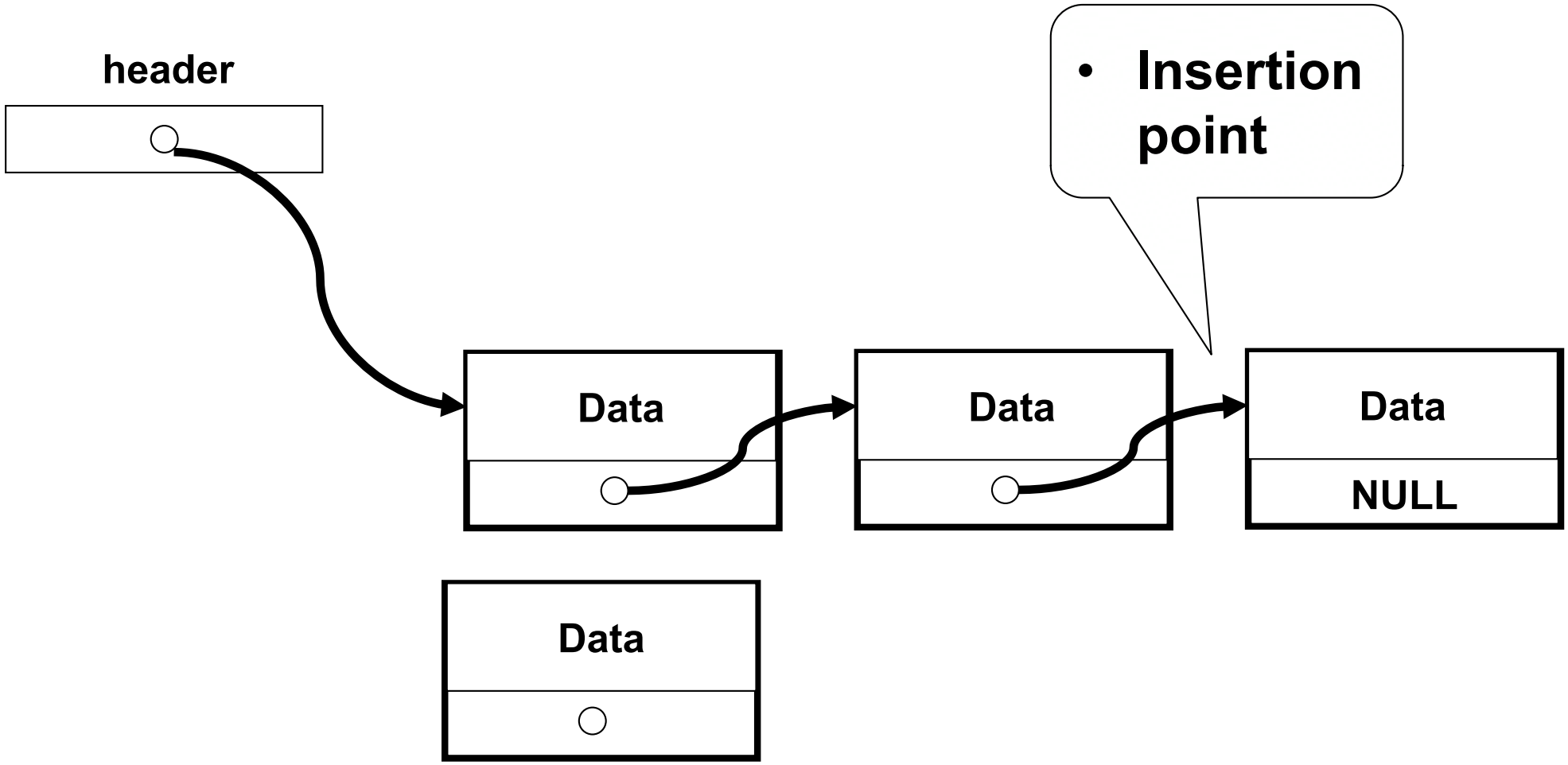
Insert operations

- ***Insert inside***

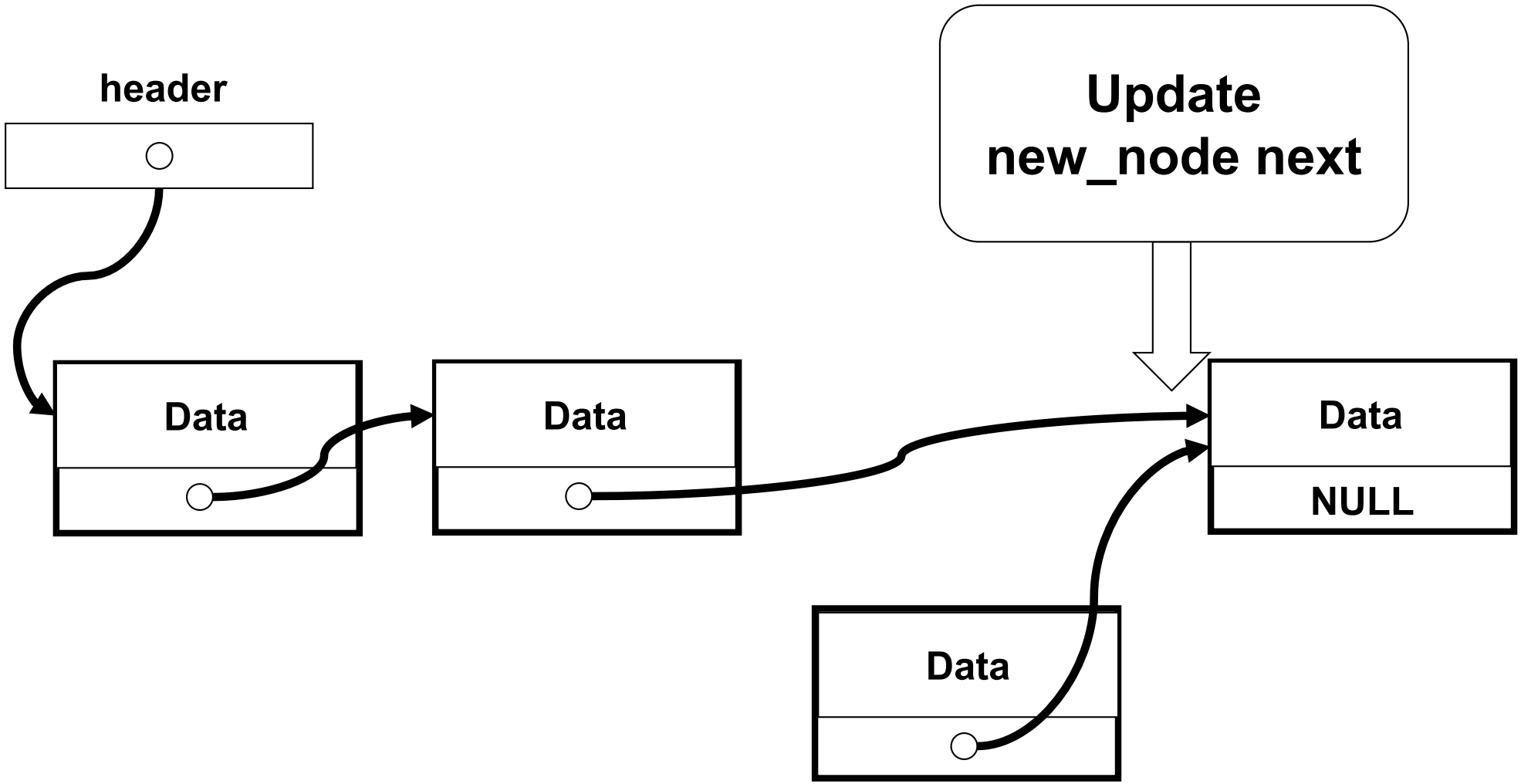
Insert inside



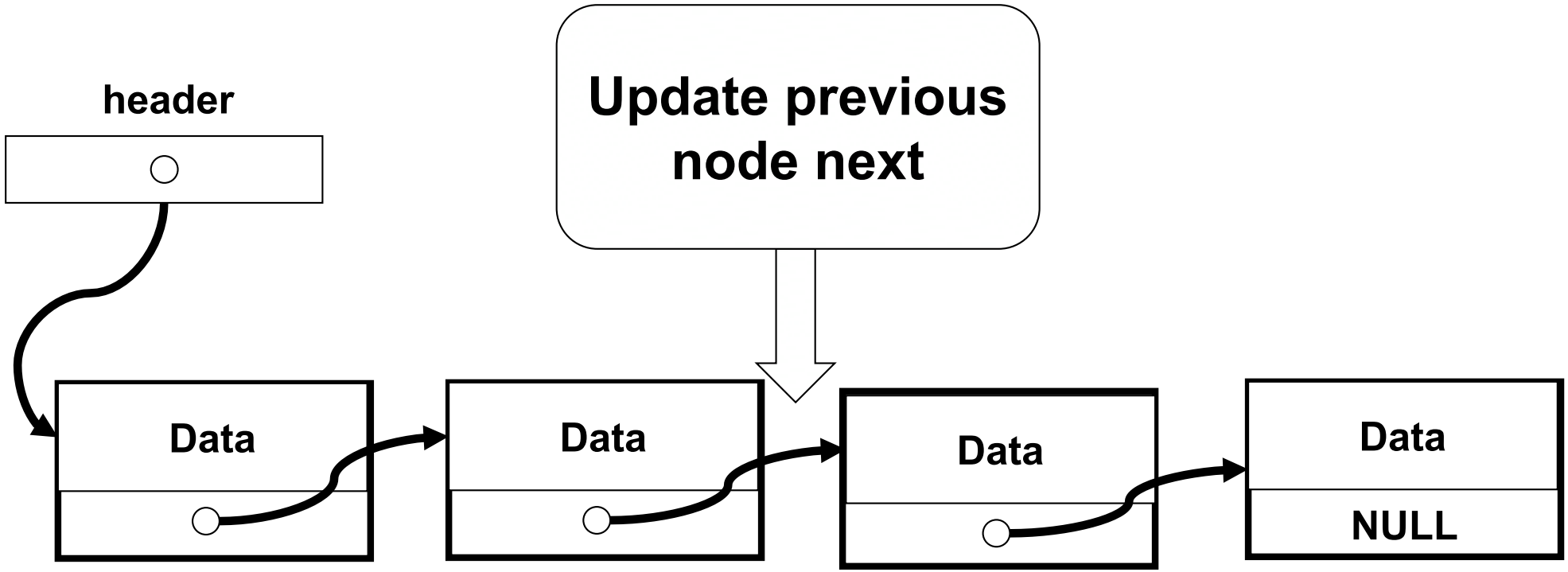
Insert inside



Insert inside



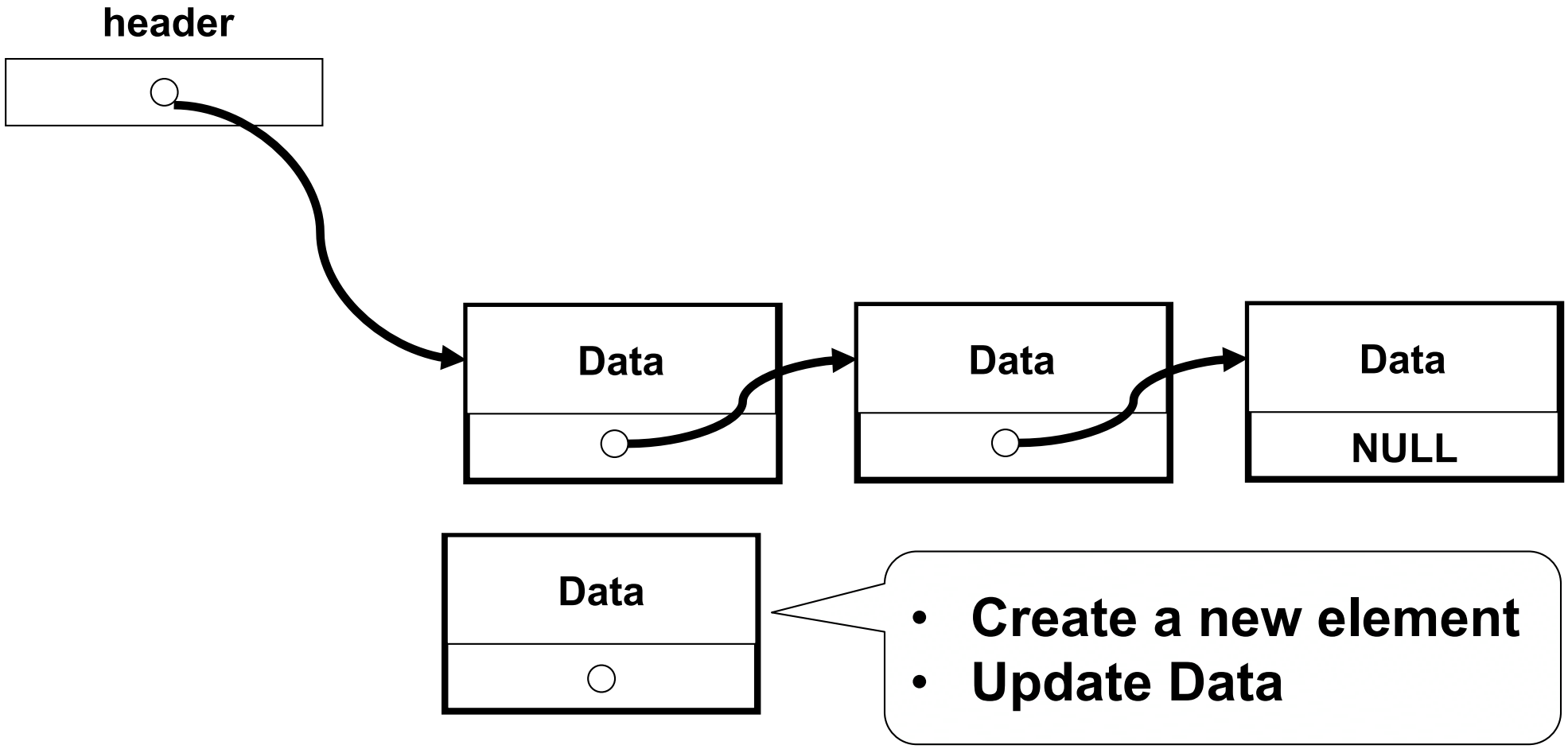
Insert inside



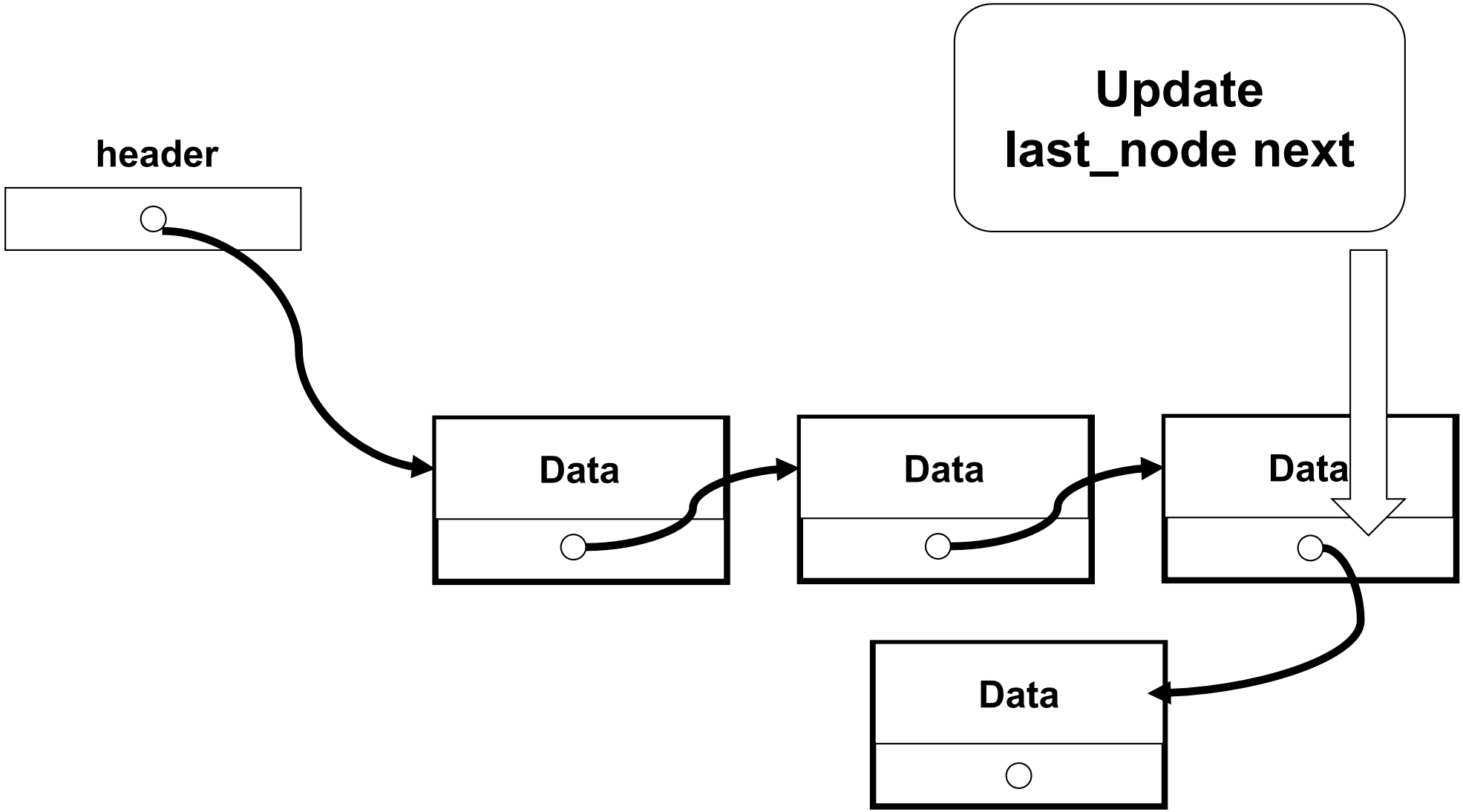
Insert operations

- ***Insert at the end***

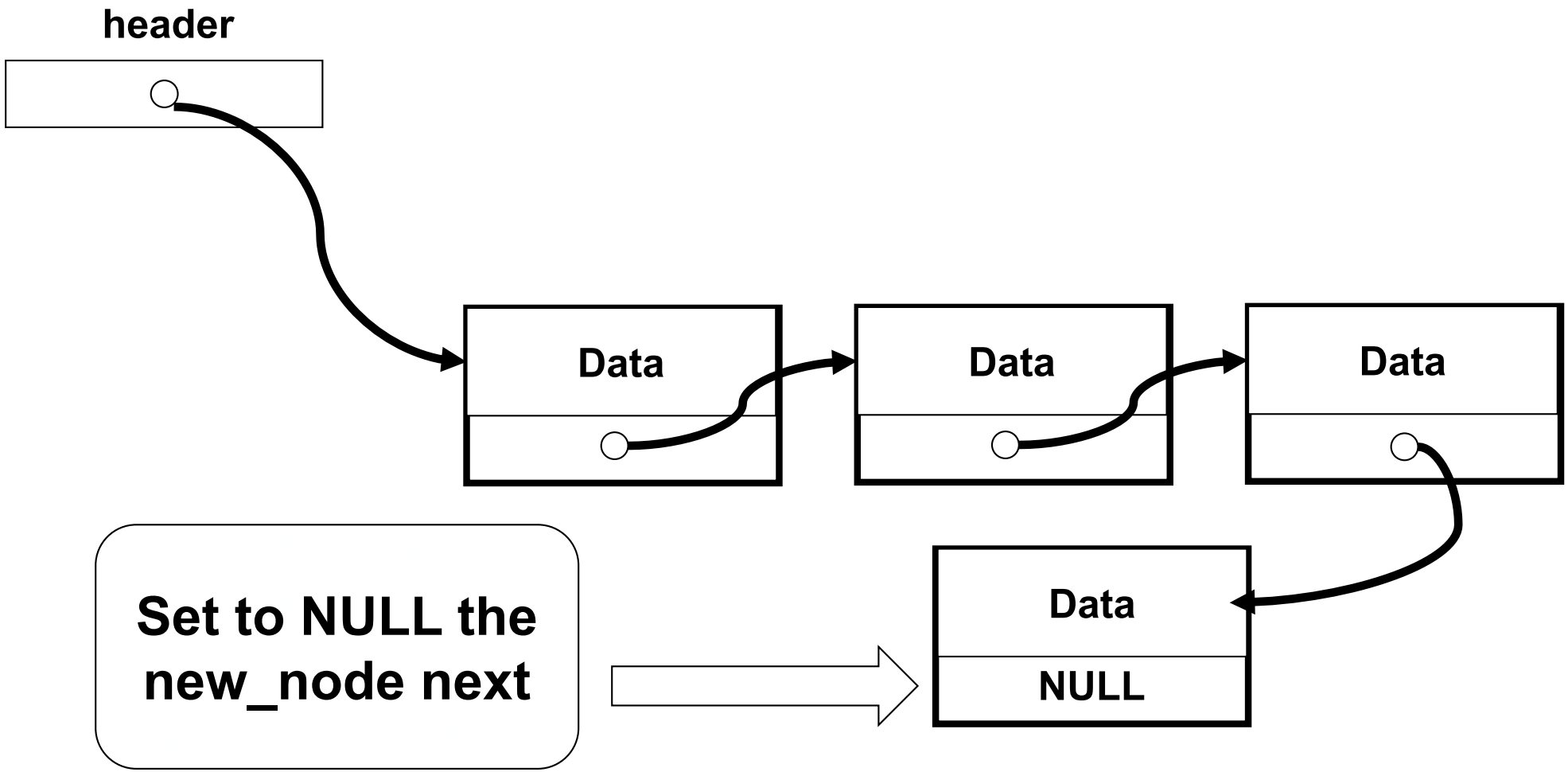
Insert at the end



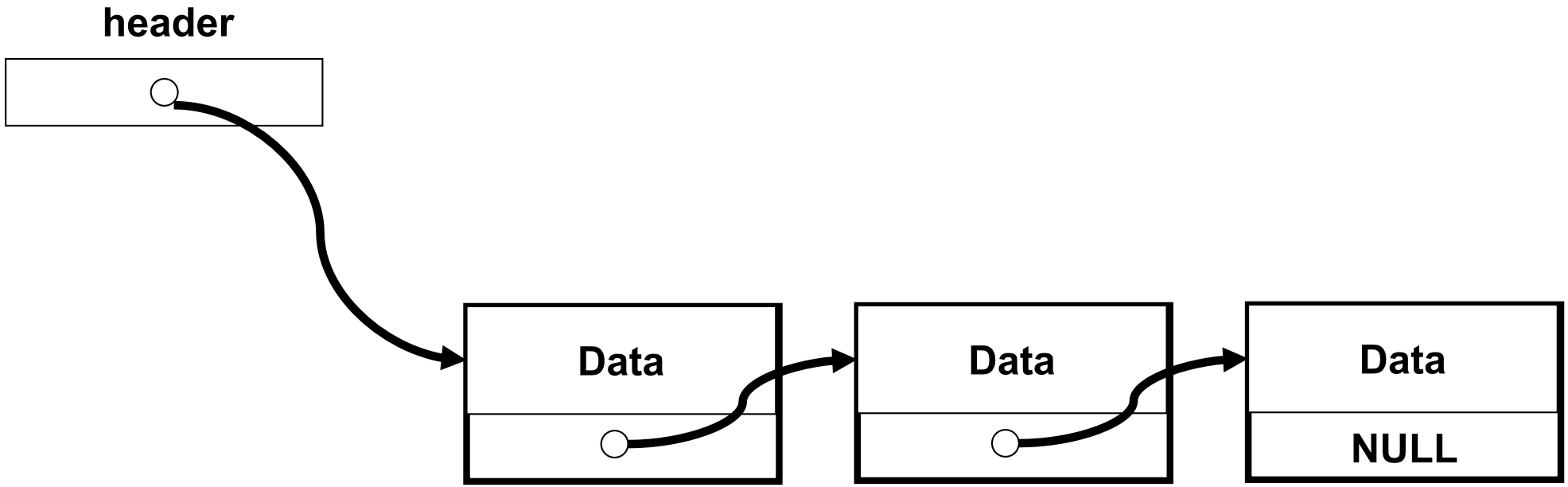
Insert at the end



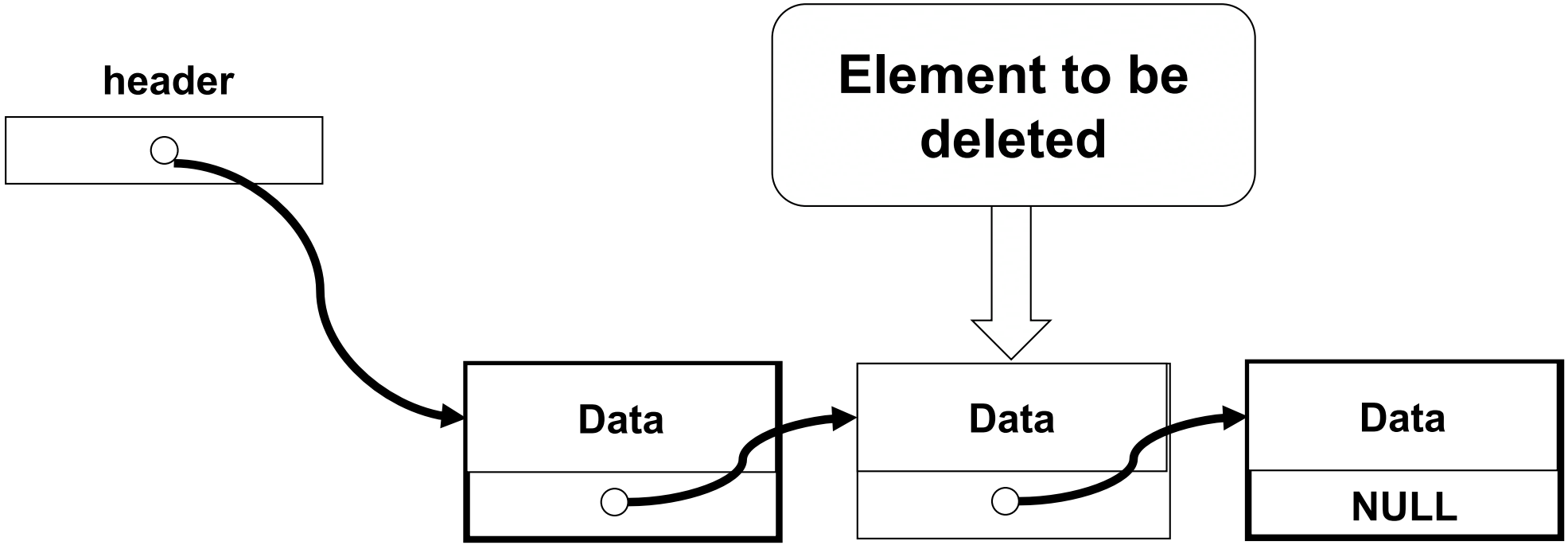
Insert at the end



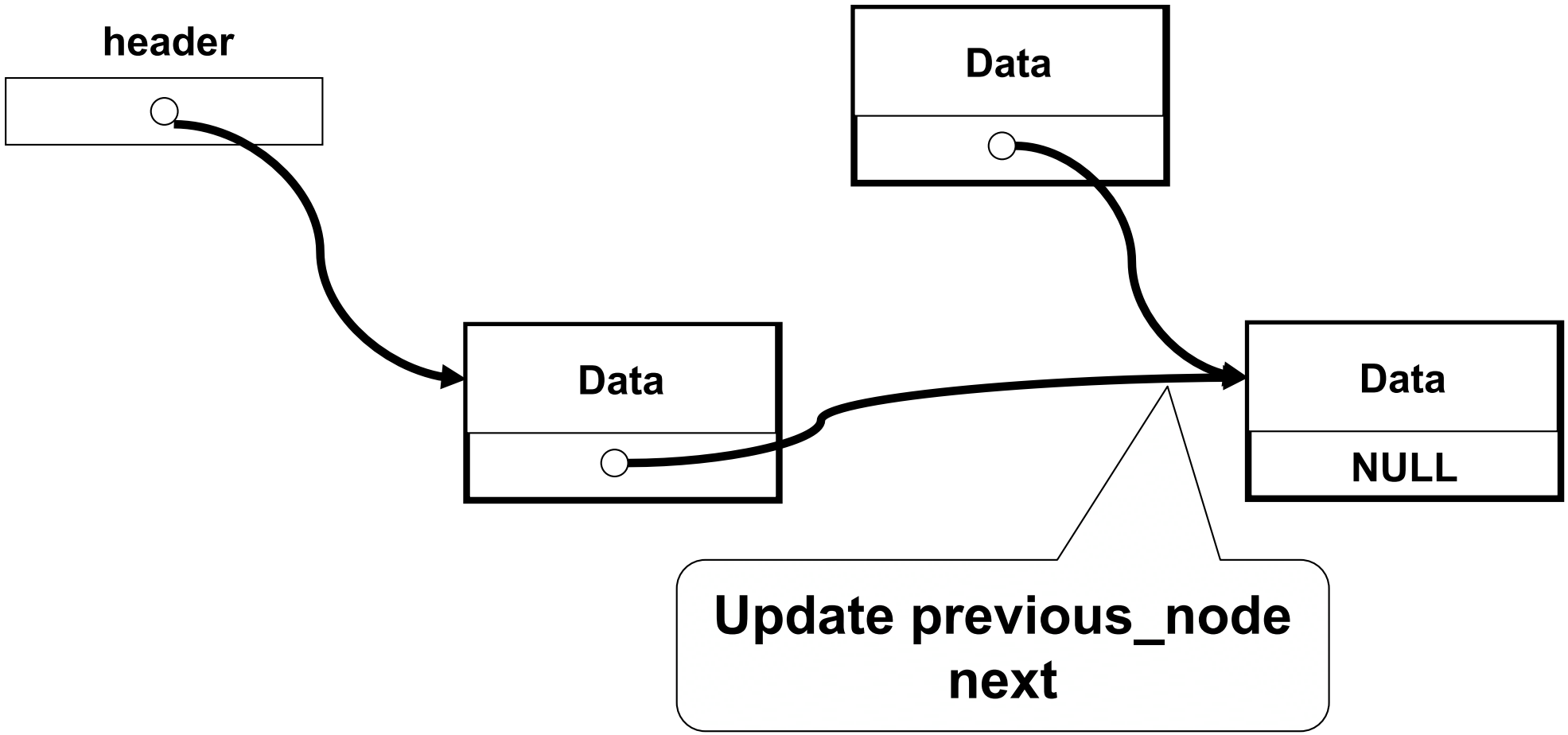
Delete operation



Delete operation



Delete operation



Outline

- Lists
- List operations
- List implementation
- Sentinels
- Doubly and multiple linked lists
- STL Lists

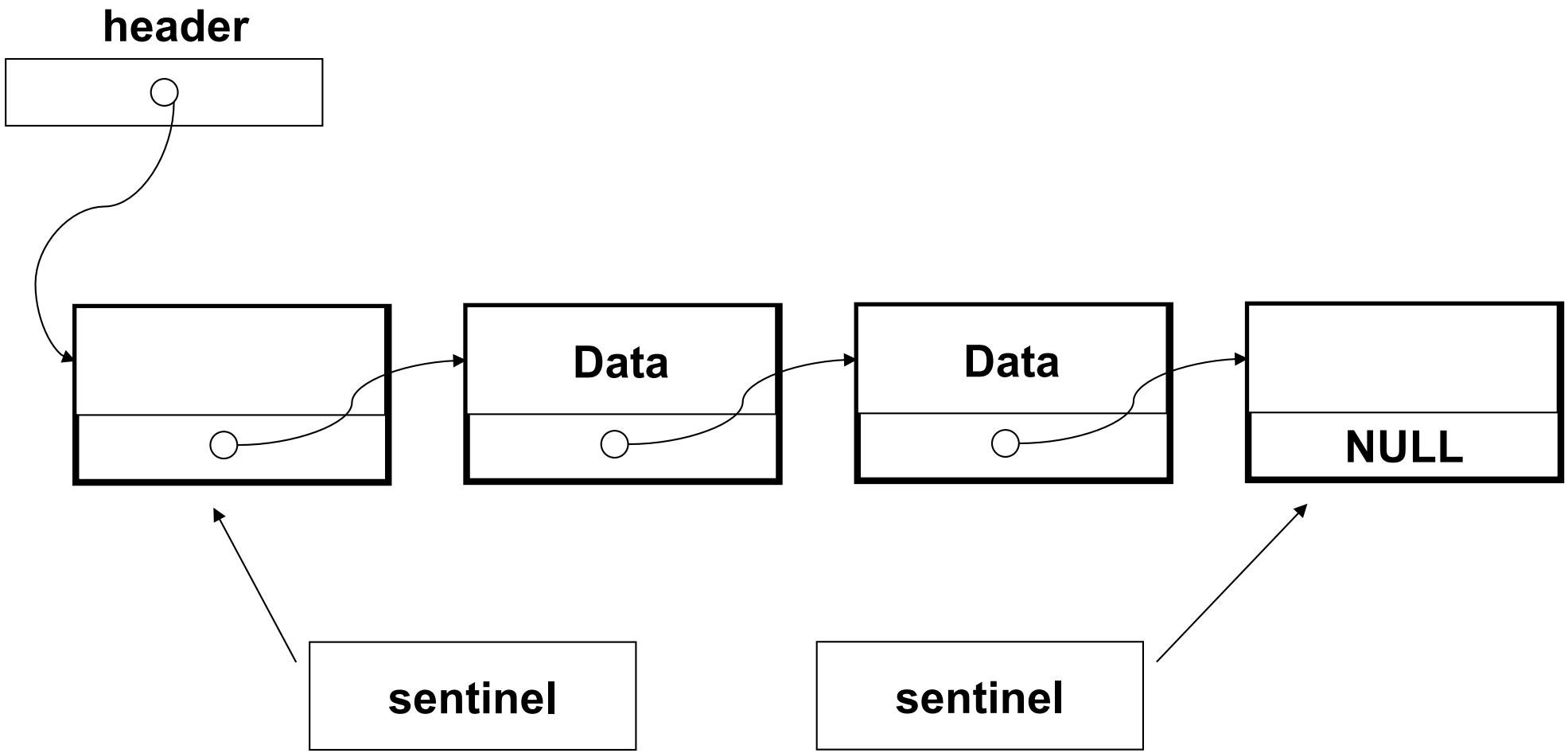
Sentinels

Dummy elements added to the head and tail of a list. Sentinels simplify boundary conditions.

Sentinels

- ***Head sentinels*:**
  used to avoid header modification during a list insertion or deletion in the list header.
- ***Tail sentinels:***
  used to eliminate the second test during the search operations.

Sentinels



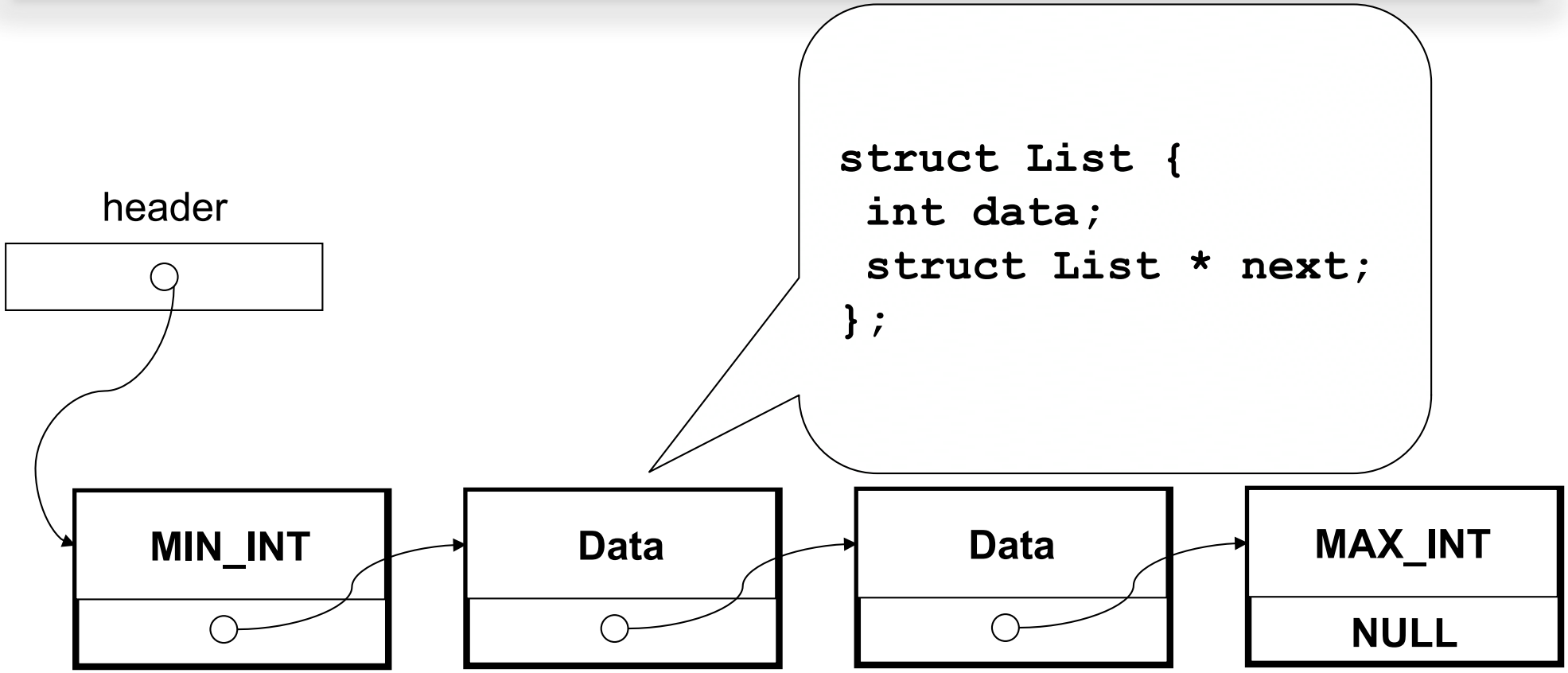
Sentinels: unsorted lists

- **Usually 2 sentinels are used, one in the head, the second one in the tail.**
- **Sentinels allow to reduce the code (i.e., it is not necessary to consider special cases for insert/delete in the head or tail).**

Sentinels: sorted lists

- **Usually 2 sentinels are used, one in the head, the second one in the tail containing the minimum and maximum values allowed in the list.**
- **Sentinels allow to simplify the code (i.e., it is not necessary to consider special cases for insert/delete in the head or tail).**
- **It is not necessary to test the end of the list.**

Example



Using the tail sentinel

- **Searching in sorted lists:**
  - **The tail sentinel must containg the maximum value allowed.**

- **Searching in unsorted lists:**
  - **Saving into the tail sentinel the searched value. If the value is present, the search finds the actual node, otherwise, the last one.**

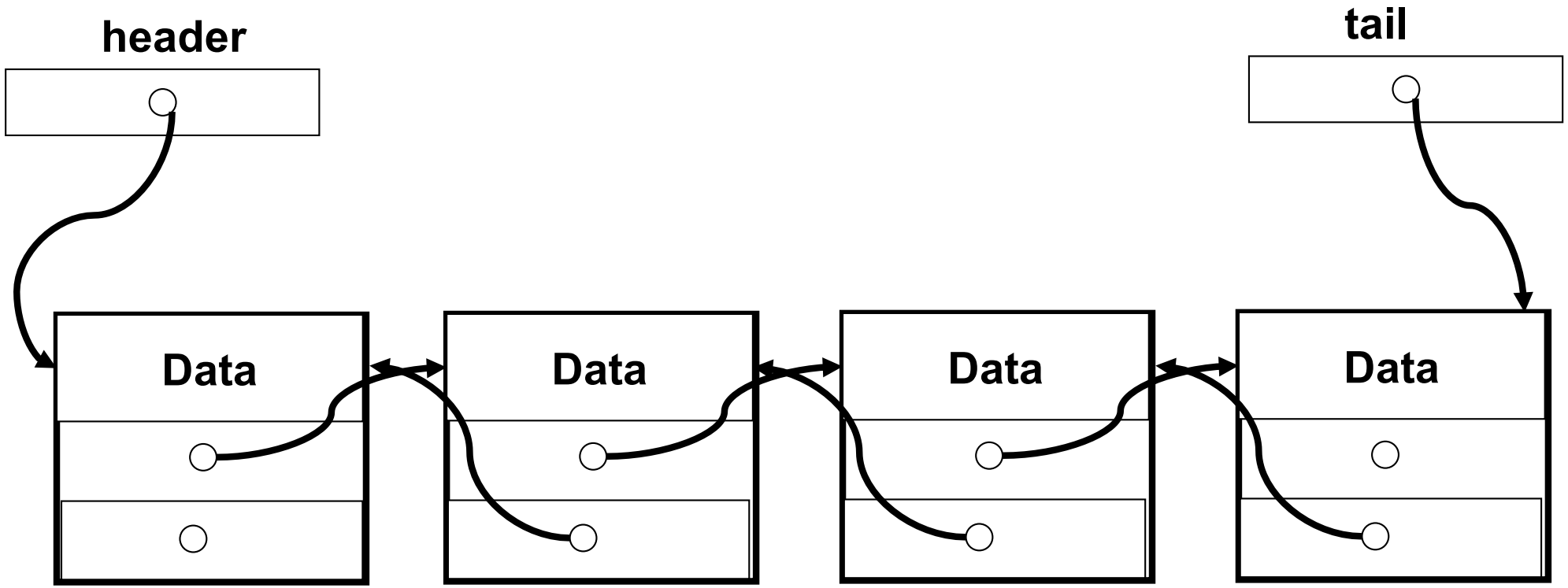
Outline

- Lists
- List operations
- List implementation
- Sentinels
- Double and multiple linked lists
- STL Lists

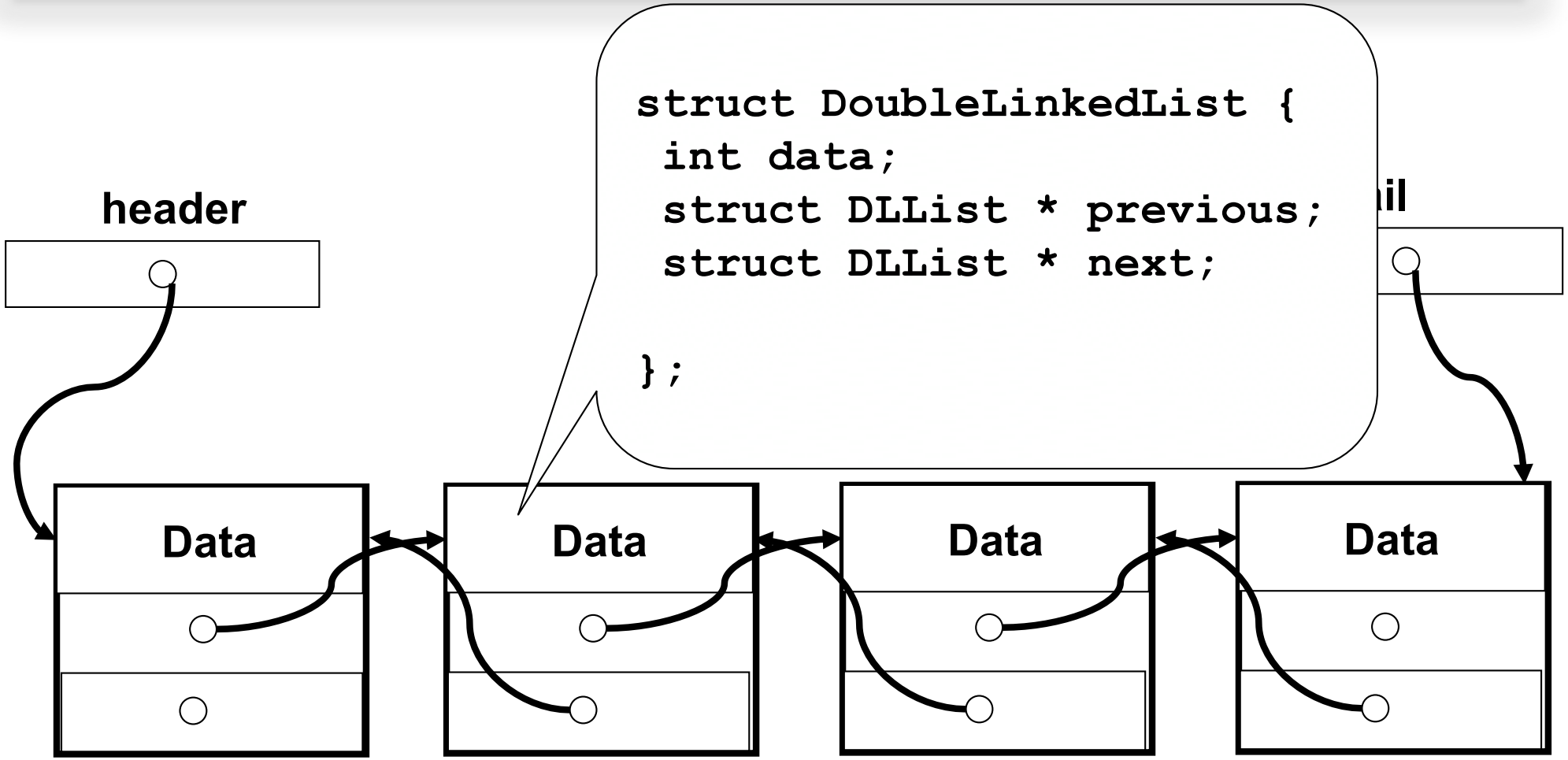
Double and multiple linked lists

- **In the cases when it is necessary to scan the list from the beginning to the end, and from the end to the beginning, it is ok to add a second pointer to every node.**
- **Two external pointers are required:**
  - **Header pointer**
  - **Tail pointer**

Double linked lists

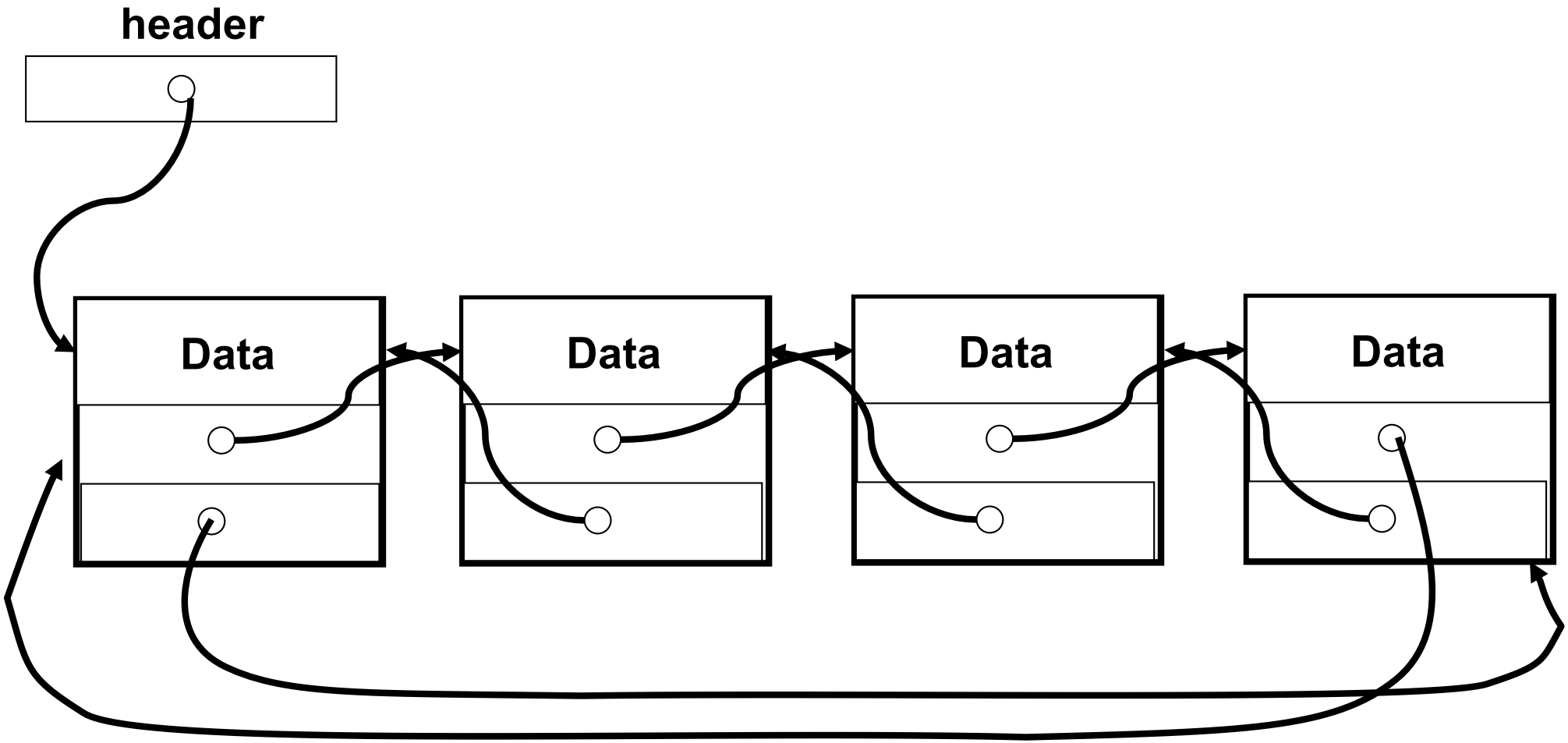


Double linked lists



Circular lists

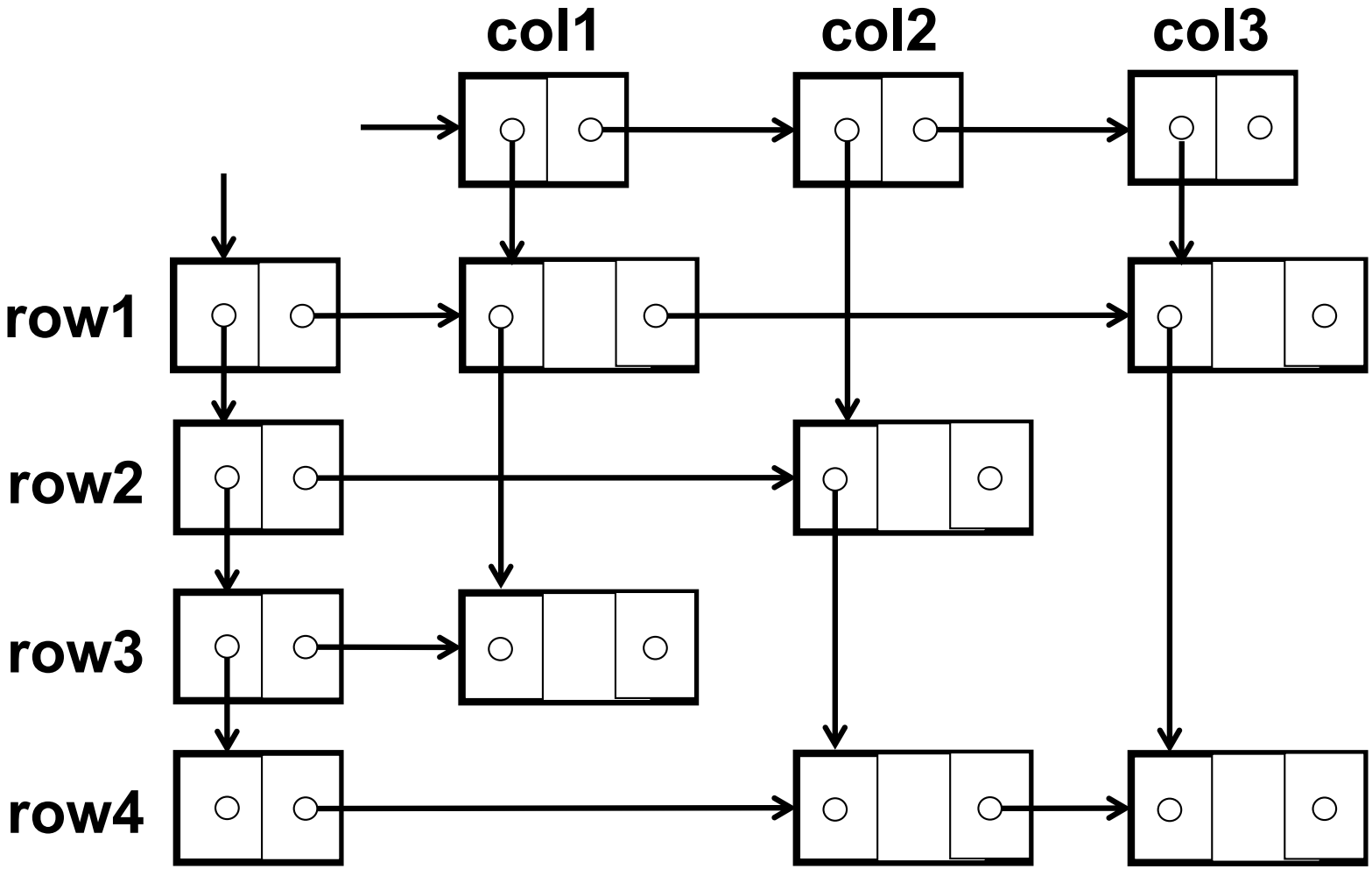
- **The last node (tail) points to the first one (head).**



Multiple lists

- **Every node in the list may point to several nodes.**

Multiple lists



Outline

- Lists
- List operations
- List implementation
- Sentinels
- Double and multiple linked lists
- STL Lists

STL Lists

- **STL list container implements a double linked list**
  - **Iterator is a Bidirectionalterator**
- **C++ v11 introduced a (single) liked list container: forward_list**
  - **In this case, operations are allowed in one direction only... (better performances but less flexibility).**

STL Lists - Insertion

- **Like vectors:**
  - **push_front**
  - **push_back**
  - **insert**

STL Lists - Insertion

- **Like vectors:**
  - **push_front**
  - **push_back**
  - **insert**

insert(iterator, value)
where iterator is not an index
but a real iterator!
(like for vectors!!!!!)

STL Lists - Deletion

- **Like vectors:**
  - **pop_front**
  - **pop_back**
  - **erase**
  - **clear**
  - **remove**

STL Lists - Deletion

- **Like vectors:**
  - **pop_front**
  - **pop_back**
  - **erase**
  - **clear**
  - **remove**

erase(iterator position) → single element
erase(iterator first, iterator last) → from a first to a last set of elements

STL Lists - Deletion

- **Like vectors:**
  - **pop_front**
  - **pop_back**
  - **erase**
  - **clear**
  - **remove**

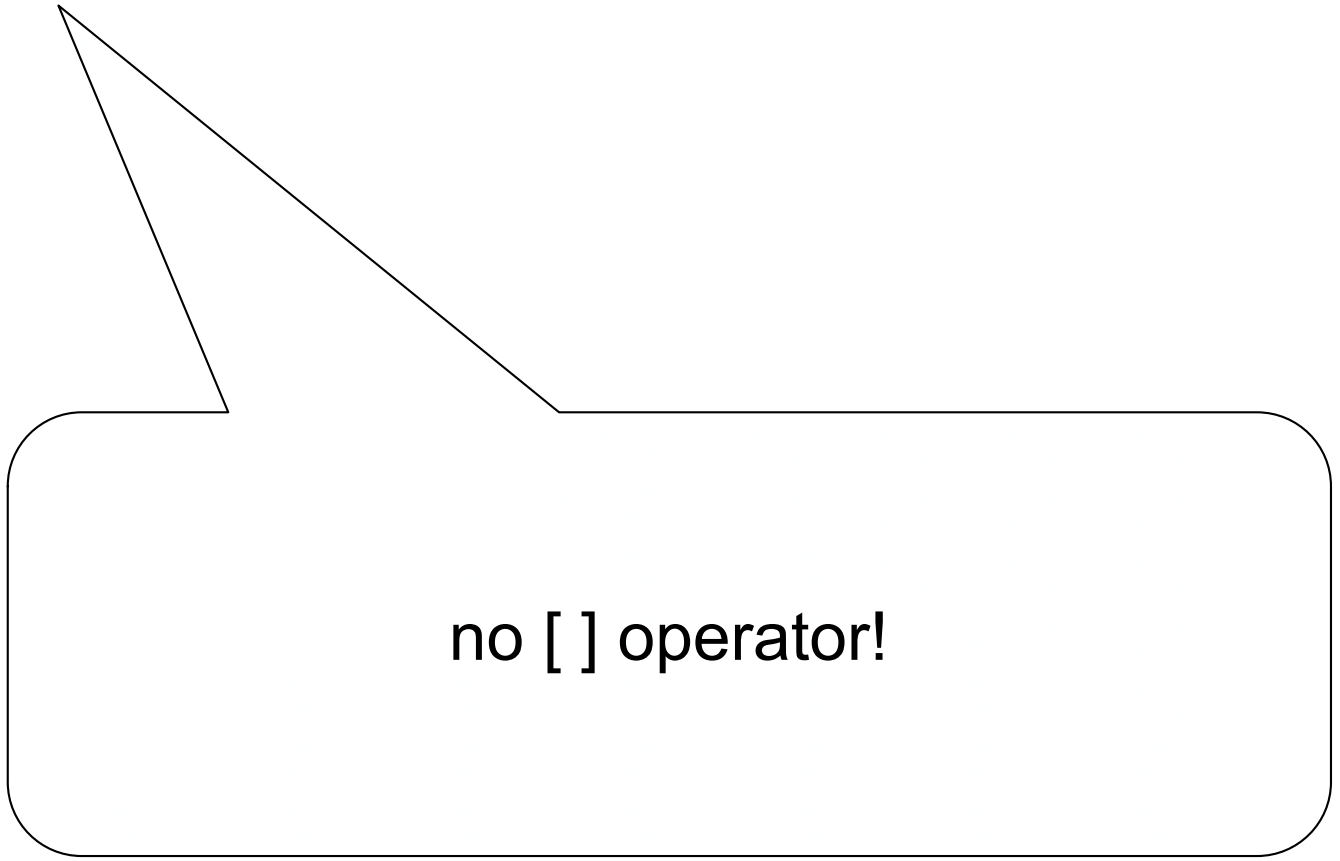
remove(value) → removes from the container all the elements that compare equal to value!

STL Lists - Element access

- **Like vectors:**
  - **front**
  - **back**

STL Lists - Element access

- **Like vectors:**
  - **front**
  - **back**



STL Lists - Specials

- **Not-Like vectors:**
  - **sort**
  - **merge**
  - **unique**
  - **reverse**

STL Lists - Specials

- **Not-Like vectors:**
  - **sort**
  - **merge**
  - **unique**
  - **reverse**

Vectors can have the same algorithms, using STL algorithms instead of specific methods

STL Lists - Specials

- **Not-Like vectors:**
  - **sort**
    - **sort elements by means of a function for the single element comparison you must provide!**
  - **merge**
  - **unique**
  - **reverse**

STL Lists - Specials

- **Not-Like vectors:**
  - **sort**
  - **merge**
    - **merge a list into another one (the one you pass as parameter into the one you called the method from)**
      - **this clear the one you passed.**
    - **overridden with a second version that accept a comparison function too.**
  - **unique**
  - **reverse**

STL Lists - Specials

- **Not-Like vectors:**
  - **sort**
  - **merge**
  - **unique**
    - **removes duplicates**
  - **reverse**

STL Lists - Specials

- **Not-Like vectors:**
  - **sort**
  - **merge**
  - **unique**
  - **reverse**
    - **reverse the list!**

References

- **A.V. Aho, J.E. Hopcroft, J.D. Ullman: "Data Structures and Algorithms," Addison Wesley, Reading MA (USA), 1983 pp 37-74**
- **J. Esakow. T. Weiss "Data structure: an advanced approach using C," Prentice Hall, Englewood Cliffs NJ (USA), 1982 pp 54-237**
- **C.J. Van Wyk: "Data Structures and C Programs," Addison Wesley, Reading MA (USA), 1988 pp 49-128**

References

- **R. Sedgewick:**
  "Algorithms in C,"
  Addison Wesley, Reading MA (USA), 1990
  pp 15-34
- **E. Horowitz, S. Sahni:**
  "Fundamentals of Computer Algorithms,"
  Pittman, London (UK), 1978
  pp 65-202
- **N. Wirth:**
  "Algorithms + Data Structures = Programs,"
  Prentice Hall, Englewood Cliffs NJ (USA), 1976
  pp 162-188


Малые Автюхи, Калинковичский район, Республики Беларусь