Managing Linked Lists with std::list

Giovanni Dicanio
AUTHOR, SOFTWARE ENGINEER

https://blogs.msmvps.com/gdicanio

Overview

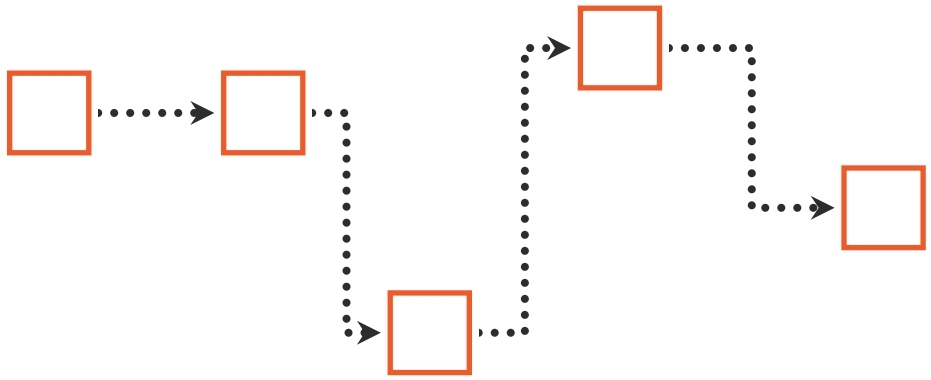
Introduction to std::list

- std::list vs. std::vector

Common operations

Demo code and fixing a subtle bug

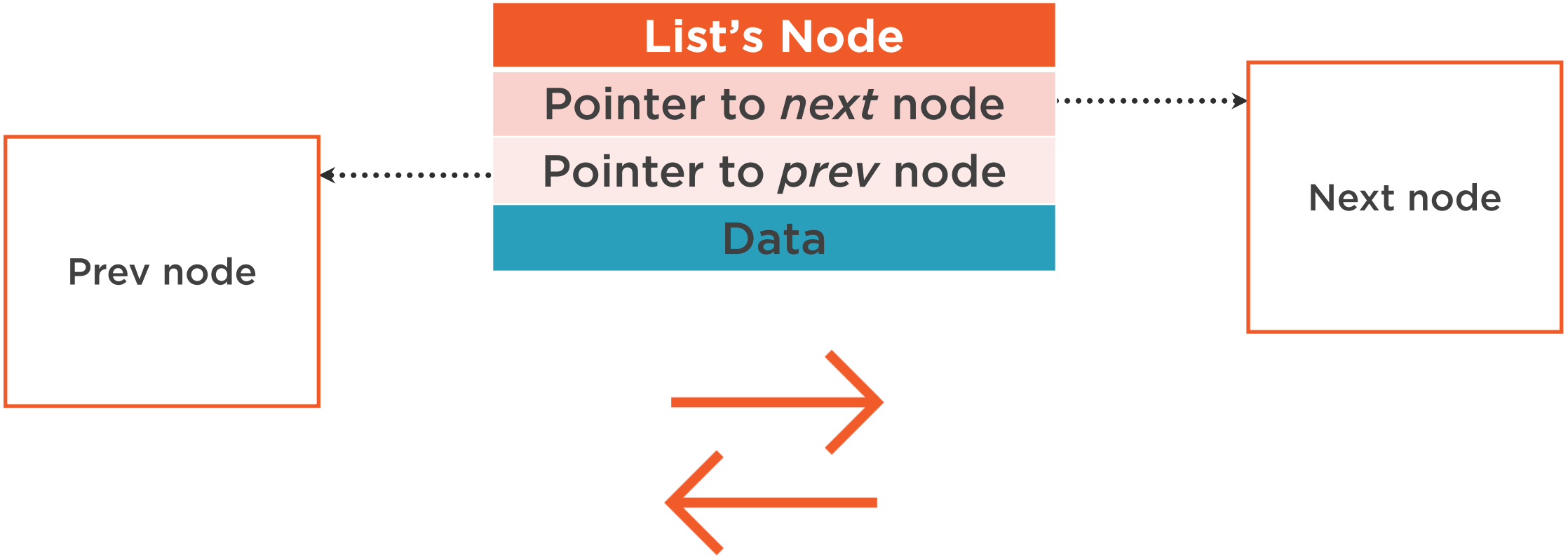
Linked List Data Structure



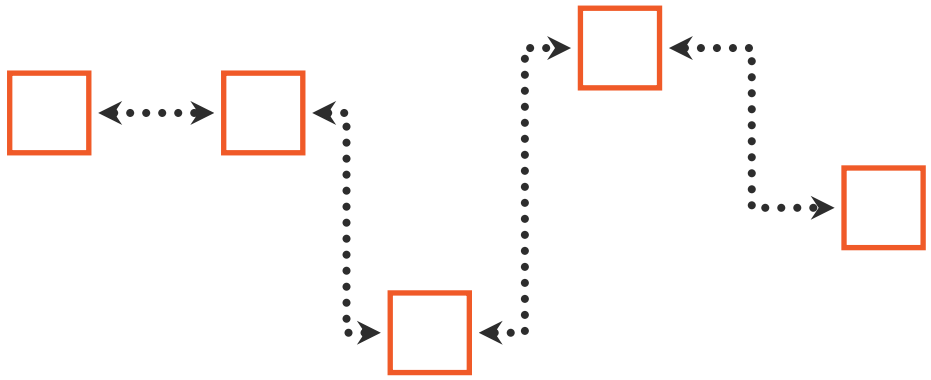
Introducing Node-based Data Structures: Linked Lists

Course:
"Introduction to Data Structures and Algorithms in C++"

Doubly-linked List

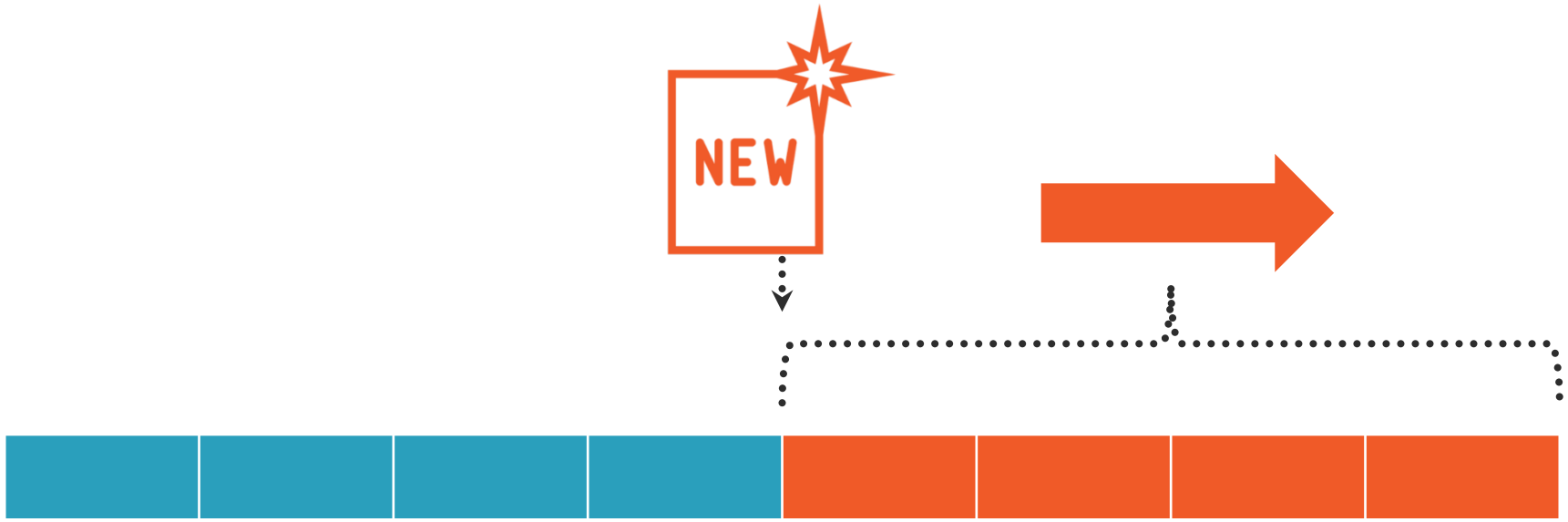


std::list's Features



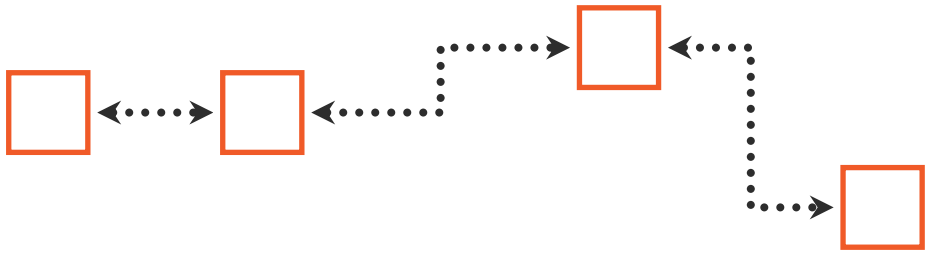
Fast *constant-time O(1)*
insertion and removal *anywhere*

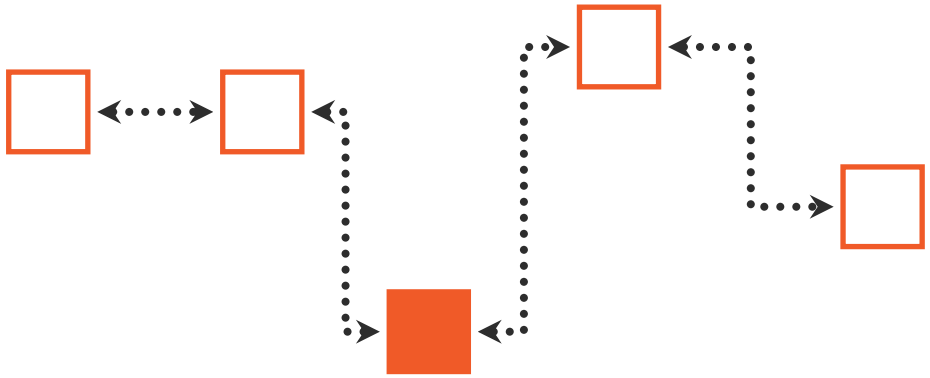
Inserting in the Middle of std::vector



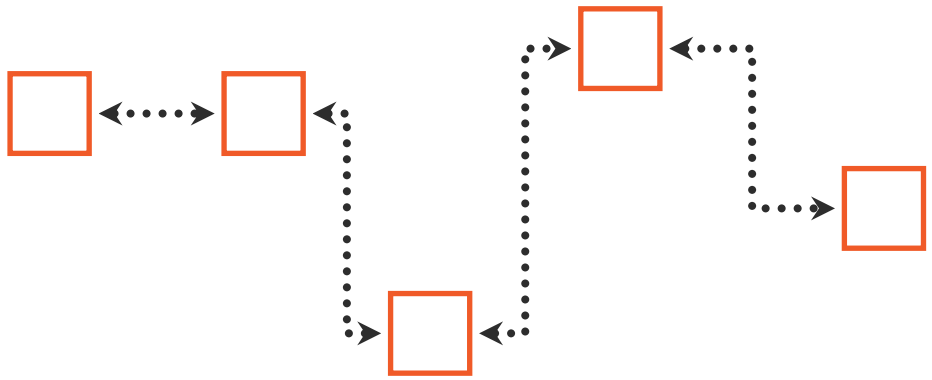
Need to make room
for the new element

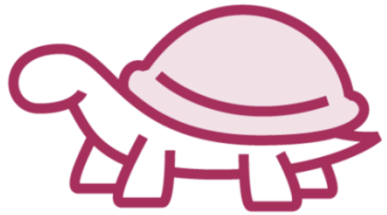
Iterator Stability: std::list vs. std::vector

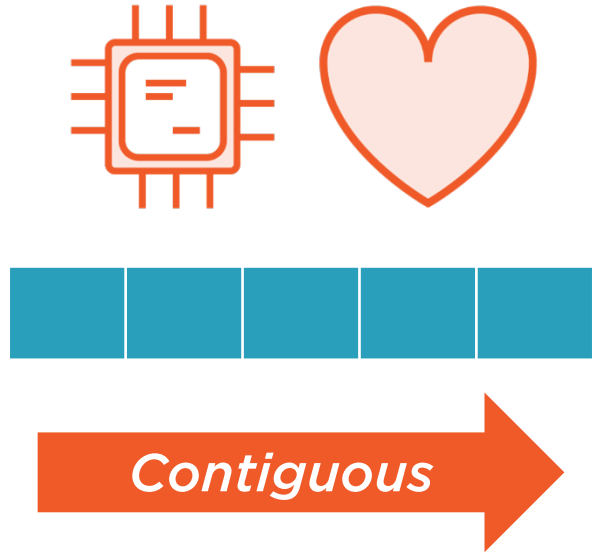




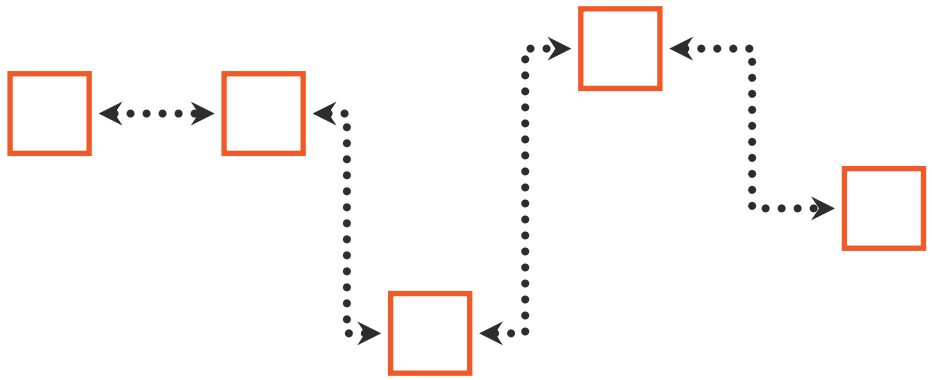
Memory Layout: std::list vs. std::vector



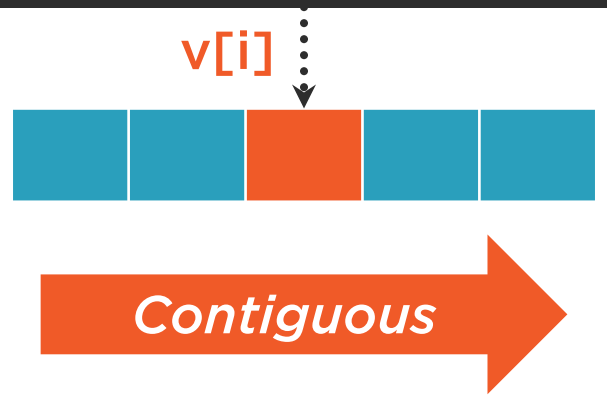




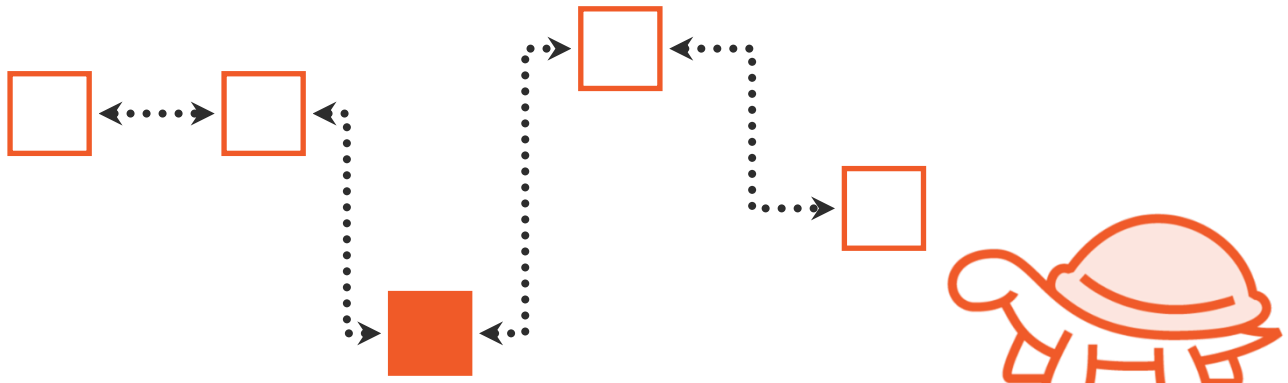
Element Access: std::list vs. std::vector



Fast O(1) element access
via integer index



Element Access: std::list vs. std::vector



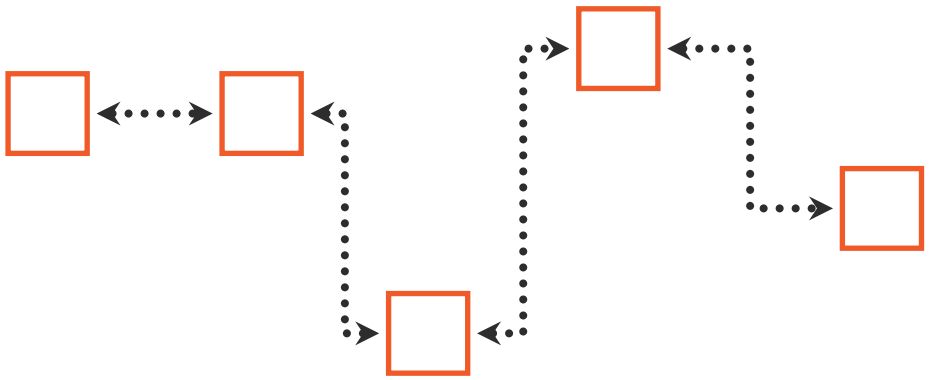
Traverse *all previous* nodes

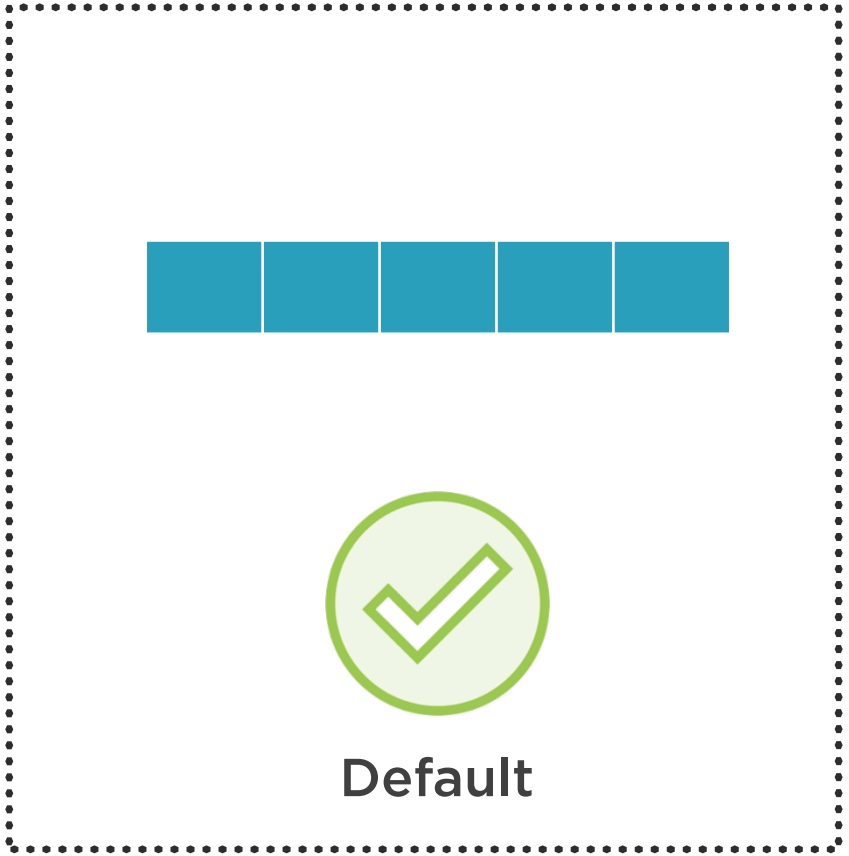
O(N)

Fast O(1) element access via integer index

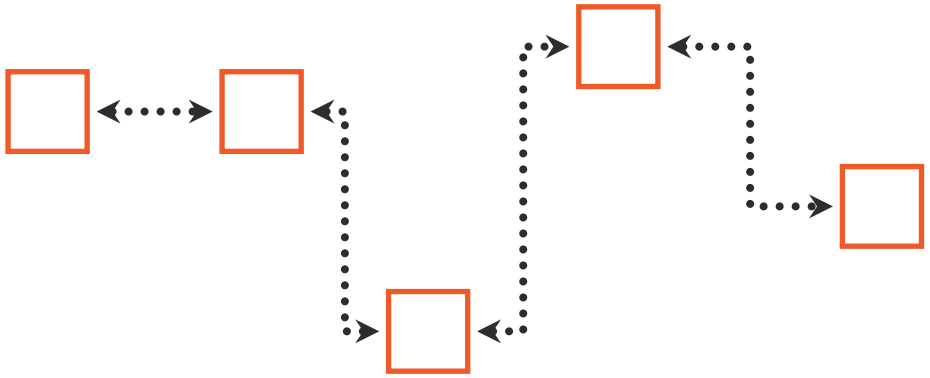
Contiguous

std::list vs. std::vector





std::list vs. std::vector



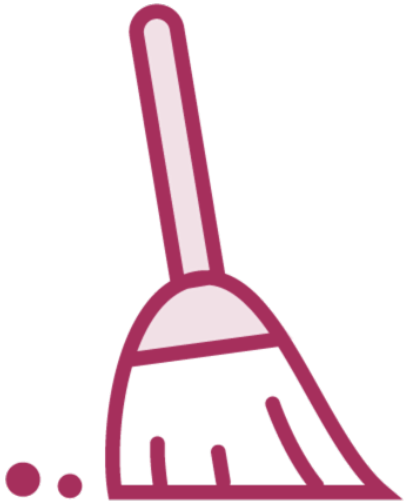
```cpp
// Initialize std::list with given values
std::list<int> l{11, 22, 33, 44, 55, 66};

// Create an empty list
std::list<int> l{};
std::list<int> l;
```

Creating a std::list

Common Operations with std::list

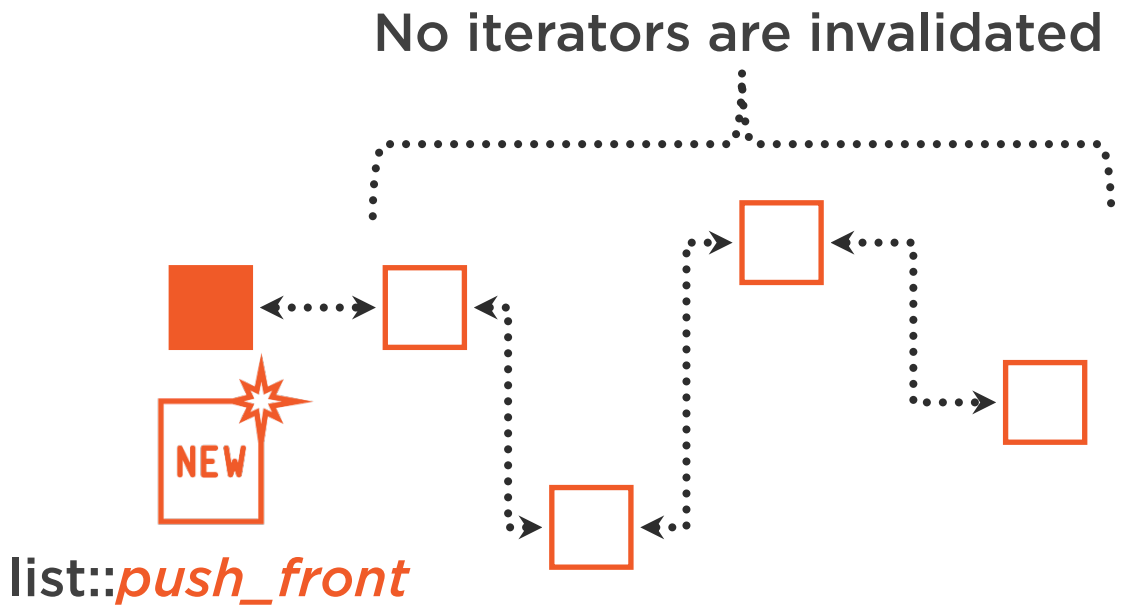
size



clear

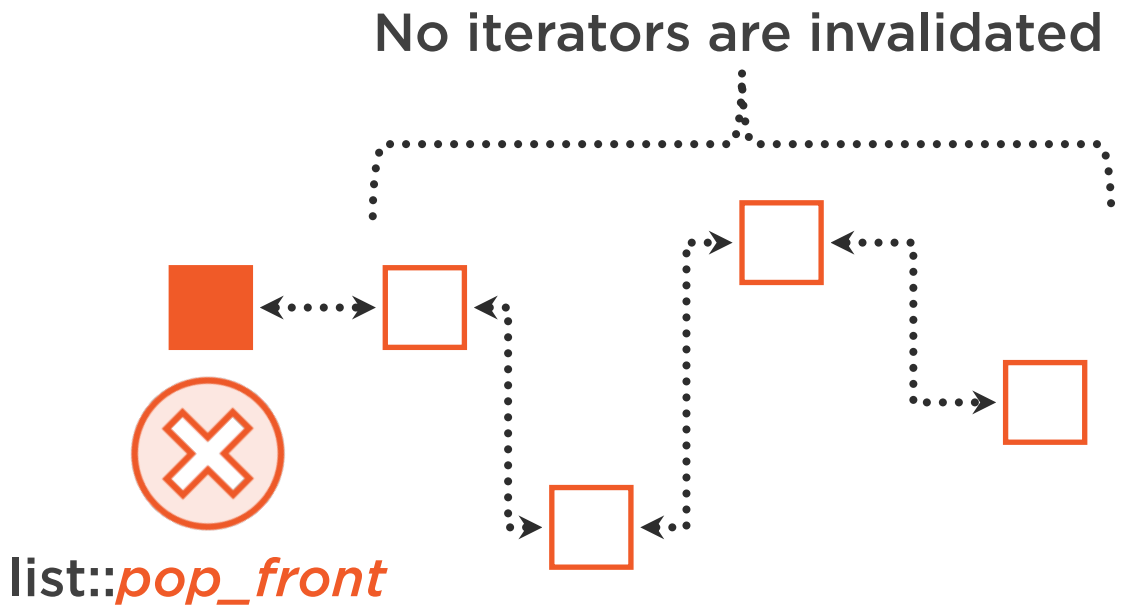
empty

Inserting a New Element at the Beginning

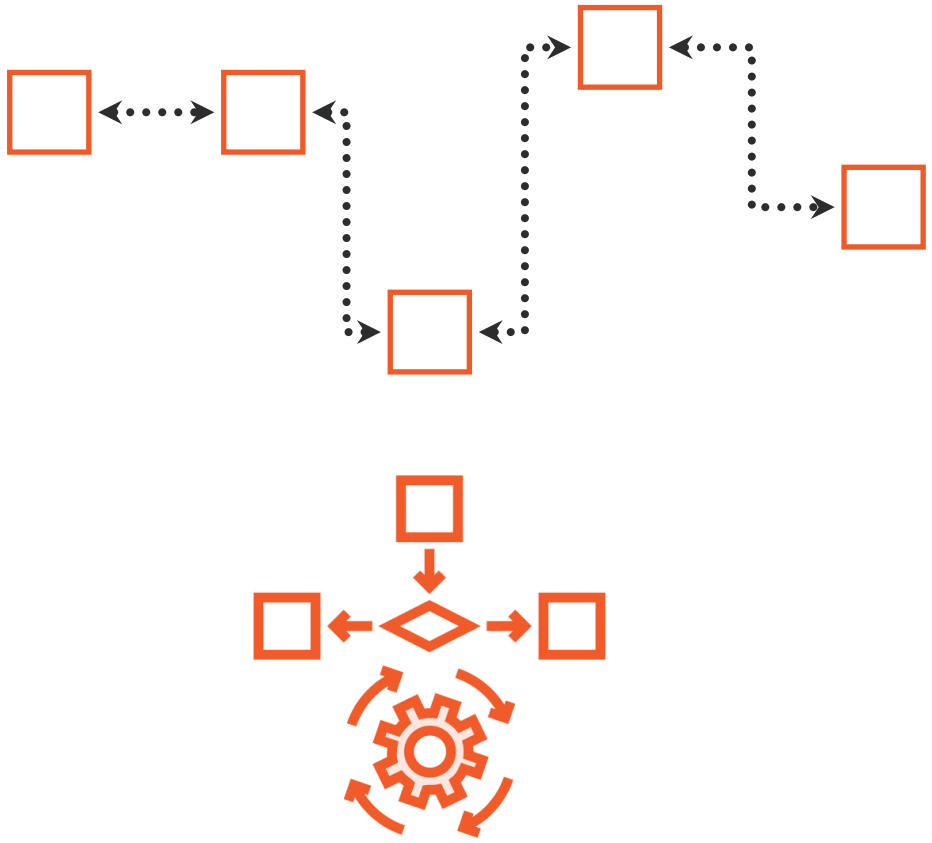


Constant-time
fast operation

Removing the First Element



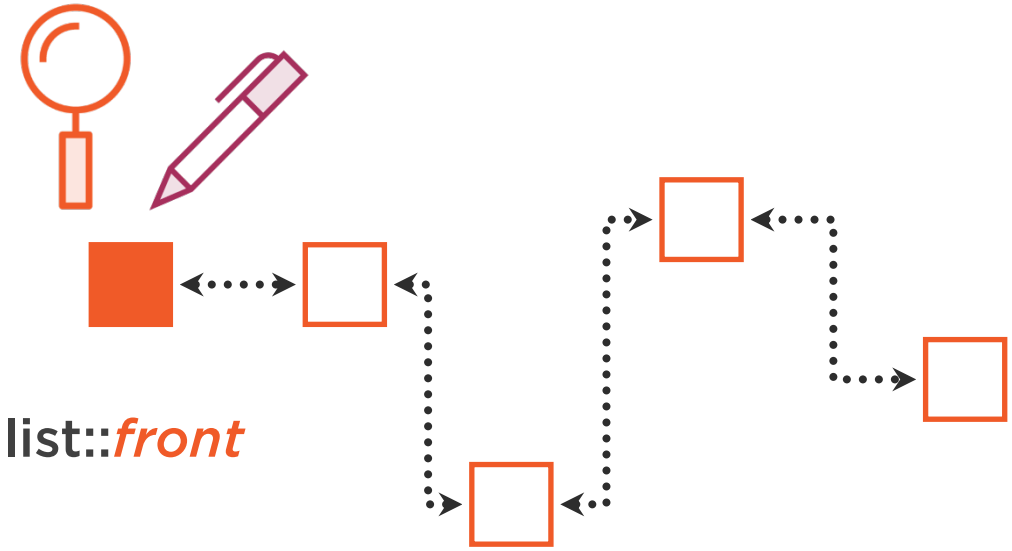
List Operations Under the Hood



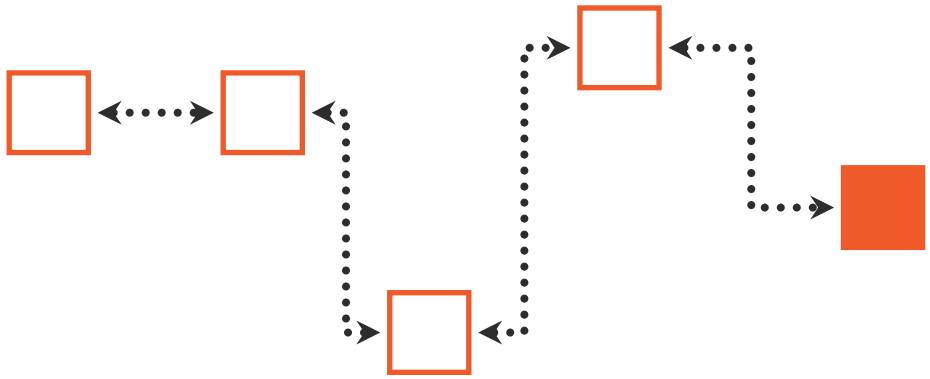
Introducing Node-based Data Structures: Linked Lists

Course:
"Introduction to Data Structures and Algorithms in C++"

Access the First Element



Operations on the Last Element



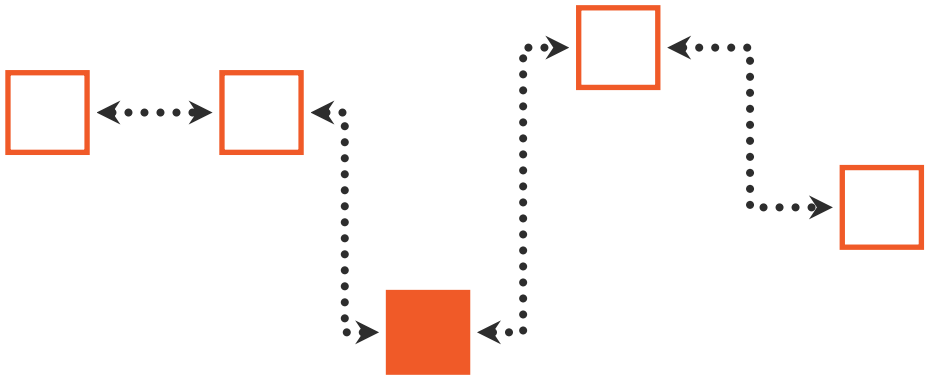
list::*push_back*

list::*pop_back*

list::*back*

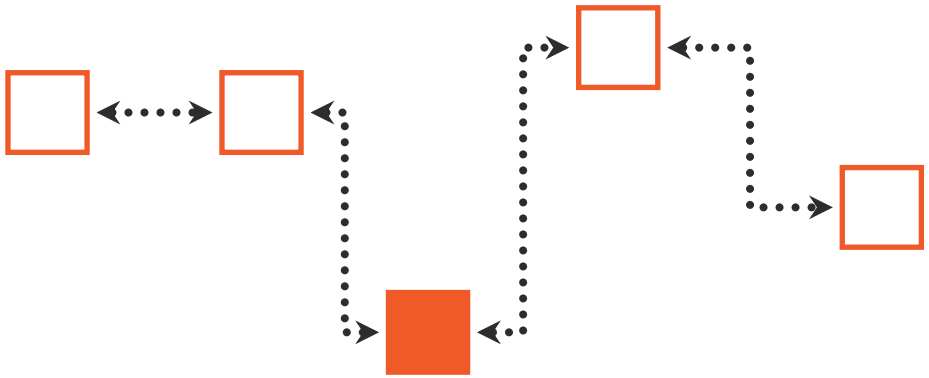
```
// Insert value before pos
myList.insert(pos, value);
```

Inserting an Element at the Specified Location



```
// Insert value before pos
myList.insert(pos, value);
```

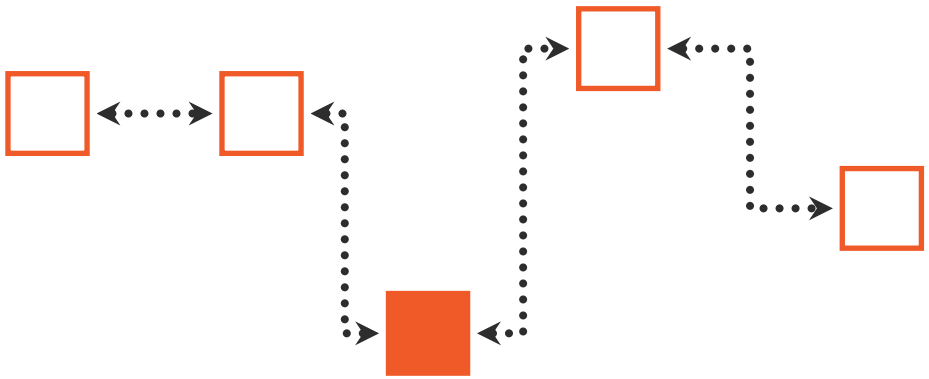
Inserting an Element at the Specified Location

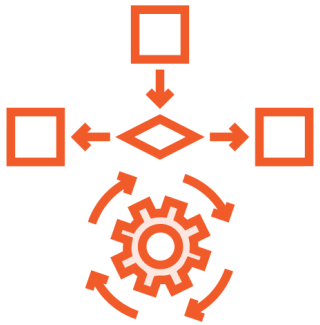


```
// Insert value before pos
myList.insert(pos, value);
```

Same public interface of std::vector

Inserting an Element at the Specified Location

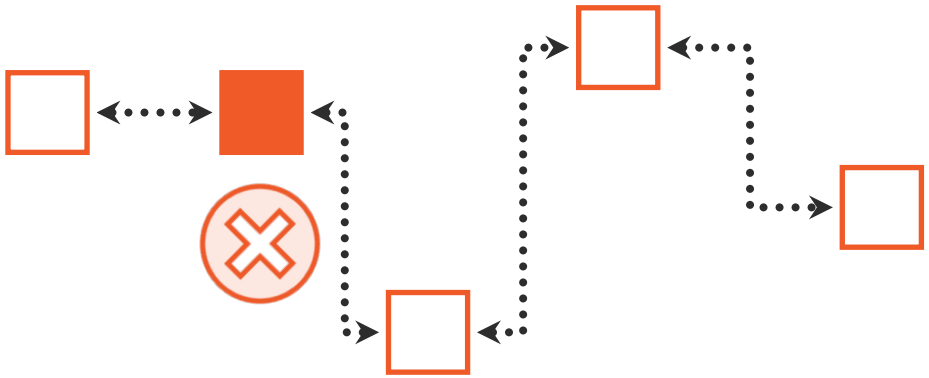




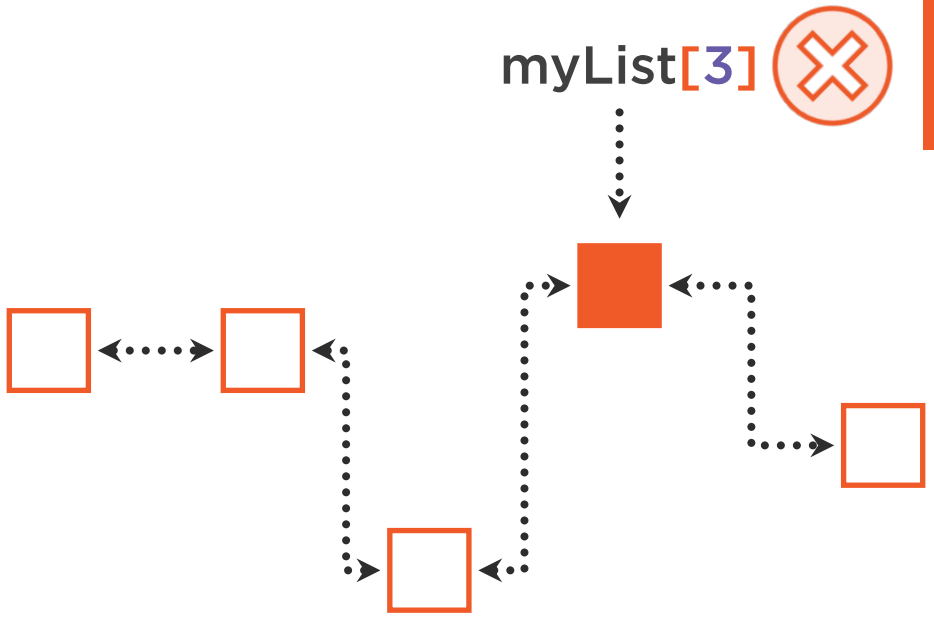
```
// Remove all elements that are equal to value
myList.remove(value);

// Remove all elements that satisfy condition
myList.remove_if(condition);
```

Removing Elements from std::list

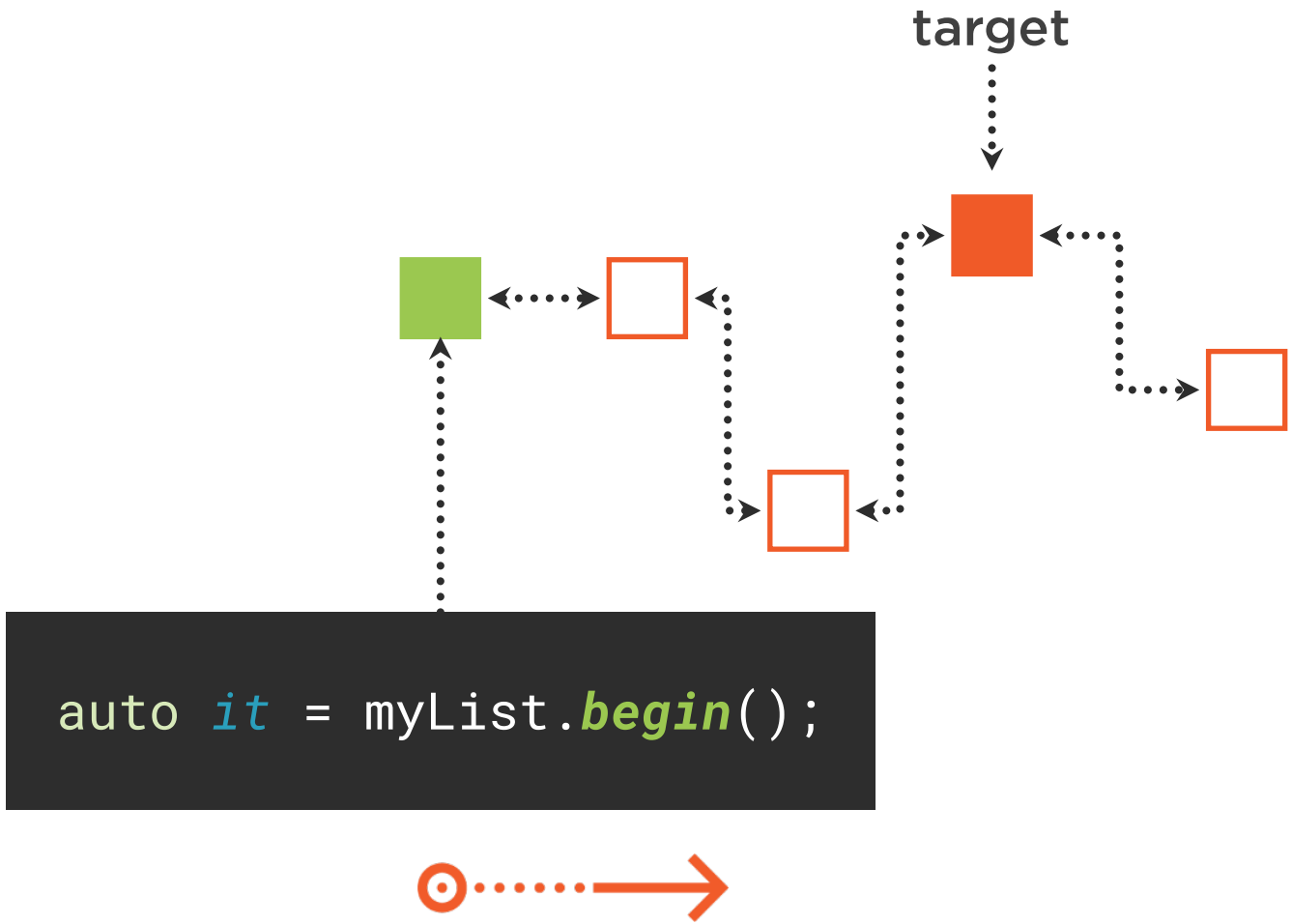


Accessing Elements (Nodes) in std::list

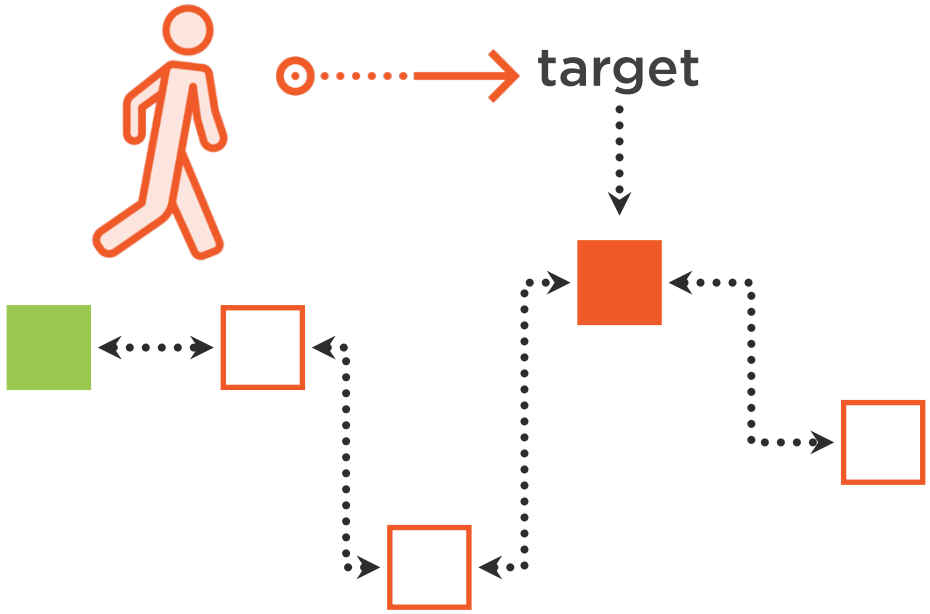


operator[]
not available

Accessing Elements (Nodes) in std::list

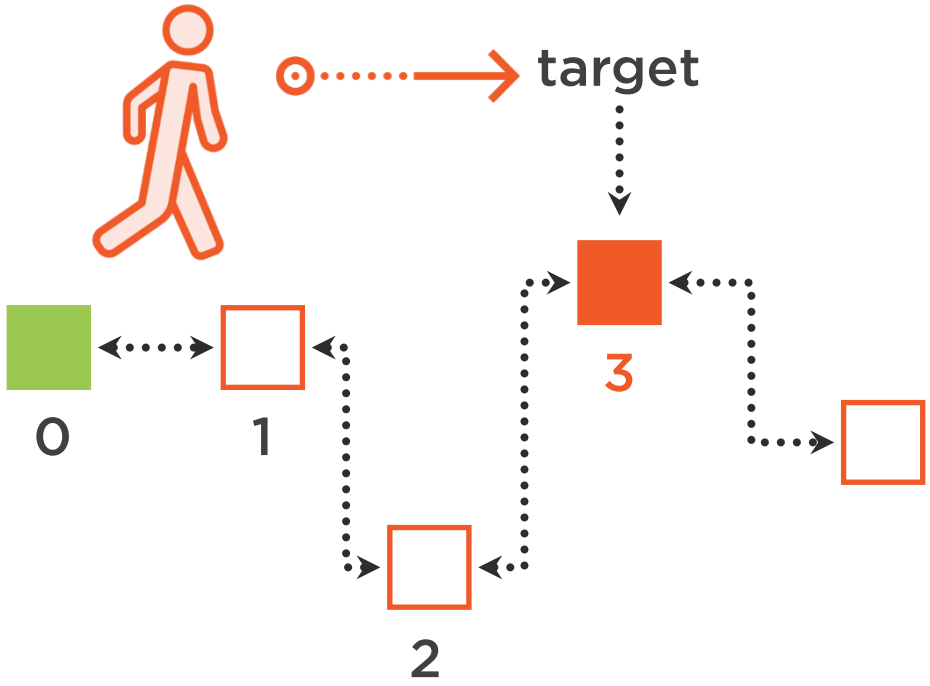


Accessing Elements (Nodes) in std::list



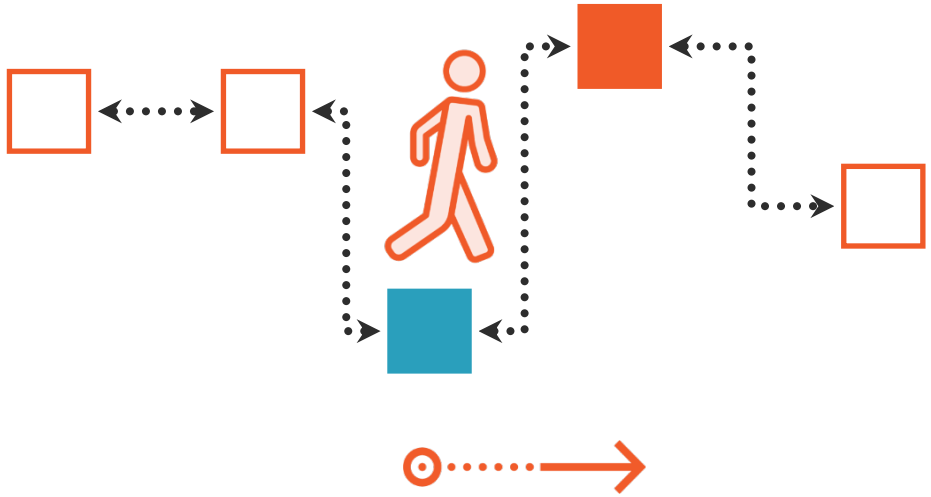

```
std::advance(it, steps);
```

Accessing Elements (Nodes) in std::list

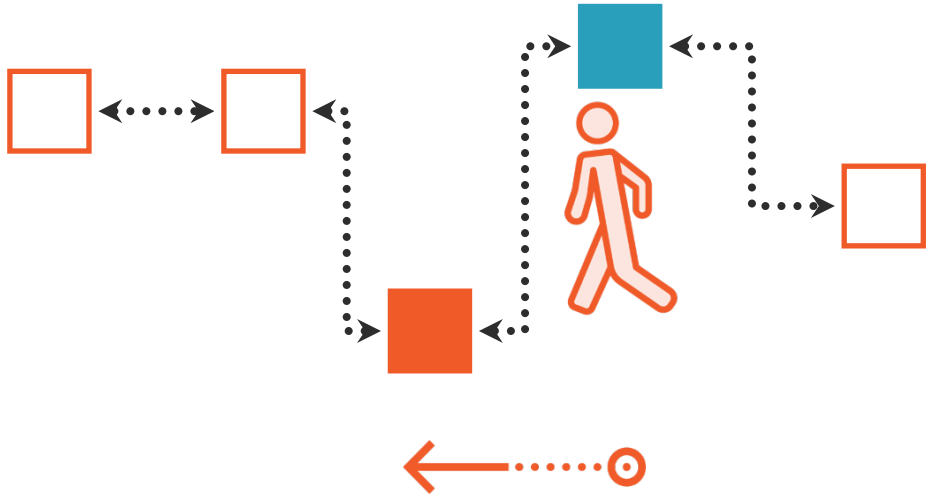



```
std::advance(it, 3);
```

std::list Provides Bidirectional Iterators

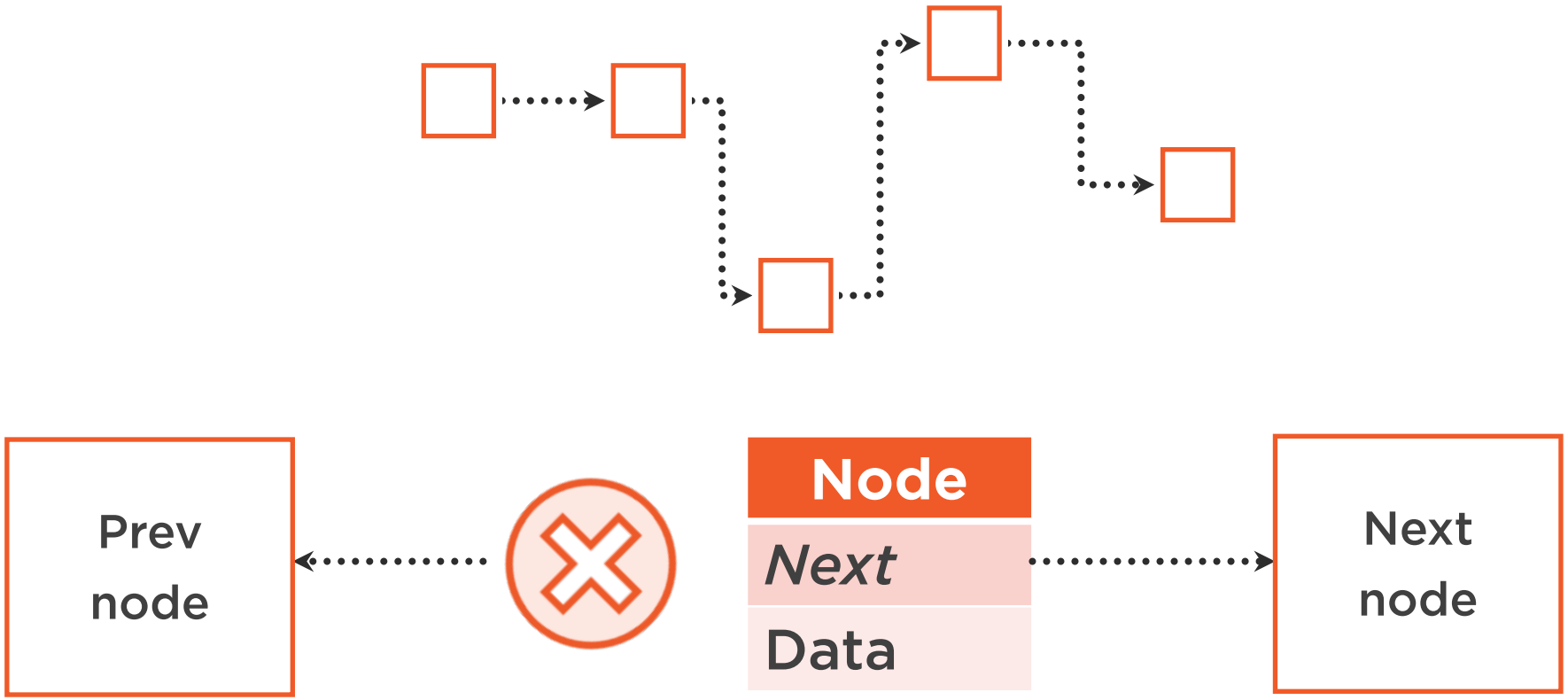


```
++it;
```

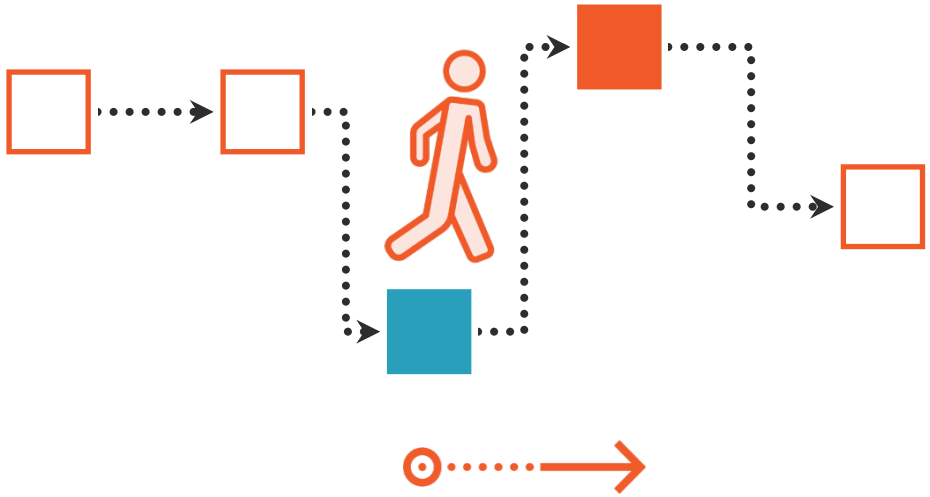


```
--it;
```

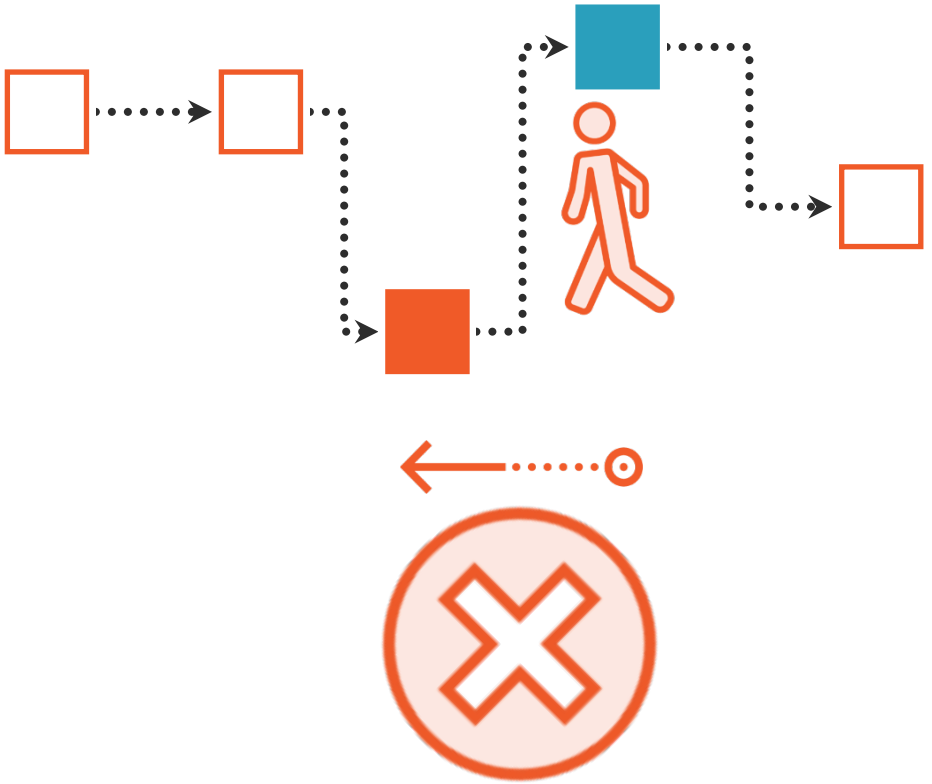
std::forward_list Is a Singly-linked List



forward_list Supports Forward-only Iteration



```
++it;
```



```
// The planets container is a std::list
sort(begin(planets), end(planets));
```

Bug: Sorting std::list

std::*sort* expects *random access* iterators

```
// The planets container is a std::list
sort(begin(planets), end(planets));
```

Bug: Sorting std::list

std::*sort* expects *random access* iterators

std::*list* provides *bidirectional* iterators

```
// The planets container is a std::list

sort(begin(planets), end(planets));
```

Bug: Sorting std::list

```
In file included from ListSortBug.cpp:3:
In file included from /usr/bin/../lib/gcc/x86_64-linux-gnu/8/../../../../include/c++/8/alg
orithm:62:
/usr/bin/../lib/gcc/x86_64-linux-gnu/8/../../../../include/c++/8/bits/stl_algo.h:1883:18:
error:
      invalid operands to binary expression
      ('std::_List_iterator<std::__cxx11::basic_string<char, std::char_traits<char>,
      std::allocator<char> > >' and 'std::_List_iterator<std::__cxx11::basic_string<char,
      std::char_traits<char>, st
      if (__last - __first > i
          ~~~~~~ ^ ~~~~~~~
/usr/bin/../lib/gcc/x86_64-linux                                    tl_algo.h:1971:9: n
ote:
      in instantiation of function template specialization
      'std::__final_insertion_sort<std::_List_iterator<std::__cxx11::basic_string<char,
      std::char_traits<char>, std::allocator<char> > >,
      __gnu_cxx::__ops::_Iter_less_iter>' requested here
          std::__final_insertion_sort(__first, __last, __comp);
                ^
/usr/bin/../lib/gcc/x86_64-linux-gnu/8/../../../../include/c++/8/bits/stl_algo.h:4834:12:
note:
      in instantiation of function template specialization
      'std::__sort<std::_List_iterator<std::__cxx11::basic_string<char,
      std::char_traits<char>, std::allocator<char> > >,
        gnu_cxx::__ops::_Iter_less_iter>' requested here
```

std::sort complains about
list iterator operator-

Sorting std::list


```
// Invoke the std::list's sort method

planets.sort()
```

Sorting std::list

Summary

- Introduction to std::list
- Pros and cons of std::list
- Important operations (insertion, removal, element access)
- Reusing Standard Library's algorithms (e.g. std::find)
- Subtle sorting bug, and how to fix it

Thank You!