Data Structure

元智競技程式人才培育社 資料結構簡報

Report Outline

01 **Common Data Structures**
A Brief Introduction to Data Structures

02 **Introductions to C++ STL**
Tell You How to Use and Introduce Its Features

03 **Competitor Customized Data Structures**
Tell You How to Use and Introduce Its Features in Detail

04 **Chapter Integration**
A Concise Summary and Quick Overview

05 **From My Perspective**
From A Contestant's Perspective, I Would Like to Say...

01 Common Data Structures

A Brief Introduction to Data Structures

C++ STL

Standard Template Library

STL is *a collection of pre-written algorithms and data structures*

1. Provides access to complex algorithms and data structures
2. Avoid implementing above mentioned components during competitions
3. Saves time and significantly reduces the likelihood of bugs
4. *Can be used without a deep understanding* of their underlying principles

However, if you have the ability, it is advisable to grasp these concepts.

Some Examples

Vector | Stack | Queue | Deque | Priority_queue

Most Commonly Used in Graph Theory

Most Commonly Used

Vector Weakened Version

Set | Multiset | Map | Unordered_map

These guys can sort the data automatically

01 Common Data Structures

A Brief Introduction to Data Structures

Competitor Customized

Although STL provided by C++ is quite comprehensive most of the time, but it may not be sufficient enough for competitive programming in certain scenarios. In these cases, competitor customized data structures become essential. However, these data structures usually take a considerable amount of time to implement manually during competitions, which is undoubtedly time-consuming. Therefore, contestants must carefully evaluate whether the use of these structures is truly necessary before making a decision.

Examples

Binary Index Tree

Abbreviation BIT
Storing Data By Bit Concept
Dynamic Prefix Sum
Support Modification

Segment Tree

Advanced BIT
Storing Data By Bit Concept
Range Queries
Support Modification
Highly Customizable

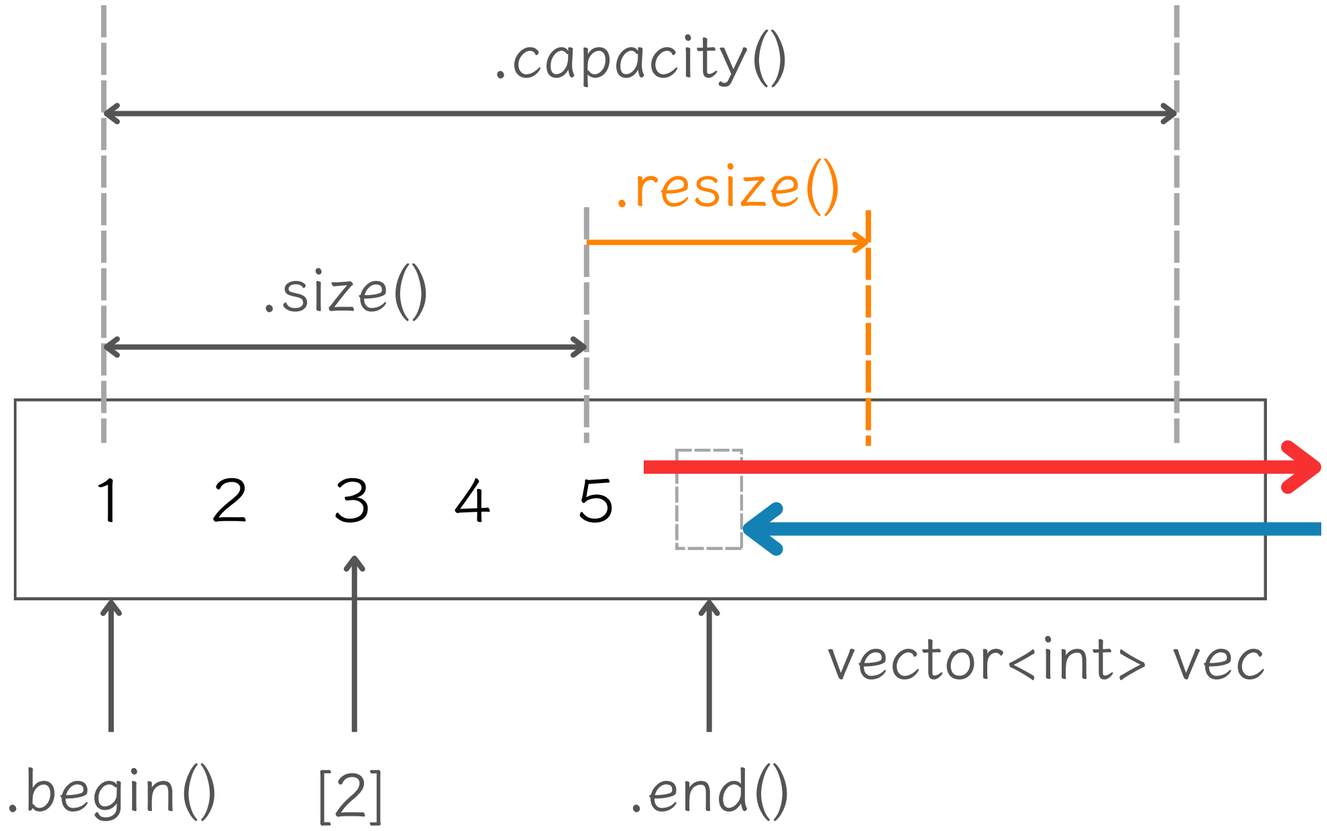
Disjoint Set Union

Abbreviation DSU
Frequently Utilized To CC
Deal With The Union and
Find Queries in Disjoint Set

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector



.empty()
.clear()

.pop_back()

.push_back(val)
.emplace_back(val)

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

Decleration

```
vector<value_type> v1;
vector<value_type> v2( default size );
vector<value_type> v3( default size, default argument );
vector<value_type> v4 = { default argument, default argument, ..... };
vector<value_type> v5 ( { default argument, default argument, ..... } );
vector<value_type> v6 { default argument, default argument, ..... };
vector<value_type> v7 = another vector;
```

Traverse

```
for( size_type it = 0 ; it < v.size() ; ++it )
for( vetor<value_type>::iterator it = v.begin() ; it != v.end() ; ++it )
for( auto it = v.begin() ; it != v.end() ; ++it )
for( auto it : v )
for( auto& it : v )
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

```
int main(){
   vector<int> v1;
   vector<int> v2(3);
   vector<int> v3(6, 10);
   vector<int> v4 = { 1, 2, 3 };
   vector<int> v5 = v4;
   for( int it = 0 ; it < v2.size() ; ++it ) cout << v2[it] << "  ";
   for( vetor<int>::iterator it = v3.begin() ; it != v3.end() ; ++it )
      cout << *it << " ";
   for( auto it = v4.begin() ; it != v4.end() ; ++it )
      cout << *it << " ";
   for( auto it : v5 ) cout << it << " ";
}
```

Output

```
0 0 0 → Random
10 10 10 10 10 10
1 2 3
1 2 3
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

Initialize

With { value1, value2, ……}

Time Complexity : O(N)

```
#include <vector>
vector<int> v = { 1, 2, 3 };
```

With a Specific Number of Elements

Time Complexity : O(N)

```
#include <vector>
vector<int> v(5);
```

Specific Number and Value of Elements

Time Complexity : O(N)

```
#include <vector>
vector<int> v(5, 10);
```

From a Different Vector

Time Complexity : O(N)

```
#include <vector>
vector<int> original = {1, 2, 3, 4, 5};
vector<int> v(original.begin(), original.end());
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

Initialize

Initialize to 2D Vector of Size (N, M)

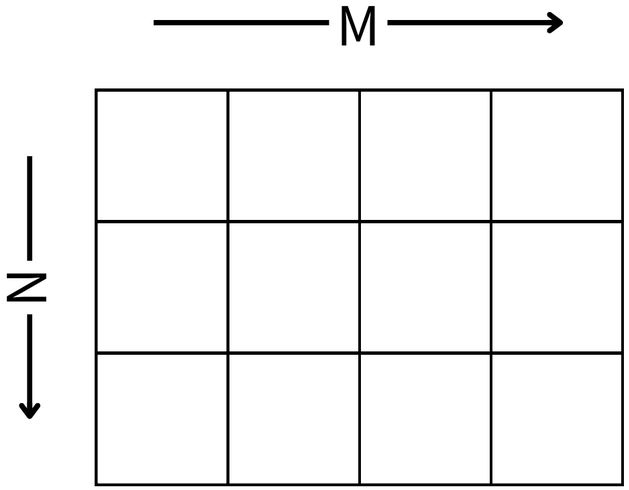
Time Complexity : O(NM)

```
#include <vector>
vector<vector<int>> v(3, vector<int>(4));
```

Specific Number and Value of Elements

Time Complexity : O(NM)

```
#include <vector>
vector<int> v(3, vector<int>(4));
vector<int> v = { { 1, 2, 3 },
                  { 4, 5, 6 } };
```



02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

Retrieve Value

*Checking An Index Out of Range Will Raise An out_of_range Exception

With operator[]

Time Complexity : O(1)

```
#include <vector>
vector<int> v = { 2, 4, 6, 8 };
cout << v[0] << "\n";   /* Output : 2 */
```

Safely Retrieve Values Using .at()

Time Complexity : O(1)

```
#include <vector>
vector<int> v = { 2, 4, 8, 6 };
cout << v.at(2) << "\n";   /* Output : 8 */
```

Retrieve the First and Last Elements

Time Complexity : O(1)

```
#include <vector>
vector<int> v = { 4, 8, 7, 6, 3 };
cout << v.front() << " " << v.back() << "\n";
                                   /* Output : 4 8 */
```

Retrieve Data Pointer

Time Complexity : O(1)

```
#include <vector>
vector<int> v = { 1, 2, 3, 4, 5 };
int* ptr = v.data();
cout << *ptr;   /* Output : 1 */
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

Capacity

Retrieve the Total Number of Elements

Time Complexity：O(1)

```
#include <vector>
vector<int> v = { 2, 5, 6, 8 };
cout << v.size() << "\n";   /* Output：4 */
```

Check Whether the Vector is Empty

Time Complexity：O(1)

```
#include <vector>
vector<int> v = { 2, 4, 8, 6 };
cout << v.empty();   /* Output：0（False） */
```

Change the Size of Vector

Time Complexity：Worse Case O(N)

```
vector<int> v = { 1, 2, 3 };
v.resize(5);
cout << v.size() << "\n";   /* Output：5 */
```

*Worse Case if New Size is Greater than Old Capacity

Guaranteed Minimum Capacity

Time Complexity：Worse Case O(N)

```
vector<int> v = { 1, 2, 3 };
v.reserve(10);
cout << v.capacity();   /* Output：At least 10 */
```

*Worse Case if New Capacity is Greater than Old Capacity

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

Modify Data

Add Elements to the End of Vector

Time Complexity : O(1)

```
#include <vector>
vector<int> v = { 2, 5, 6, 8 };
v.push_back(4);   /* New : { 2, 5, 6, 8, 4 } */
```

Remove Elements from the End of Vector

Time Complexity : O(1)

```
#include <vector>
vector<int> v = { 2, 4, 8, 6 };
v.pop_back();   /* New : { 2, 4, 8 } */
```

Clear All Elements in Vector

Time Complexity : O(N)

```
vector<int> v = { 1, 2, 3 };
v.clear();   /* New : { } */
cout << v.size() << "\n";   /* Output : 0 */
```

*Clears All Elements in Vector, But Doesn't Change the Capacity

Use .erase() to Remove Elements

Time Complexity : O(N)

```
vector<int> v = { 1, 2, 3, 4, 5 };
v.erase(v.begin() + 1);   /* New : { 1, 3, 4, 5 }*/
v.erase(v.begin(), v.end() - 1);   /* New : { 5 }*/
```

*Think : How to Achieve O(1) Data Deletion ?

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

Sort Data

*By Default, std::sort() Arranges the Elements in Ascending Order

Sorting with C++ std::sort()

Time Complexity : O(NlogN)

```
#include <algorithm>
vector<int> v = { 2, 5, 1, 8 };
sort(v.begin(), v.end());   /* New : { 1, 2, 5, 8 } */
```

Sorting With Pre-Written Comparator

Time Complexity : O(NlogN)

```
vector<int> v = { 2, 4, 8, 6 };
sort(v.begin(), v.end(), greater<int>());
                            /* New : { 8, 6, 4, 2 } */
```

Reverse Using reverse() Function

Time Complexity : O(N)

```
vector<int> v = { 1, 2, 3 };
reverse(v.begin(), v.end());   /* New : { 3, 2, 1} */
```

Sorting With Custom Comparator

Time Complexity : O(NlogN)

```
vector<int> v = { 1, 2, 3 };
sort(v.begin(), v.end(), cmp);
      /* New : Depends on How You Define cmp */
```

*cmp is a Bool Function that You Need to Write By Yourself

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Vector

Iterator

Use rbegin() and rend() to Obtain Reverse Iterators

rbegin() Returns a Reverse Iterator that Points to the Last Element of the Vector

rend() Returns a Reverse Iterator that Points to the Position Before the First Element of the Vector

```
vector<int> v = { 1, 2, 3 };
for( auto rit = v.rbegin() ; rit != v.rend() ; ++rit )
   cout << *rit << " ";   /* Output : 3 2 1 */
```

Modifying Elements Using Iterators

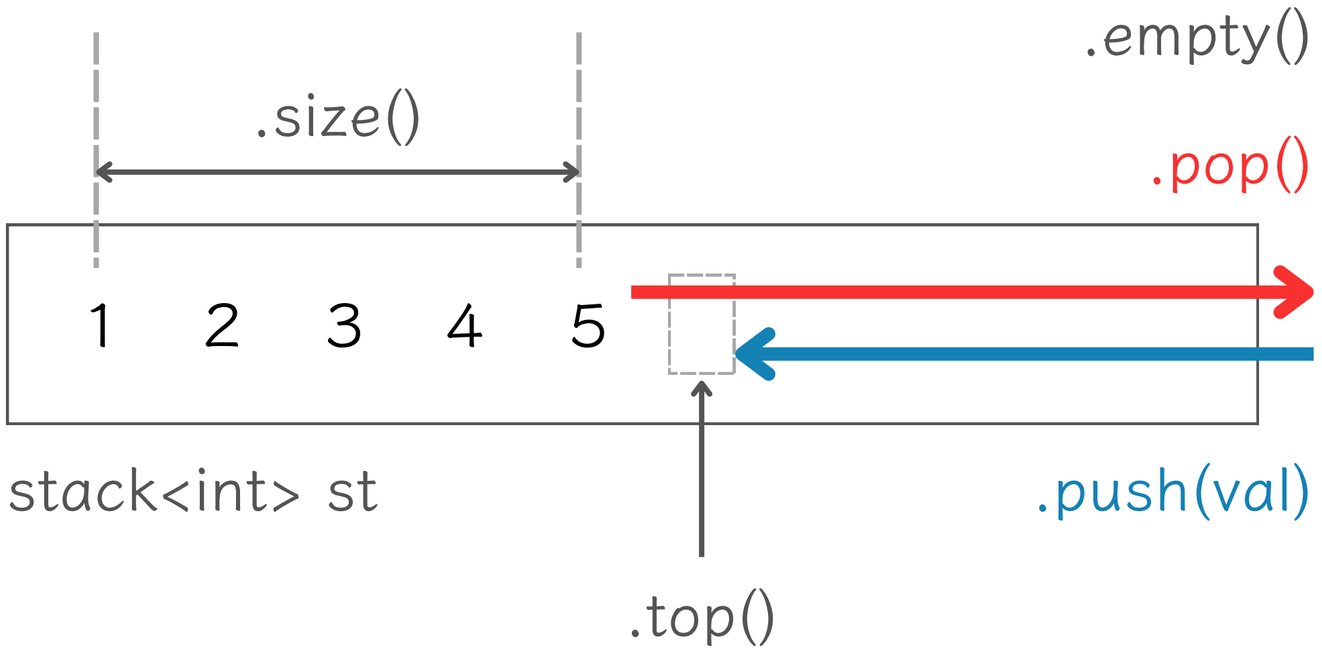
```
vector<int> v = { 1, 2, 3 };
auto it = v.begin(); *it = 4; // Change the First Element to 4
cout << v[0];   /* Output : 4 */
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Stack

LIFO



02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Stack

Data Operations

Add Elements at the Top

Time Complexity : O(1)

```
#include <stack>
stack<int> st;
st.push(10);   /* Now : { 10 } */
```

Remove the Top Element

Time Complexity : O(1)

```
stack<int> st;
st.push(48763);   /* Now : { 48763 } */
st.pop();   /* Now : { } */
```

Clear All Elements from the Stack

Time Complexity : O(1)

Although stack doesn't have .clear() function, you can use the .swap() method on an empty stack to effectively clear it.

```
#include <stack>
stack<int> st;
st.push(10);   /* Now : { 10 } */
st.push(20);   /* Now : { 10, 20 } */
st.push(30);   /* Now : { 10, 20, 30 } */
stack<int> emptyStack; // Create an empty Stack
st.swap(emptyStack);   /* Now : { } */
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Stack

Retrieve

Get the Top Element

Time Complexity : O(1)

```
stack<int> st;
st.push(10);
cout << st.top() << "\n";   /* Output : 10 */
```

Get the Number of Elements

Time Complexity : O(1)

```
stack<int> st;
st.push(30);
cout << st.size();   /* Output : 1 */
```

Check Whether It is Empty or Not

Time Complexity : O(1)

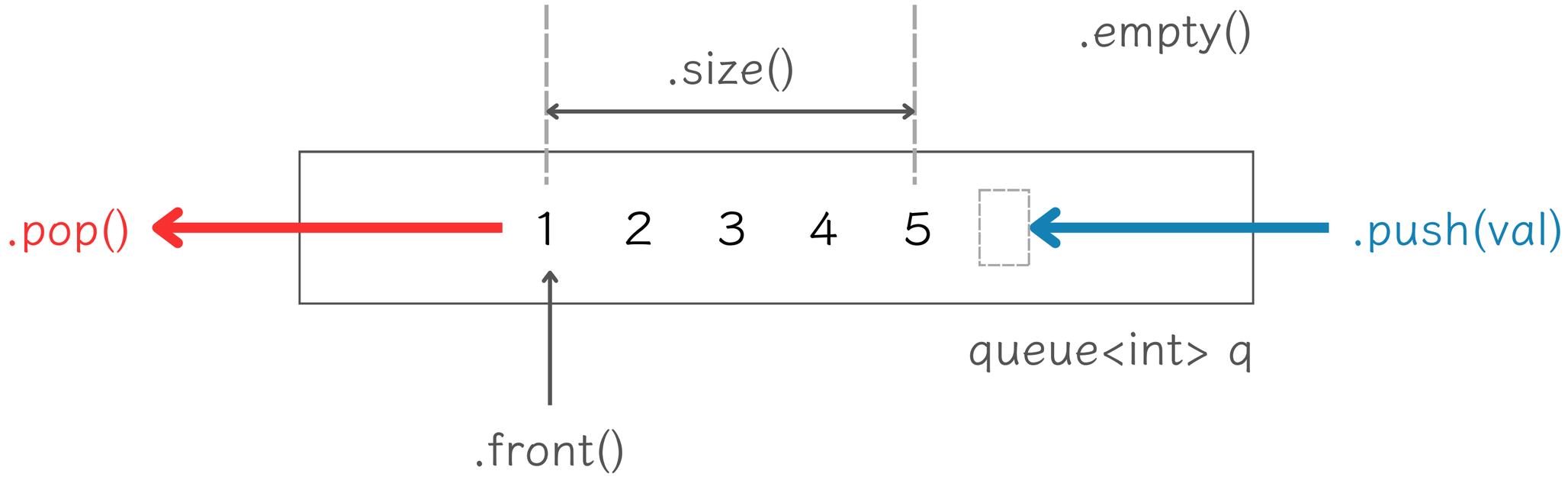
```
#include <stack>
stack<int> st;
cout << st.empty();   /* Output : 0 (False) */
      /* Stack Currently Contains No Elements */
st.push(10);
cout << st.empty();   /* Output : 1 (True) */
          /* Stack Currently Contains Elements */
st.pop();
cout << st.empty();   /* Output : 0 (False) */
      /* Stack Currently Contains No Elements */
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Queue

FIFO



02 Introductions to C++ STL

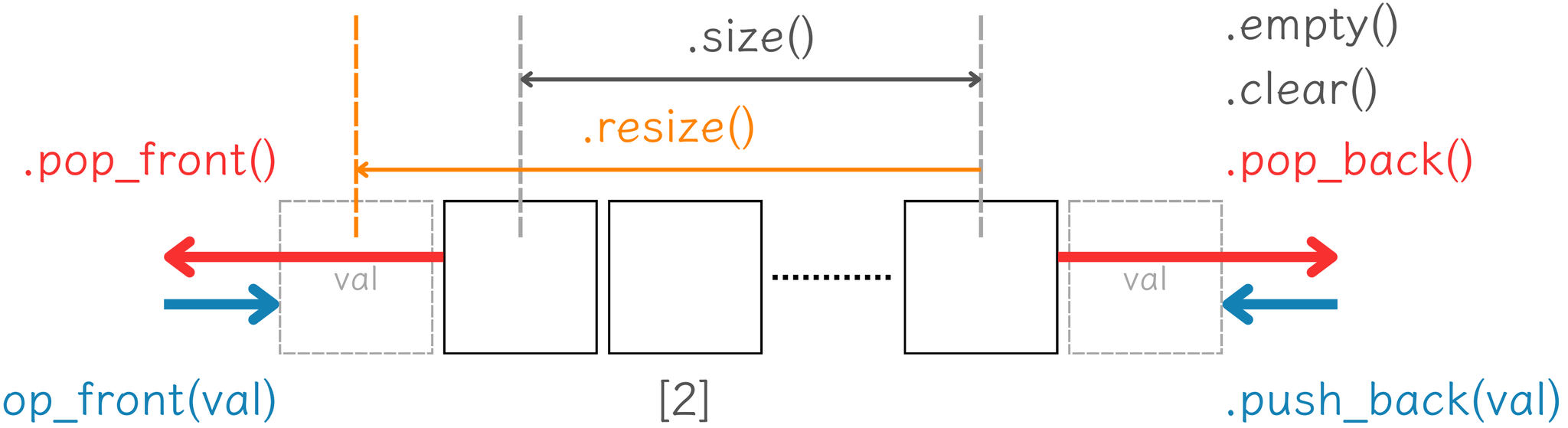
Tell You How to Use and Introduce Its Features

Deque

Double Ended Queue

Deque is a dynamically sized sequence container in C++ STL. Unlike queue, deque allows fast insertion and deletion of elements at both ends and supports direct access to individual elements through random access iterators.

A deque provides functionality and an interface similar to a vector. However, unlike a vector, a deque does not guarantee that all elements are stored in consecutive memory locations. Therefore, accessing an element in a deque by offsetting a pointer to another element may result in undefined behavior.



02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Deque

Initialization & Retrieval

Create an Empty Deque

Time Complexity : O(1)

```
#include <deque>
using namespace std;
deque<int> dq;
```

Add Elements at the Front

Time Complexity : O(1)

```
deque<int> dq = { 10, 20, 30 };
cout << dq.front();   /* Output : 10 */
```

Create a Deque from an Existing Array

Time Complexity : O(N)

```
vector<int> v = { 4, 8, 7, 6 };
deque<int> dq(v.begin(), v.end());
                        /* Now : { 4, 8, 7, 6 }*/
```

Add Elements at the End

Time Complexity : O(1)

```
deque<int> dq = { 10, 20, 30 };
cout << dq.back();   /* Output : 30 */
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Deque

Capacity

Check Whether It is Empty or Not

Time Complexity : O(1)

```
deque<int> dq;
cout << dq.empty();   /* Output : 0 (False) */
```

Get the Number of Elements

Time Complexity : O(1)

```
deque<int> dq = { 10, 20, 30 };
cout << dq.size();   /* Output : 3 */
```

Change Size

Time Complexity : O(N)

```
deque<int> dq = { 10, 20, 30 };
dq.resize(5);
/* Resize to 5; Set New Elements to Zero */
/* New : { 10, 20, 30, 0, 0 } */
dq.resize(2);
/* Resize to 2; Remove Redundant Elements */
/* New : { 10, 20 } */
dq.resize(7, 9);
/* Resize to 7; Set New Elements to Zero */
/* New : { 10, 20, 9, 9, 9, 9, 9 } */
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

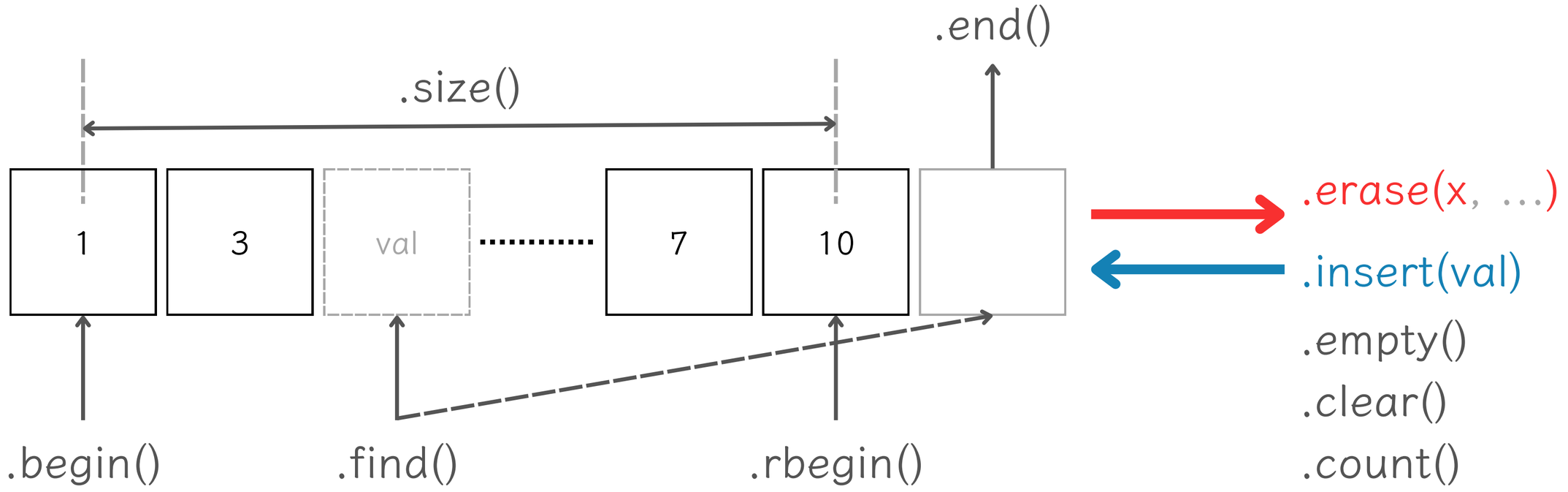
Set
Container Properties

Associative	Elements in associative containers are referenced by their key and not by their absolute position in the container
Ordered	The elements in the container follow a strict order at all times. All inserted elements are given a position in this order *Hint：The Elements in Set are Arranged from Small to Large by Default
Set	The value of an element is also the key used to identify it
Unique Keys	No two elements in the container can have equivalent keys

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set



02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set
Initialization

*As set is Ordered Container, It will Arrange the Elements of the Initializer List According to the Specified Order

Create an Empty Set

Time Complexity : O(NlogN)

```
#include <set>
using namespace std;
set<int> s = { 1, 9, 7 };   /* Now : { 1, 7, 9 } */
```

From Another std::set

Time Complexity : O(NlogN)

```
set<int> s1 = { 11, 9, 7, 45 };
set<int> s2 = s1;
```

From Another STL Container or Array

Time Complexity : O(NlogN)

```
set<int> s1(arr, arr + n);
set<int> s2(v.begin(), v.end());
```

One by One Initialization

Time Complexity : O(NlogN)

```
set<int> s;
s.insert(11);   /* Now : { 11 } */
s.insert(9);   /* Now : { 9, 11 } */
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Element Processing

Insert An Element Into Set

Time Complexity：O(logN)

```
#include <set>
set<int> s = { 11, 1, 9 };
s.insert(5);   /* Now：{ 1, 5, 9, 11 } */
```

Use .erase() to Remove Elements

Time Complexity：O(logN)

```
set<int> s = { 11, 9, 7, 45 };
s.erase(9);   /* Now：{ 7, 11, 45 }*/
s.erase(s.begin());   /* Now：{ 11, 45 } */
```

Clear All Elements in Set

Time Complexity：O(N)

```
set<int> s = { 11, 9, 7, 45 };
s.clear();
```

Checks Whether Element Exists

Time Complexity：O(N)

```
set<int> s = { 11, 9, 7, 45 };
cout << s.count(9);   /* Output：1（True）*/
cout << s.count(10);   /* Output：0（False）*/
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Retrieval

*Using next() or prev() on a Set when the Destination is Out of Range May Result in Undefined Behavior

Get the First Element in a Set

Time Complexity : O(1)

```
set<int> s = { 11, 9, 7, 45 };
cout << *s.begin() << "\n";   /* Output : 11 */
```

Get the Element Through next()

Time Complexity : O(D)

```
set<int> s = { 7, 9, 11, 45 };
cout << *next(s.begin());   /* Output : 9 */
cout << *next(s.begin(), 3);   /* Output : 45 */
```

Get the Last Element in a Set

Time Complexity : O(N)

```
set<int> s = { 11, 9, 7, 45 };
cout << *s.rbegin() << "\n";   /* Output : 11 */
```

Get the Element Through prev()

Time Complexity : O(D)

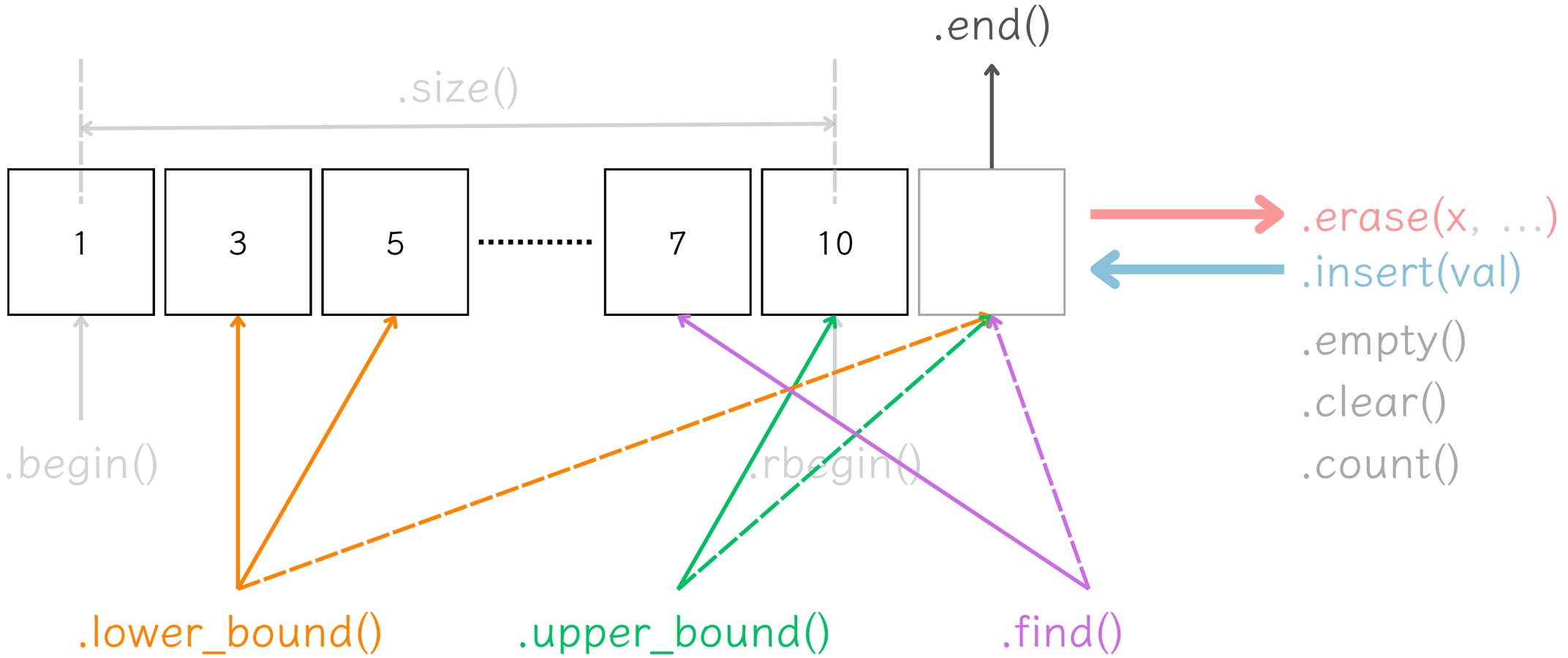
```
set<int> s = { 7, 9, 11, 45 };
cout << *prev(s.rbegin());   /* Output : 11 */
cout << *prev(s.rbegin(), 3);   /* Output : 7 */
```

02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching

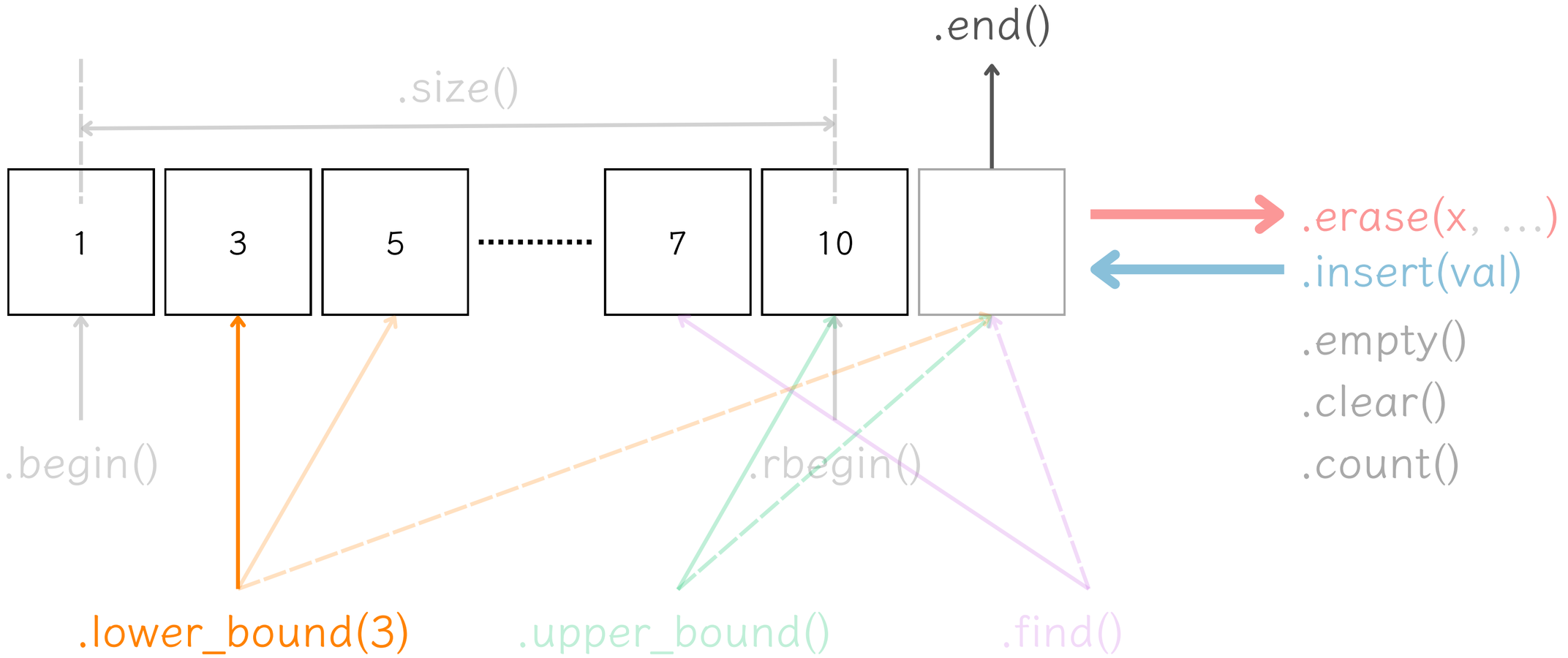


02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching

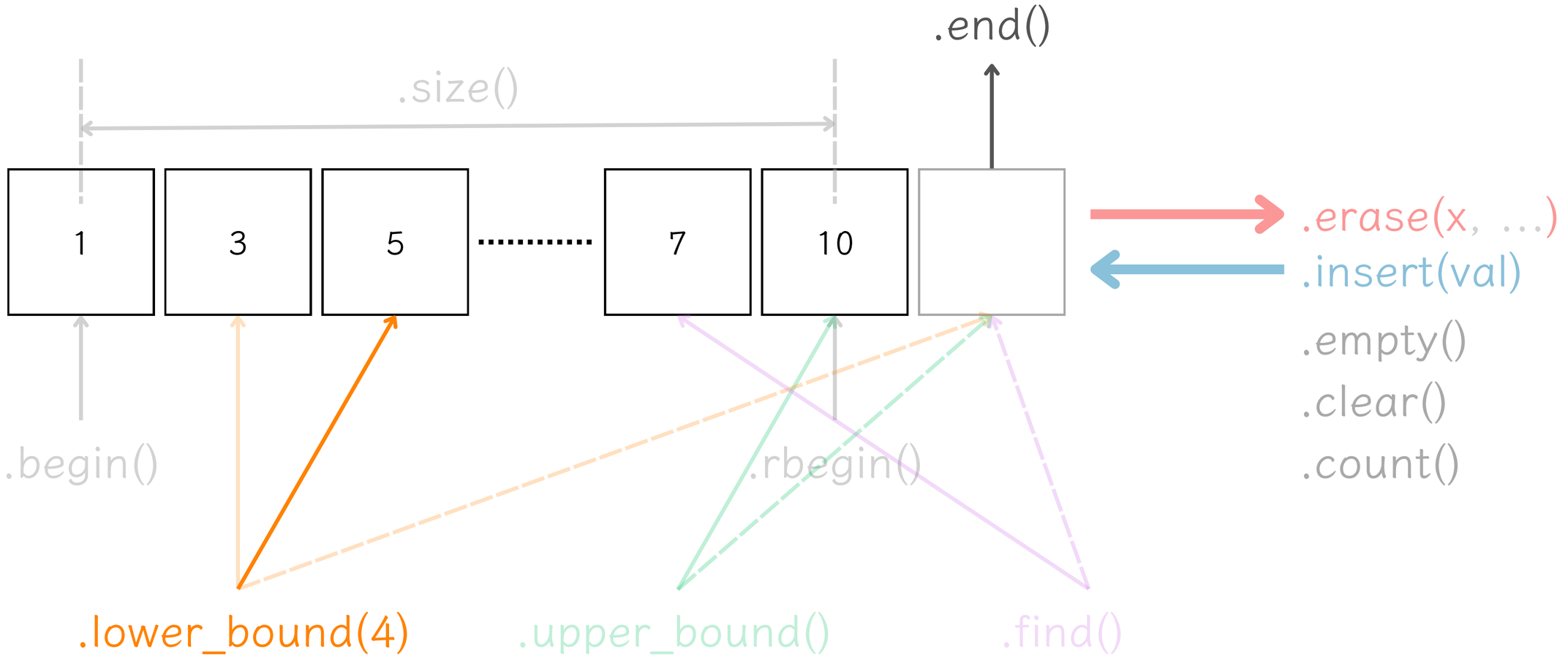


02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching

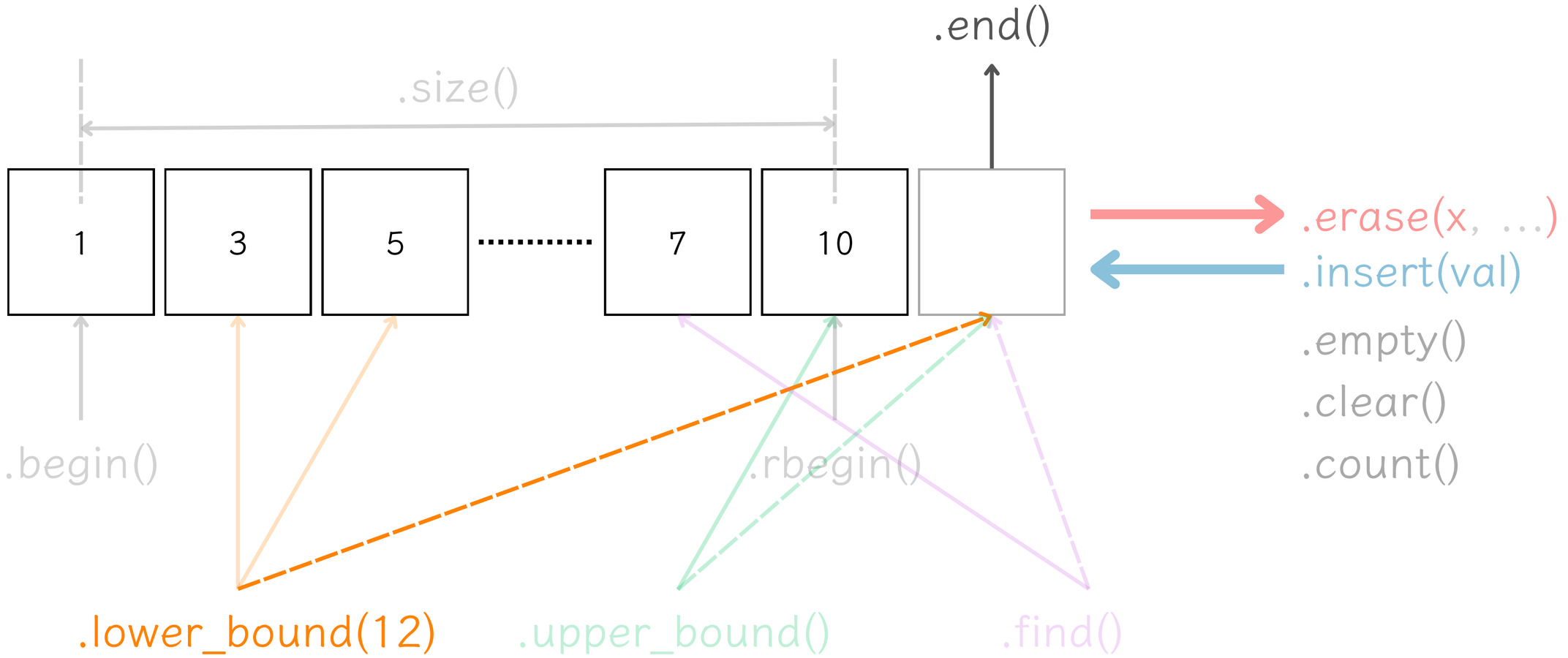


02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching

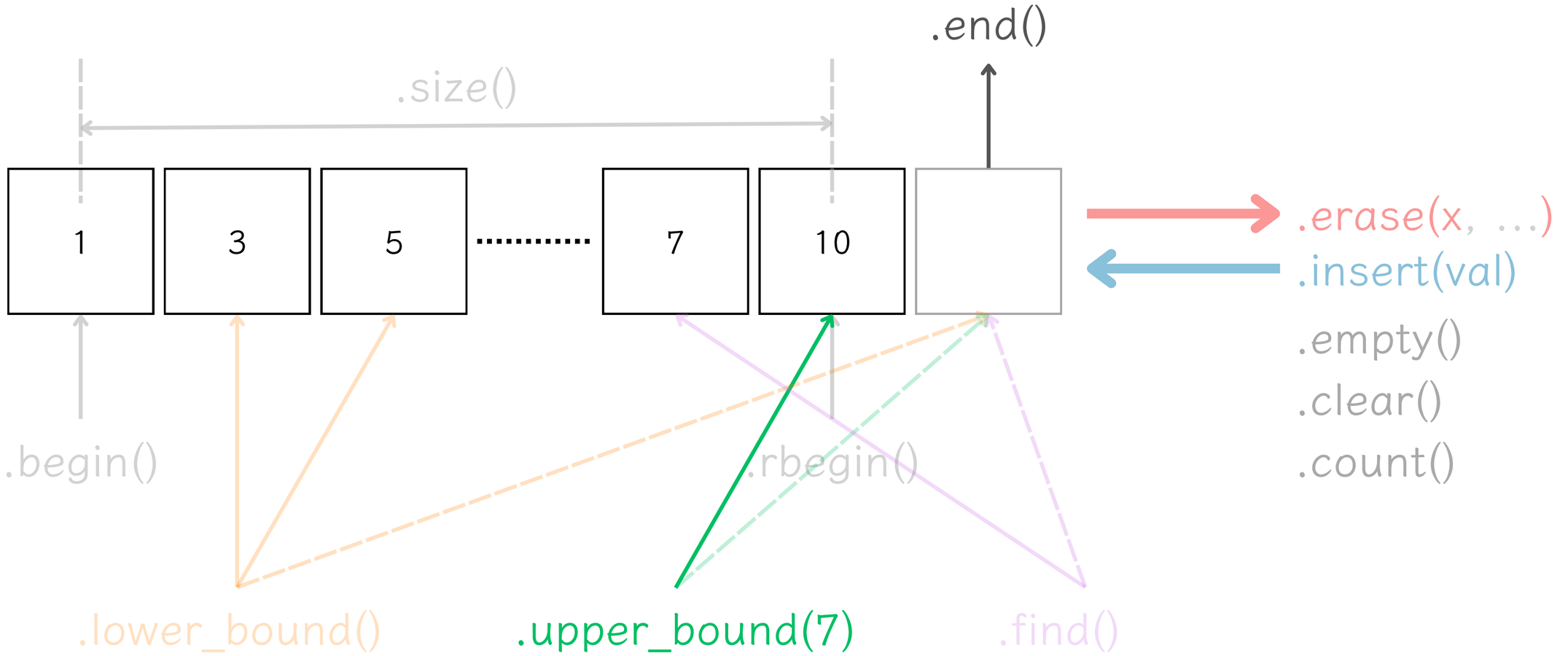


02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching

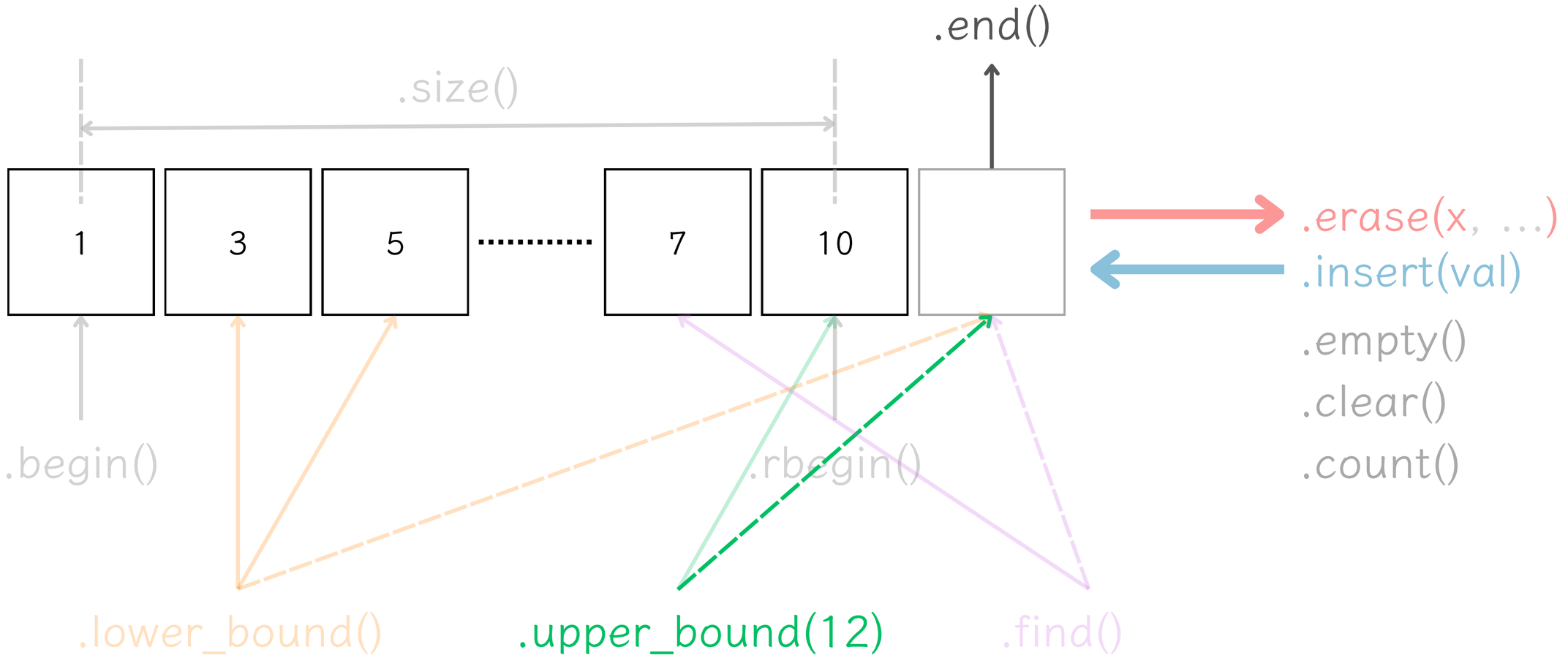


02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching

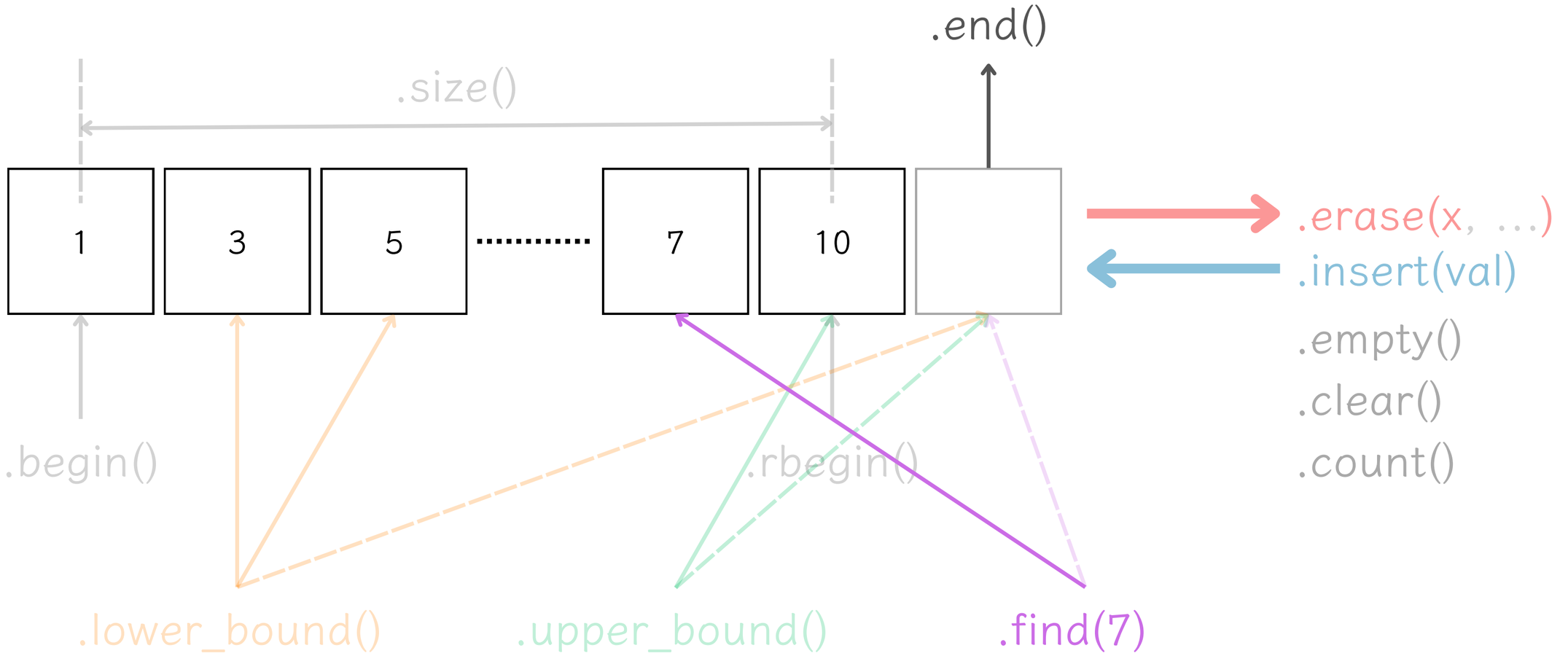


02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching

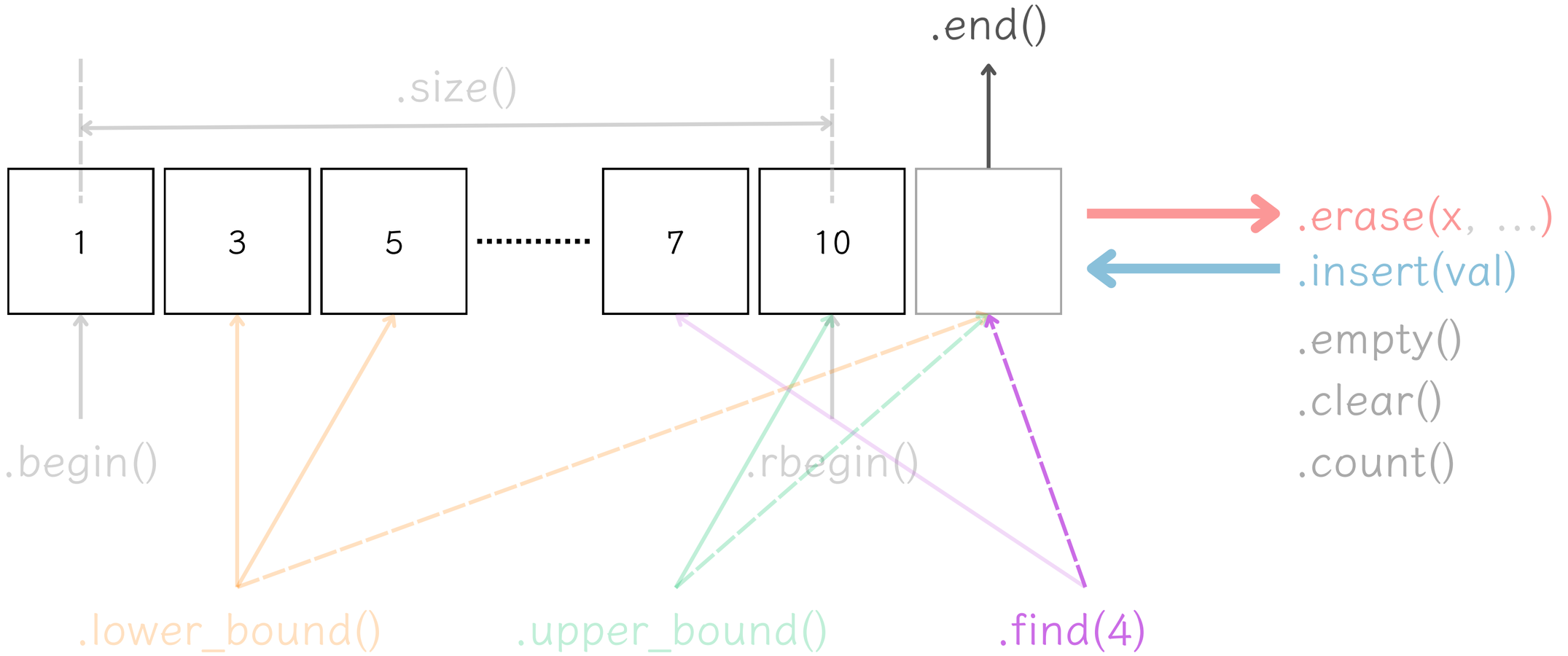


02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching



02 Introductions to C++ STL

Tell You How to Use and Introduce Its Features

Set

Searching

Using .lower_bound() in a Set

Time Complexity : O(logN)

```
set<int> s = { 2, 4, 6, 8, 9, 10 };

/* 1. Smaller Than All the Elements */
cout << *s.lower_bound(1);   /* Output : 2 */
/* 2. Exactly in the Set */
cout << *s.lower_bound(2);   /* Output : 2 */
/* 3. Between the Min. and Max. of Set */
cout << *s.lower_bound(3);   /* Output : 4 */
/* 4. Greater Than All The Elements */
cout << *s.lower_bound(12);   /* Output : ? */
```

Using .upper_bound() in a Set

Time Complexity : O(logN)

```
set<int> s = { 2, 4, 6, 8, 9, 10 };

/* 1. Smaller Than All the Elements */
cout << *s.upper_bound(1);   /* Output : 2 */
/* 2. Exactly in the Set */
cout << *s.upper_bound(2);   /* Output : 4 */
/* 3. Between the Min. and Max. of Set */
cout << *s.upper_bound(3);   /* Output : 4 */
/* 4. Greater Than All The Elements */
cout << *s.upper_bound(12);   /* Output : ? */
```