Linked Memory

Prof. Wade Fagen-Ulmschneider

Linked memory stores data together with a "link" to the location (in memory) of the next list node:

List Nodes

A **list node** refers to pair of both the data and the link.

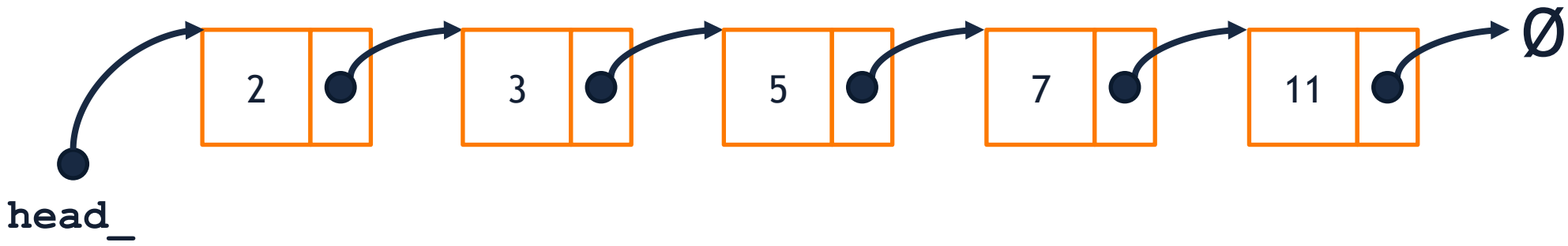
Graphicly:

C++ ListNode Class:

```
template <typename T>
class ListNode {
public:
  T & data;
  ListNode *next;
  ListNode(T & data) : data(data), next(NULL) { }
};
```

Linked List

Zero or more `ListNode` elements linked together form a **Linked List.**



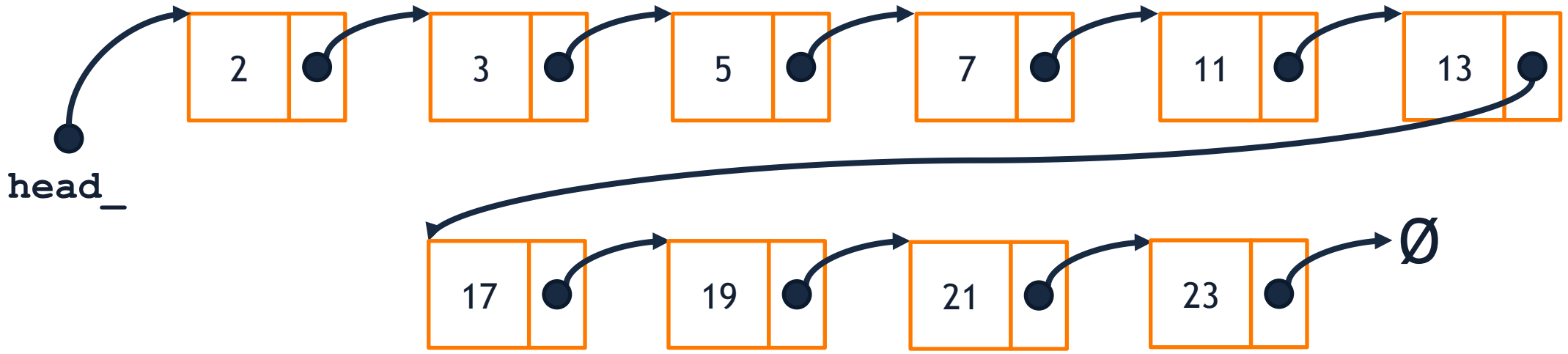
- A pointer called the "**head pointer**" stores the link to the beginning of the list.
- A pointer to NULL (Ø) marks the end of the list.

linked-memory/List.h

```
8  #pragma once
9
10 template <typename T>
11   class List {
12     public:
13       const T & operator[](unsigned index);
14       void insertAtFront(const T & data);
15
16     private:
17       class ListNode {
18         public:
19           const T & data;
20           ListNode *next;
21           ListNode(const T & data) : data(data), next(nullptr) { }
22       };
23
24     ListNode *head_; /*< Head pointer for our List */
27 };
28
29 #include "List.hpp"
```

List::get

Objective: Return the element at index **k**.



linked-memory/List.cpp

```
10 template <typename T>
11 const T & List<T>::operator[](unsigned index) {
12   // Start a `thru` pointer to advance thru the list:
13   ListNode *thru = head_;
14
15   // Loop until the end of the list (or until a `nullptr`):
16   while (index > 0 && thru->next != nullptr) {
17     thru = thru->next;
18     index--;
19   }
20
21   // Return the data:
22   return thru->data;
23 }
```

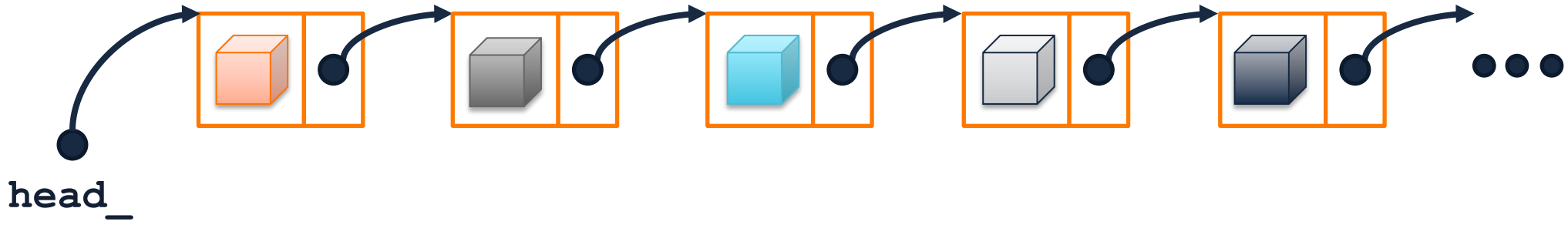
List Capacity

- In a list, the time it takes to access a given index grows based on the size of the list.
  - *In contrast, an array can access any element in a constant, fixed amount of time. Therefore, for accessing a given index, an array is faster than a list.*

Array: O(1)
List: O(n)

List::insert

Objective: Add an element to the list.



linked-memory/List.hpp

```
25 template <typename T>
26 void List<T>::insertAtFront(const T & data) {
27   // Create a new ListNode on the heap:
28   ListNode *node = new ListNode(data);
29
30   // Set the new node's next pointer point the current
31   // head of the List:
32   node->next = head_;
33
34   // Set the List's head pointer to be the new node:
35   head_ = node;
36 }
```

List Capacity

- In a list, the **capacity** is bounded only by the memory available on the system.
  - *In contrast, an array has a fixed capacity. A list is a more flexible data structure than an array.*

Data Types

- In both arrays and lists, all data within an instance of a container must be the same type.
  - An integer {array, list} must only contain integers.
  - A string {array, list} must only contain strings.
  - …

linked-memory/main.cpp

```
8  int main() {
9    List<int> list;
10
11   std::cout << "Inserting element 3 at front..." << std::endl;
12   list.insertAtFront(3);
13   std::cout << "list[0]: " << list[0] << std::endl;
14
15   std::cout << "Inserting element 30 at front..." << std::endl;
16   list.insertAtFront(30);
17   std::cout << "list[0]: " << list[0] << std::endl;
18   std::cout << "list[1]: " << list[1] << std::endl;
19
20   return 0;
21 }
```

Linked Memory

- Linked memory stores data in “nodes” linked together by “links” (pointers).

- A basic linked memory structure is a Linked List, which consists of zero or more **ListNode**s lined together and a **head** pointer.

- A linked list provides a flexible alternative to an array.