Inserting, Removing, and Searching Elements

Giovanni Dicanio
AUTHOR, SOFTWARE ENGINEER
https://blogs.msmvps.com/gdicanio

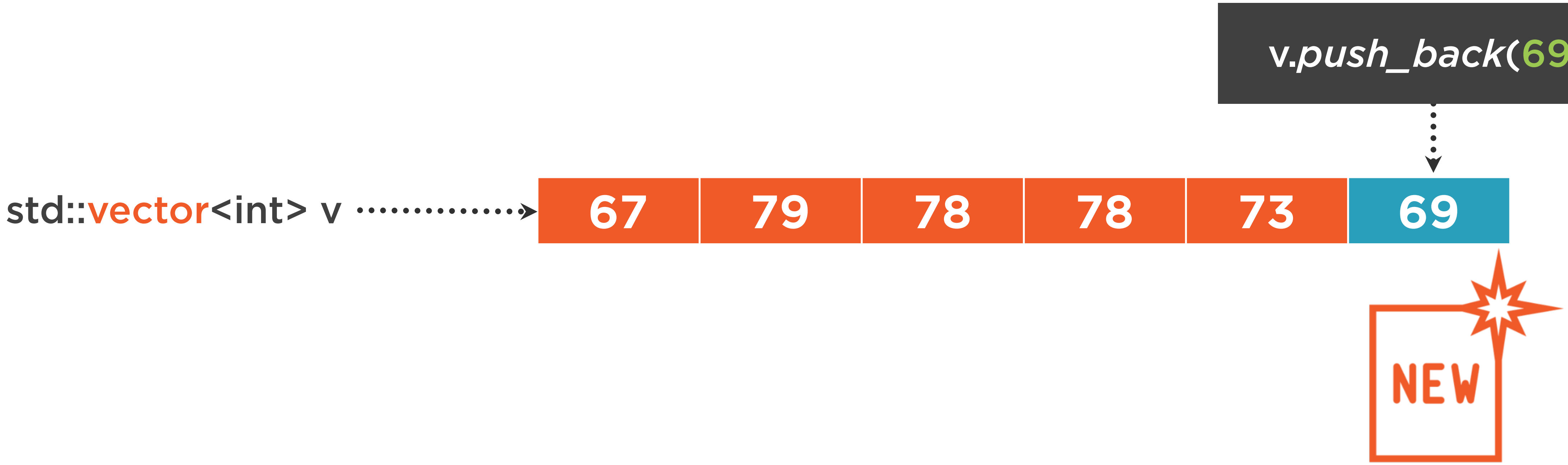
Overview

Inserting elements

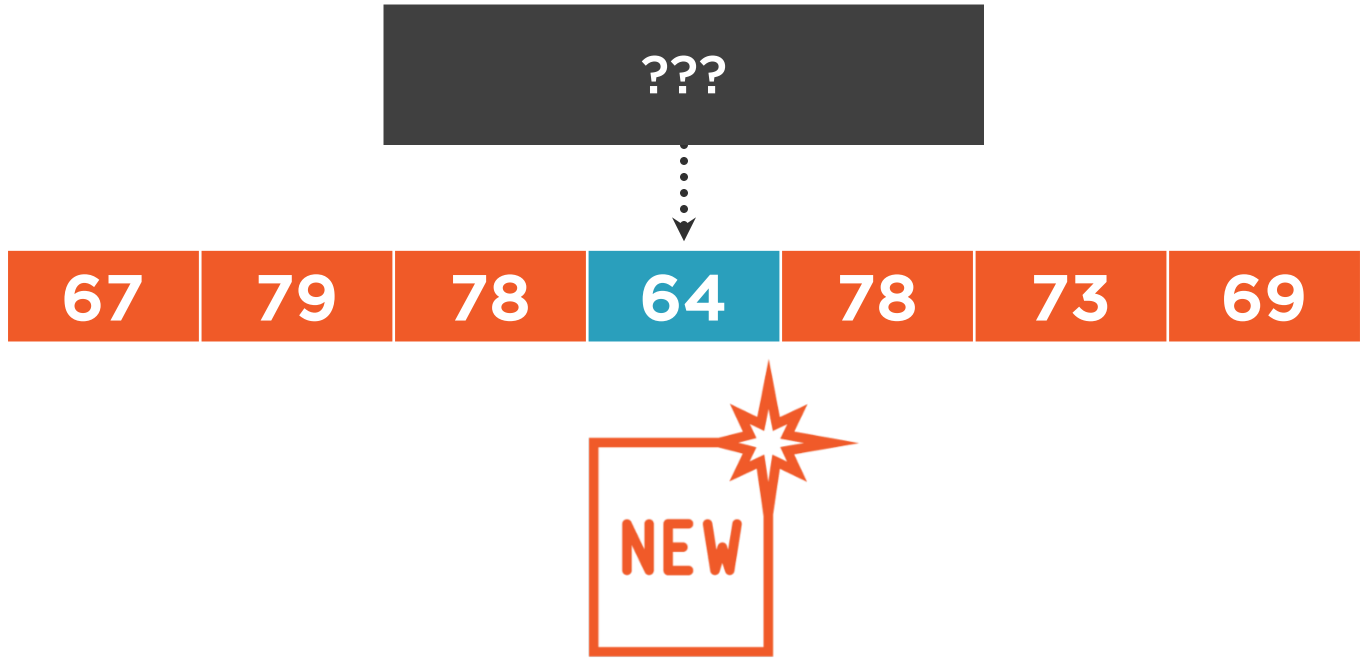
Removing elements (*erase-remove*)

Searching elements with std::*find*/*find_if*

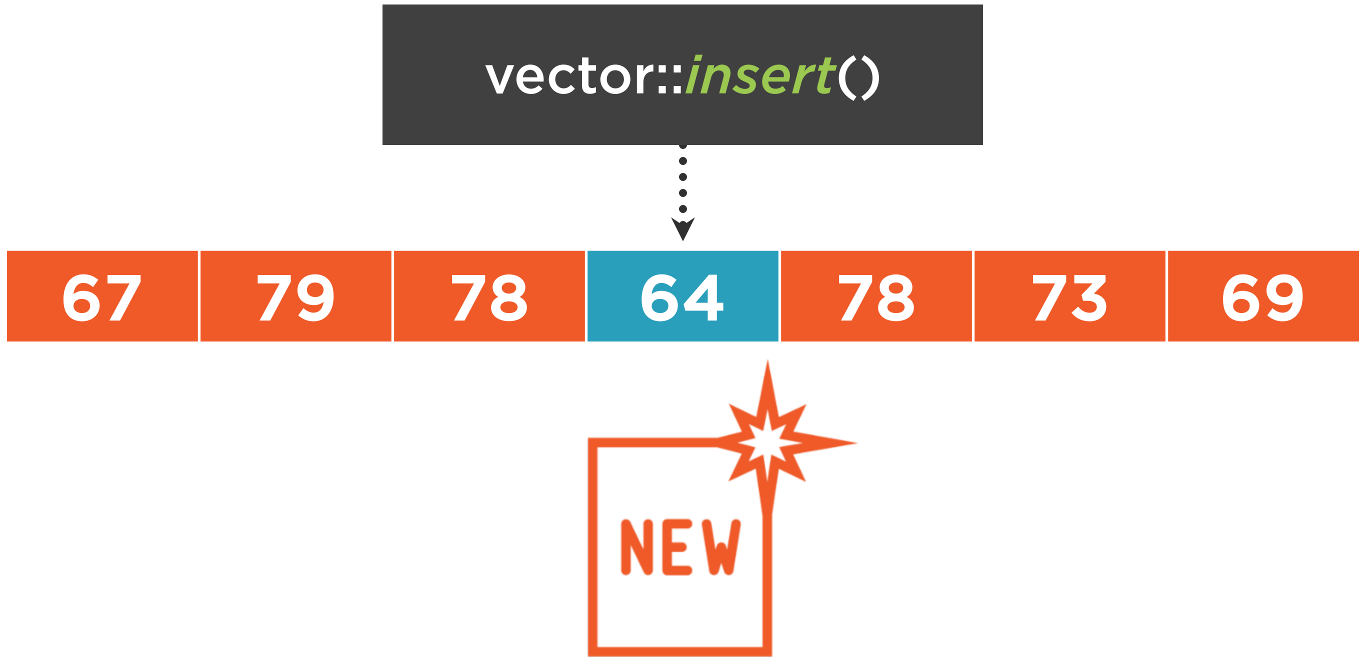
Appending Elements in std::vector



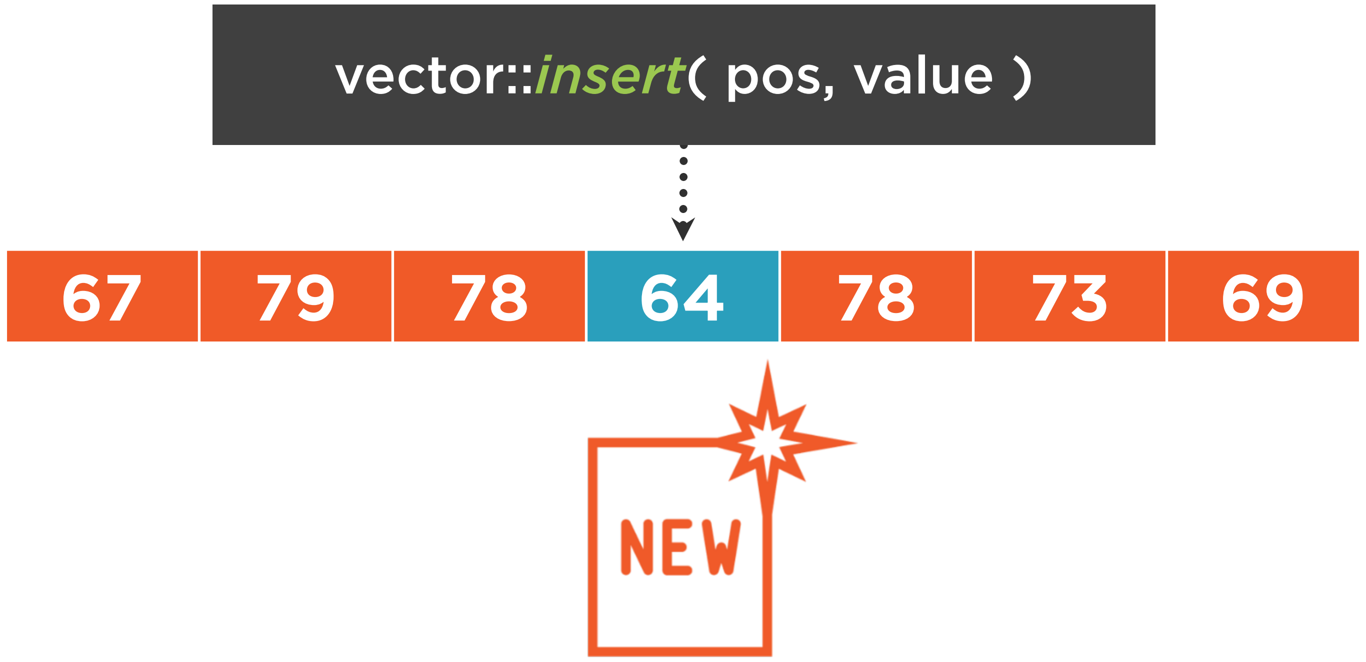
Inserting Elements in std::vector



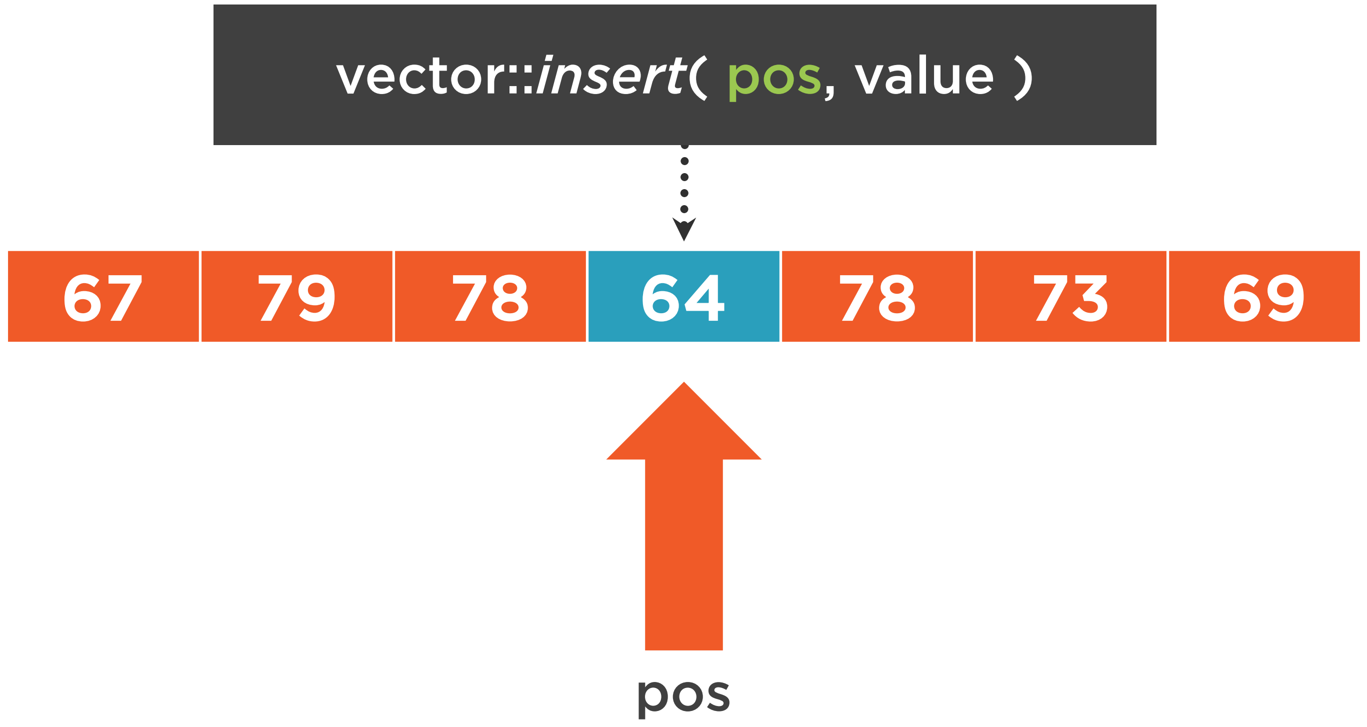
Inserting Elements in std::vector



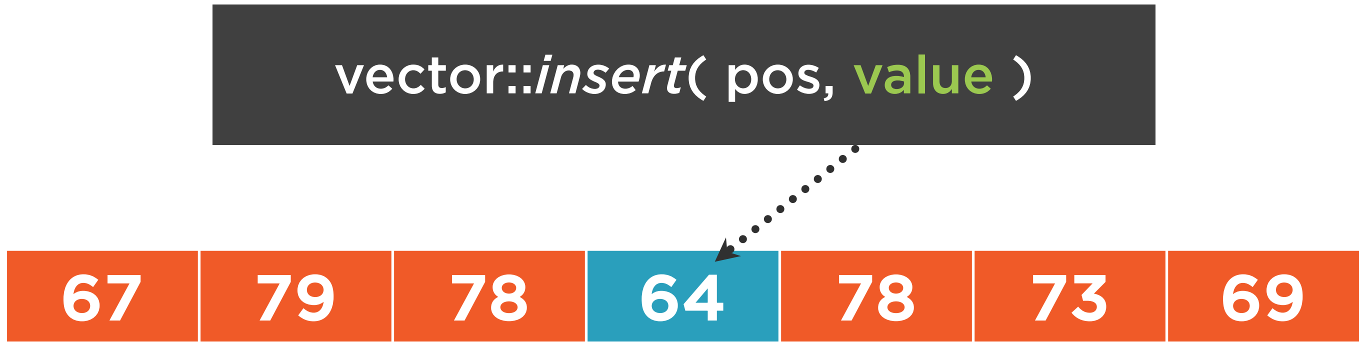
Inserting Elements in std::vector

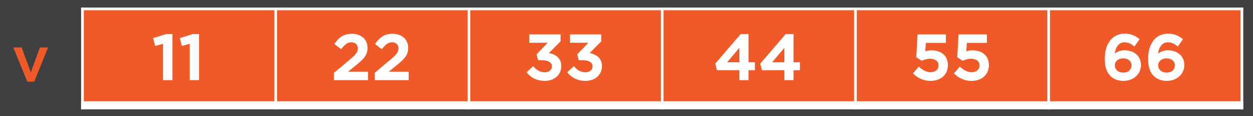


Inserting Elements in std::vector

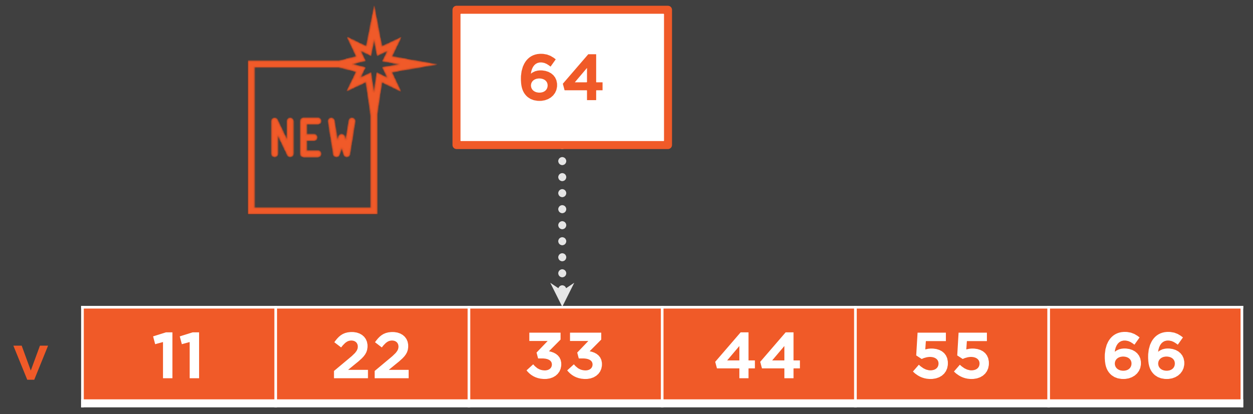


Inserting Elements in std::vector

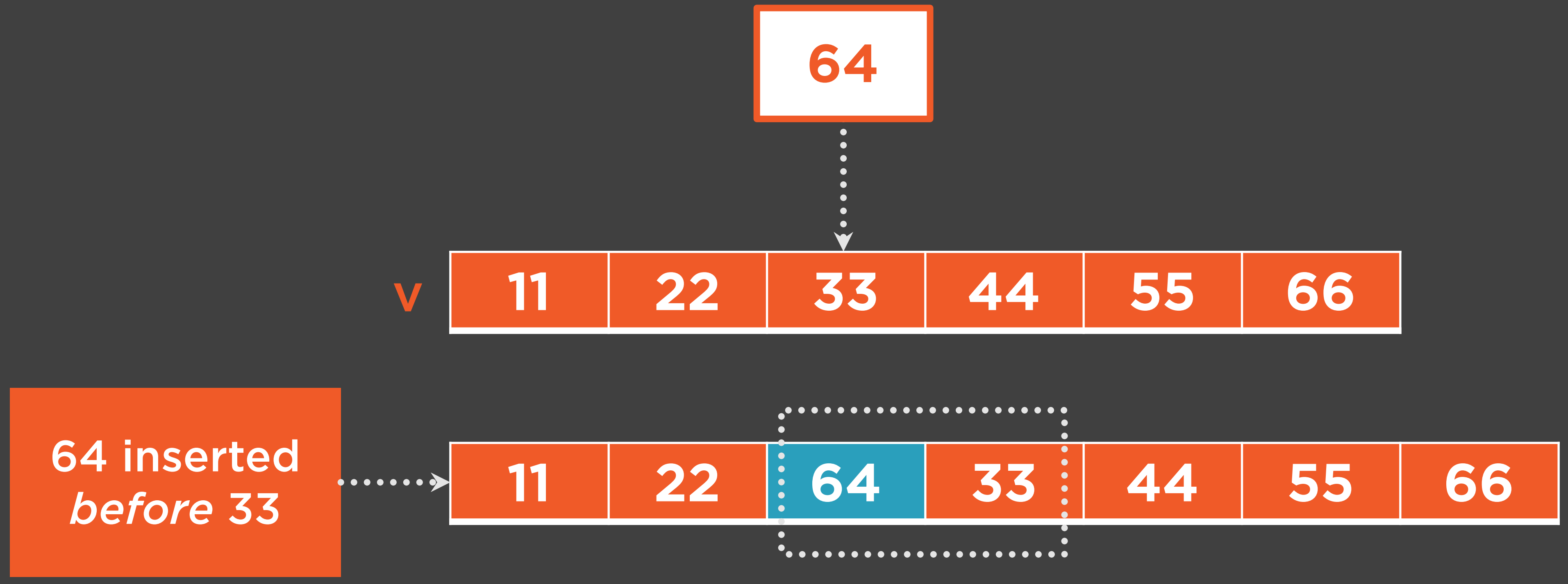




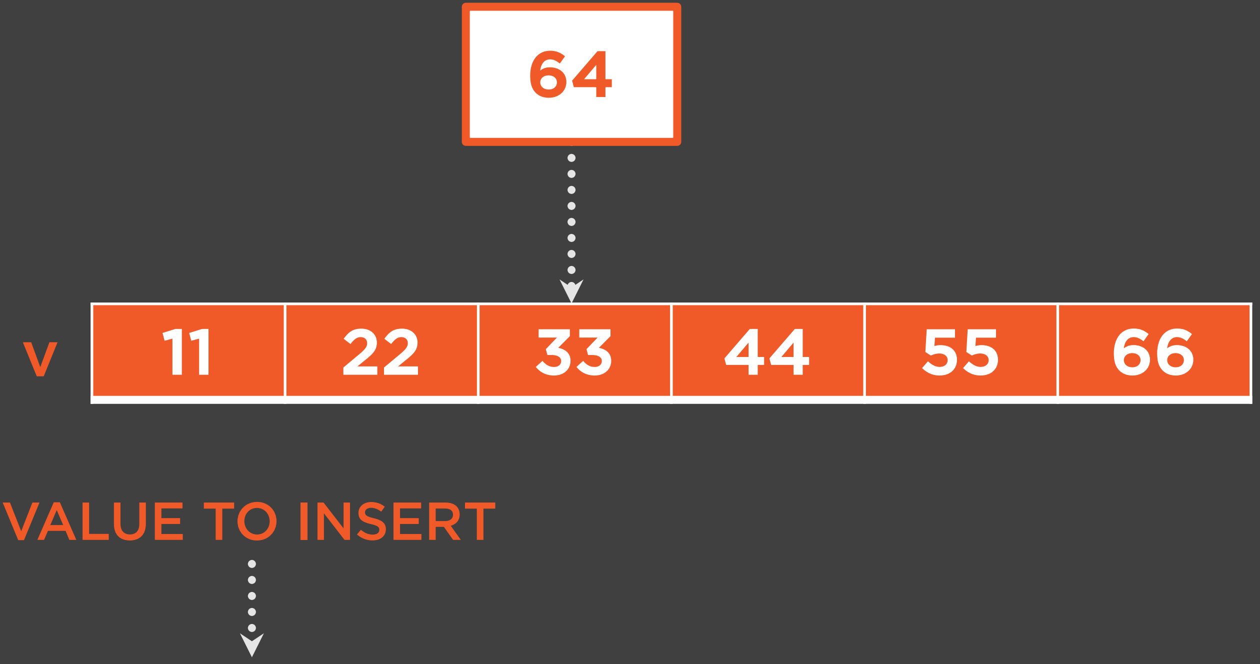
Inserting Elements with std::vector::insert



Inserting Elements with std::vector::**insert**

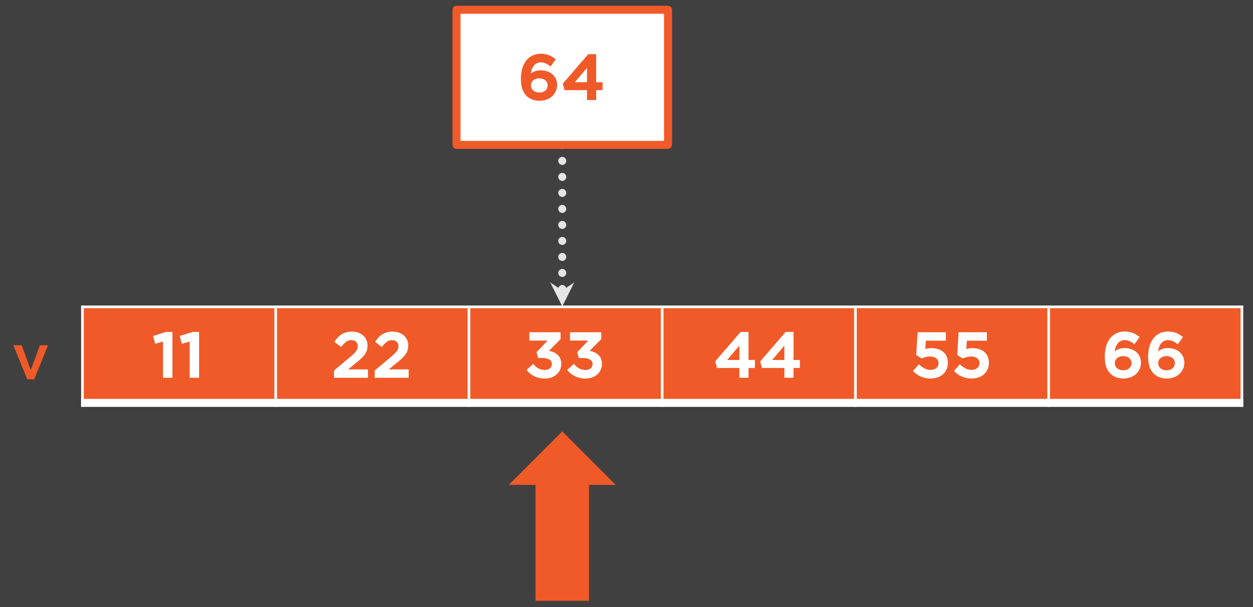


Inserting Elements with std::vector::**insert**



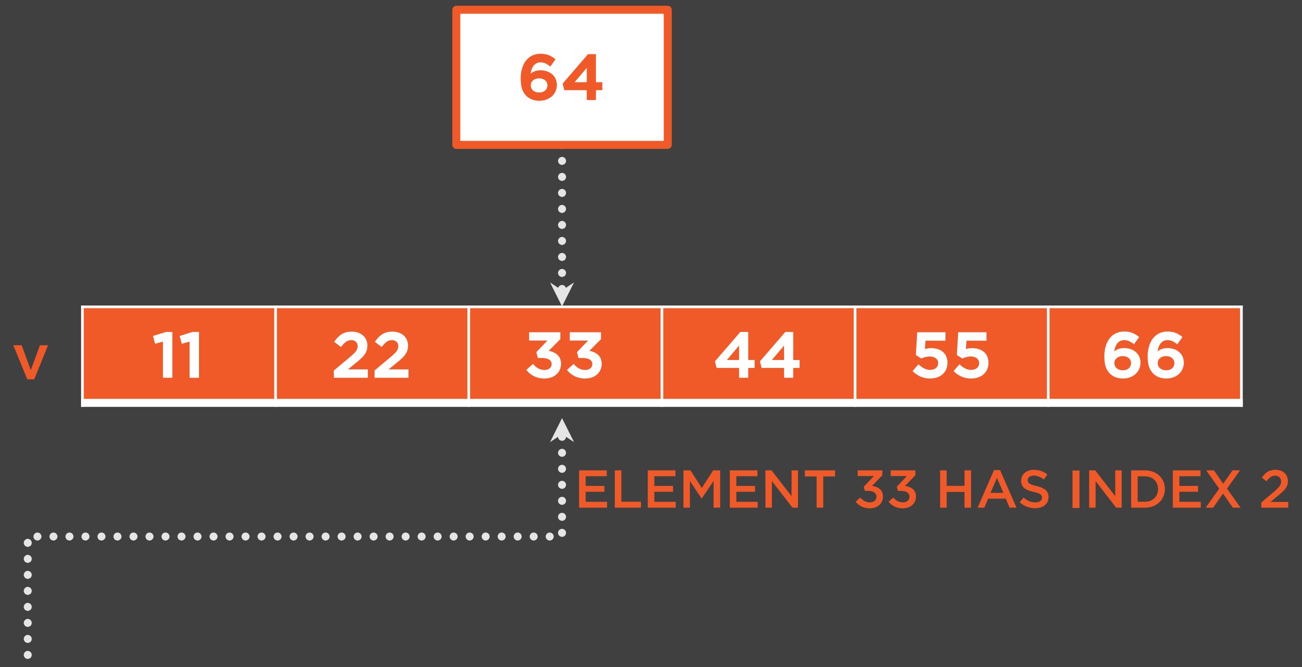
```
v.insert(     , 64 );
```

Inserting Elements with std::vector::insert



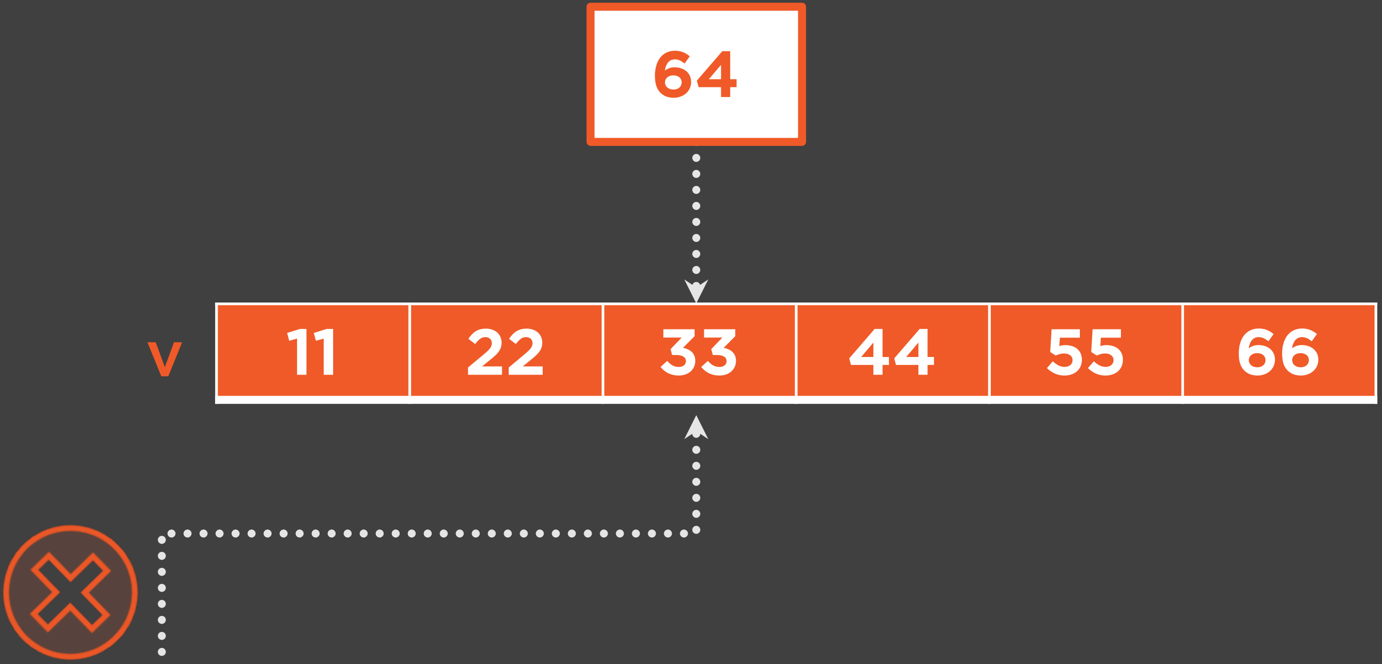
```
v.insert( ???, 64 );
```

Inserting Elements with std::vector::**insert**



```
v.insert( 2 /* 3rd element index */, 64 );
```

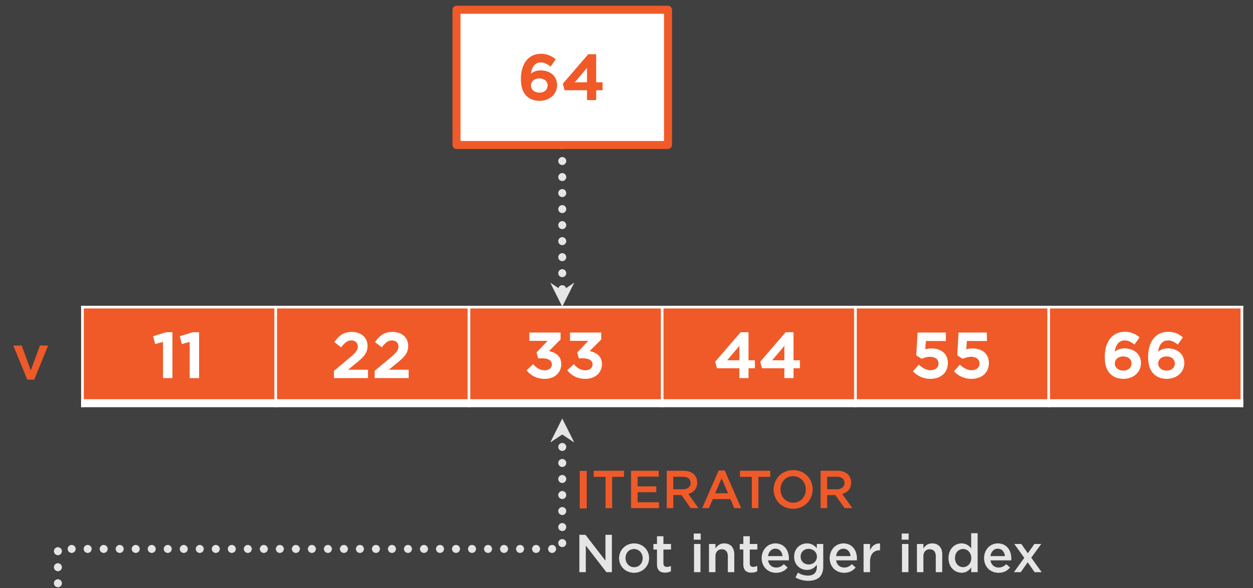
Inserting Elements with std::vector::insert



```
v.insert( 2 /* 3rd element index */, 64 );
```

Inserting Elements with std::vector::insert

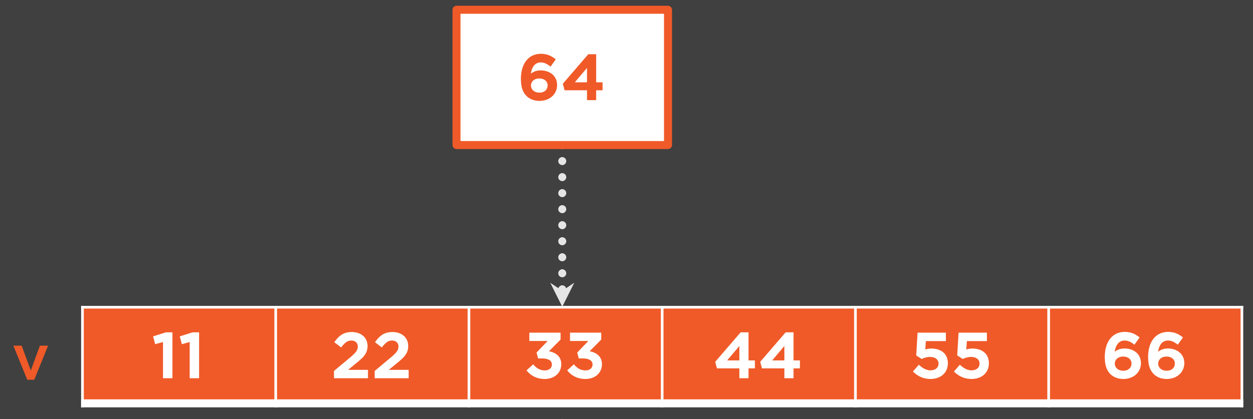
Insertion position is *not* an integer index



```
v.insert( pos /* iterator */, 64 );
```

Inserting Elements with std::vector::insert

Insertion position is an *iterator*

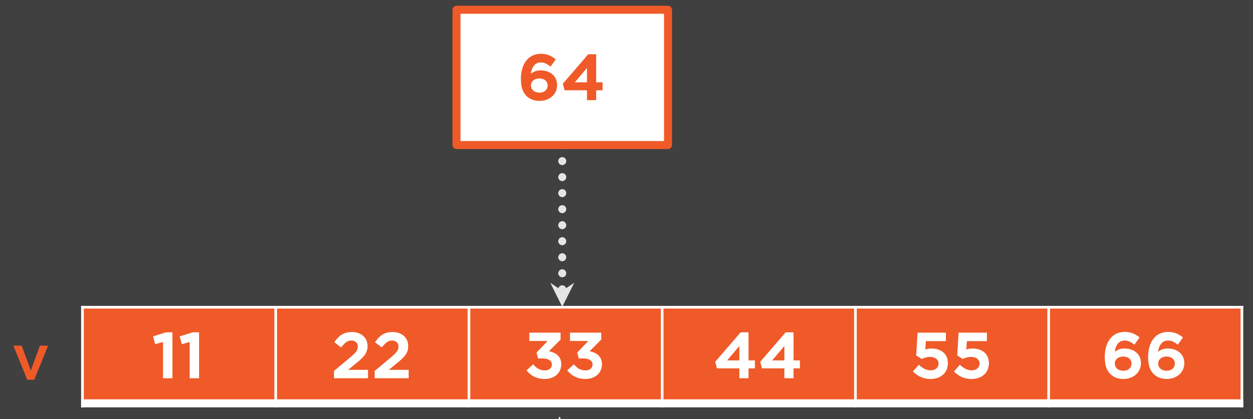


```
v.insert( pos /* iterator */, 64 );
```

Inserting Elements with std::vector::insert

vector::*insert*(pos, value)

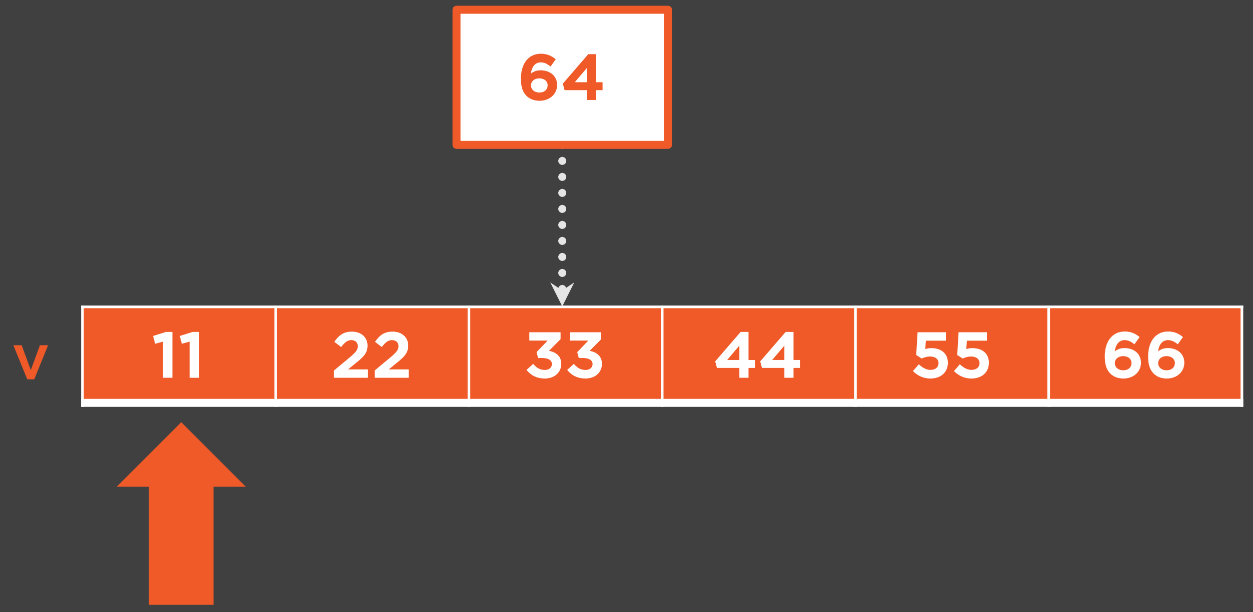
Inserts value *before* pos



```
v.insert( begin(v) + 2, 64 );
```

Inserting Elements with std::vector::insert

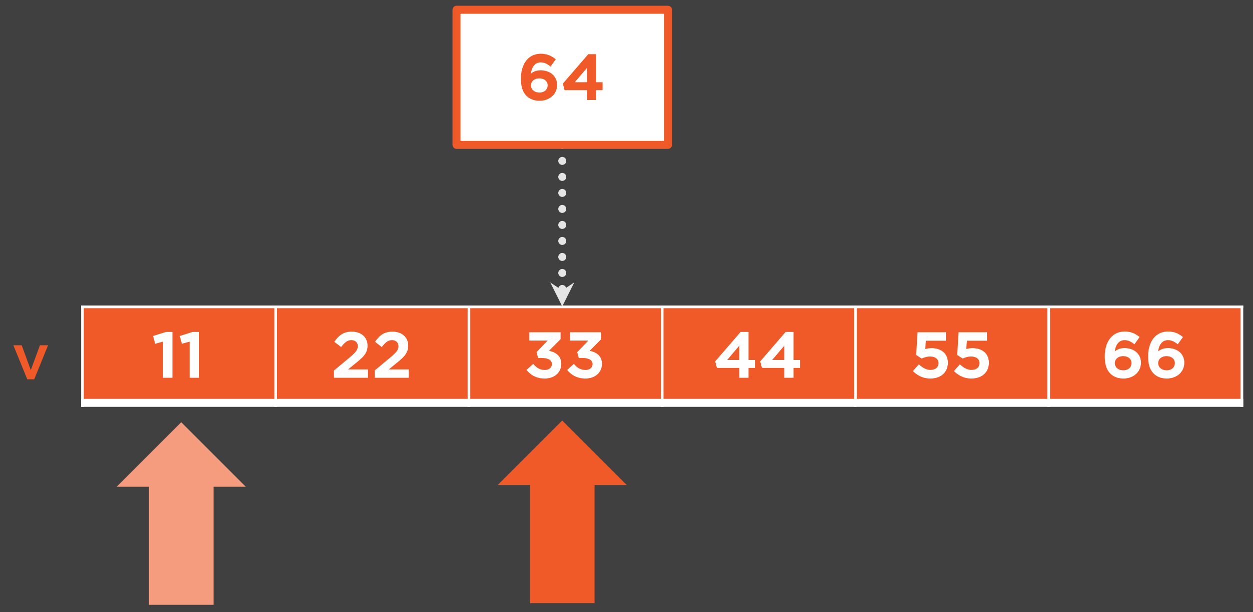
Insertion position *iterator* = *begin*(v) + 0-based index



```
v.insert( begin(v) + 2, 64 );
```

Inserting Elements with std::vector::**insert**

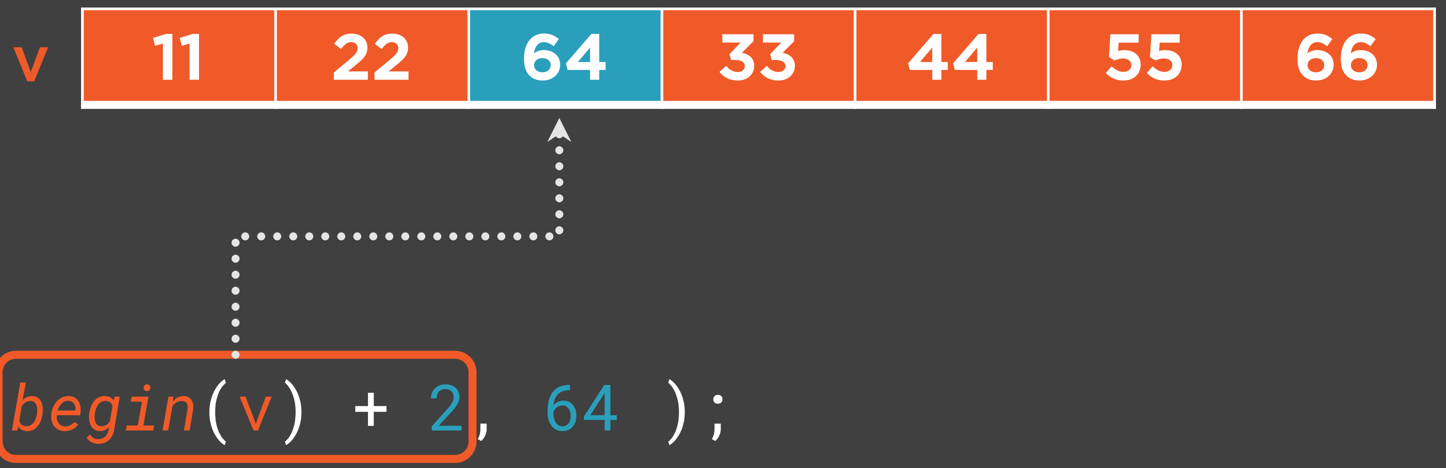
Insertion position *iterator* = *begin*(v) + 0-based index



```
v.insert( begin(v) + 2, 64 );
```

Inserting Elements with std::vector::insert

Insertion position *iterator* = *begin*(v) + 0-based index



```
v.insert( begin(v) + 2, 64 );
```

Inserting Elements with std::vector::**insert**

Insertion position *iterator* = *begin*(v) + 0-based index

MEMBER FUNCTION

vector::*begin*

```
v.insert( v.begin() + 2, 64 );
```

Inserting Elements with std::vector::**insert**

COPY COUNT

```
v.insert( pos, count, value );
```

Inserting Elements with std::vector::insert

Insert *count* copies of the *value* before *pos*

INITIALIZER LIST

```
v.insert( pos, {100, 200, 300} );
```

Inserting Elements with std::vector::**insert**

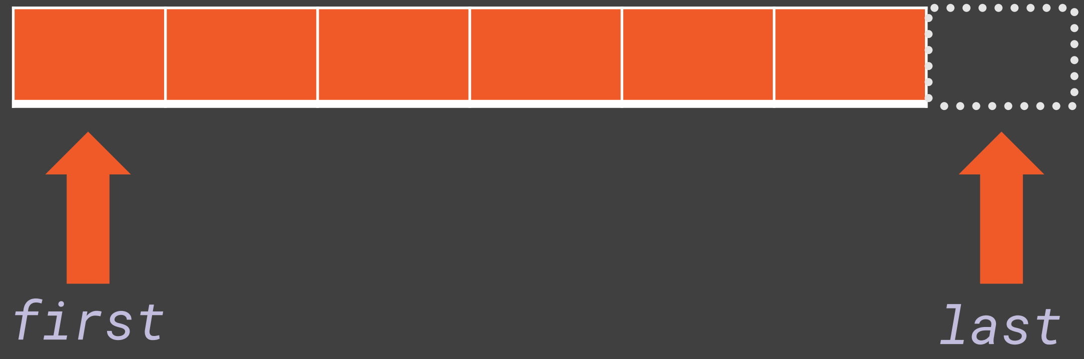
Insert elements from the *initializer list* before pos

SOURCE ELEMENT RANGE

```
v.insert( pos, first, last );
```

Inserting Elements with std::vector::**insert**

Insert elements from the source range [*first*, *last*)



SOURCE ELEMENT RANGE

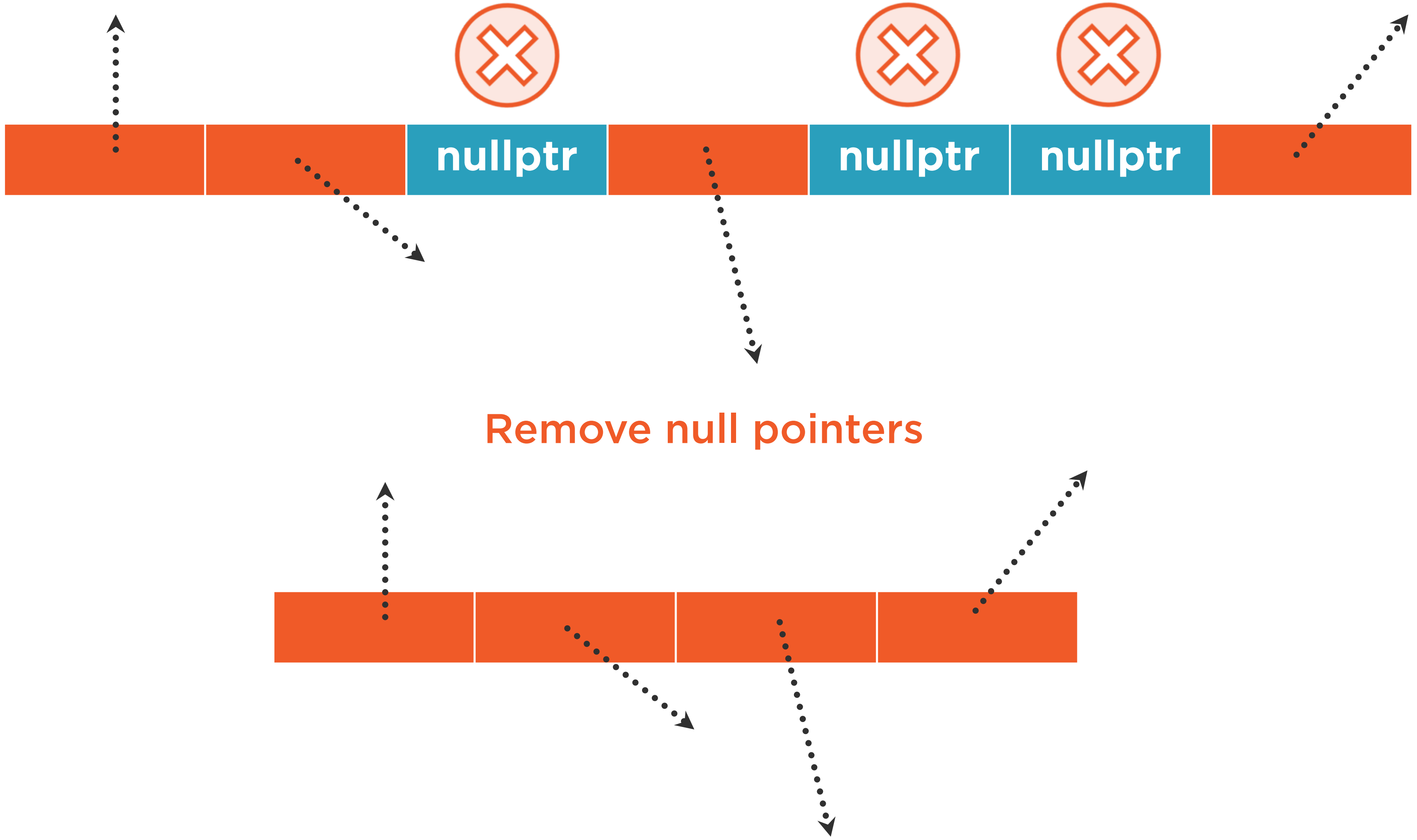
```
v.insert( pos, first, last );
```

Inserting Elements with std::vector::insert

Insert elements from the source range [*first*, *last*)

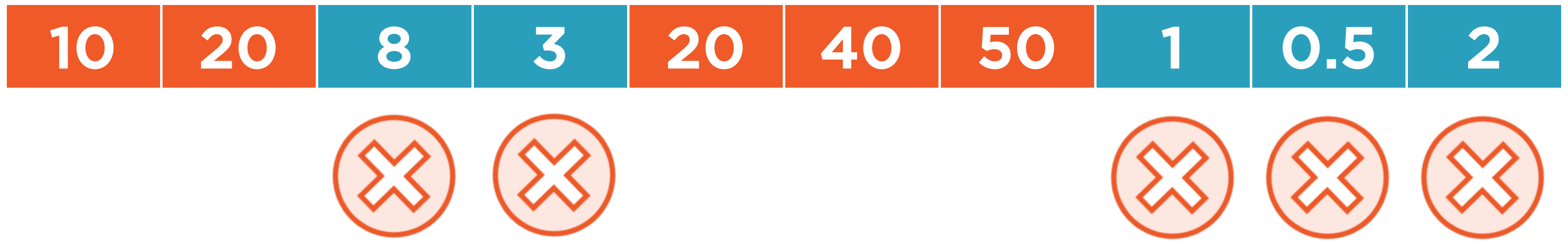
first is *included*
last is *excluded*

Removing Elements

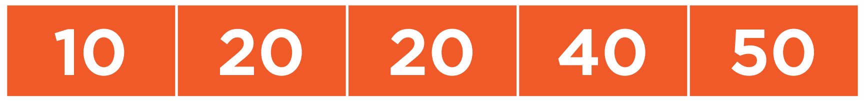


Remove null pointers

Removing Elements



Remove all values < 10



Removing Elements

Exact value

Condition

Removing Elements

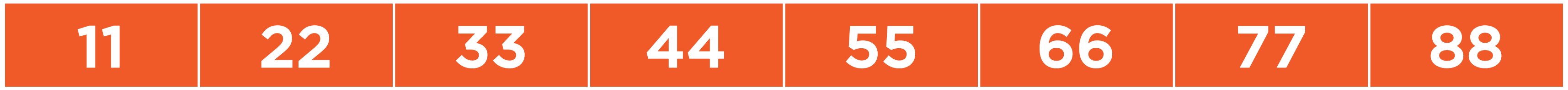
Writing explicit iteration code

Potentially inefficient and bug-prone

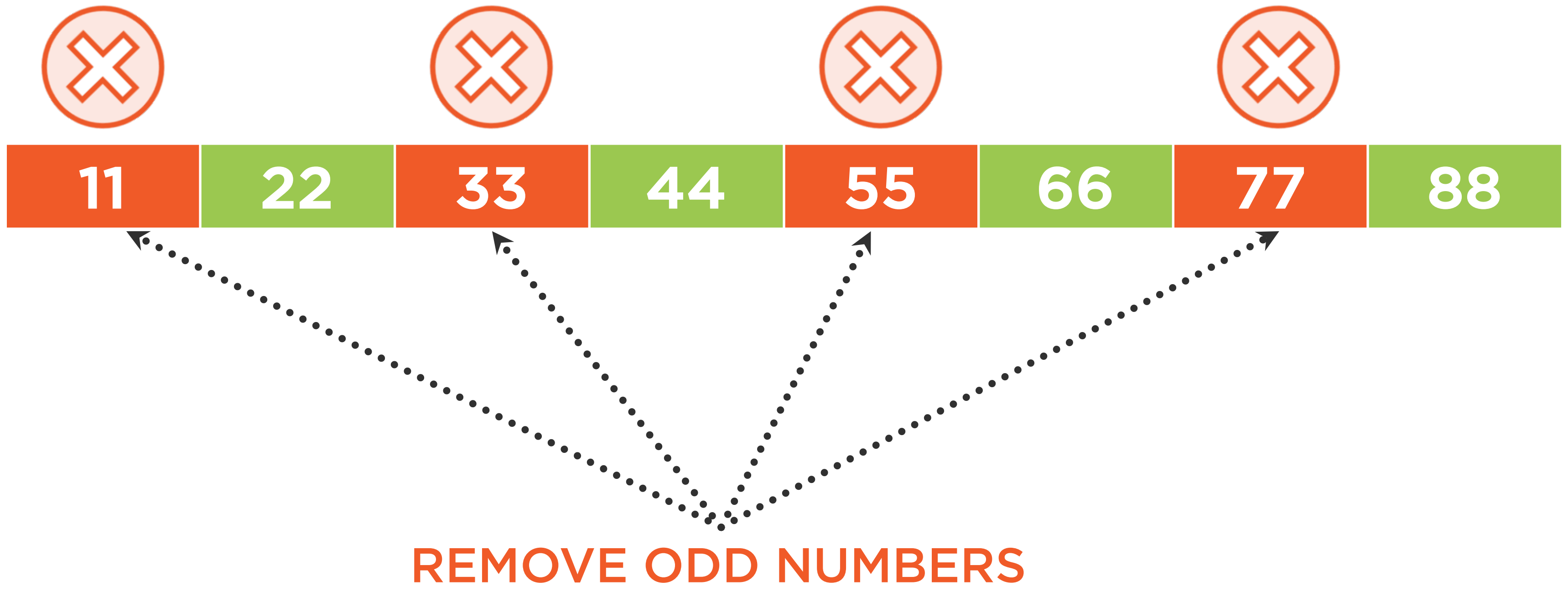
Removing Elements

Reuse code from the C++ Standard Library

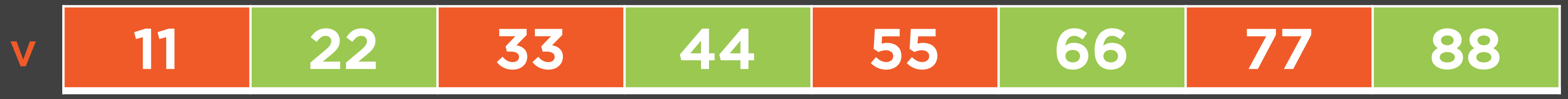
Example: Removing Odd Numbers from vector



Example: Removing Odd Numbers from vector

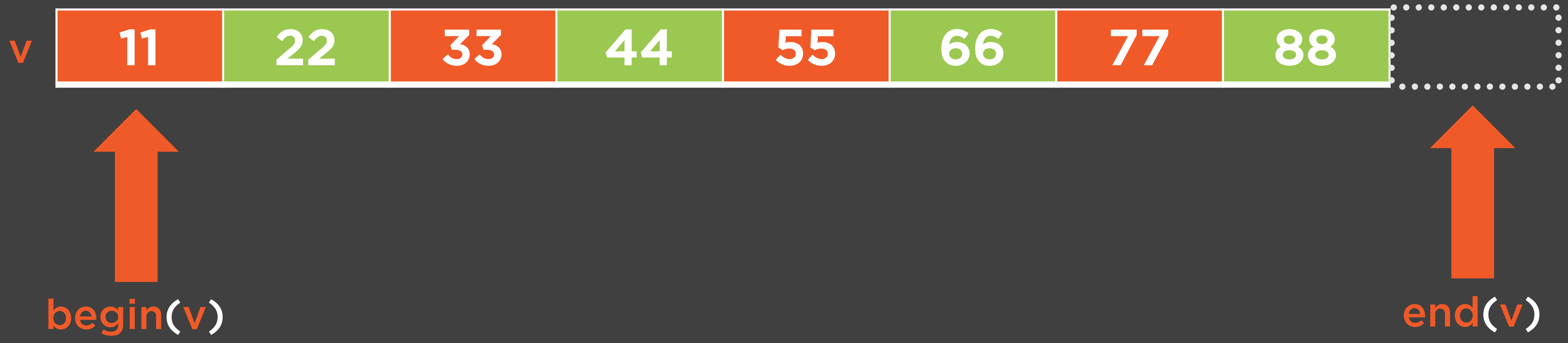


std::*remove_if*



```
std::remove_if(first, last, predicate)
```

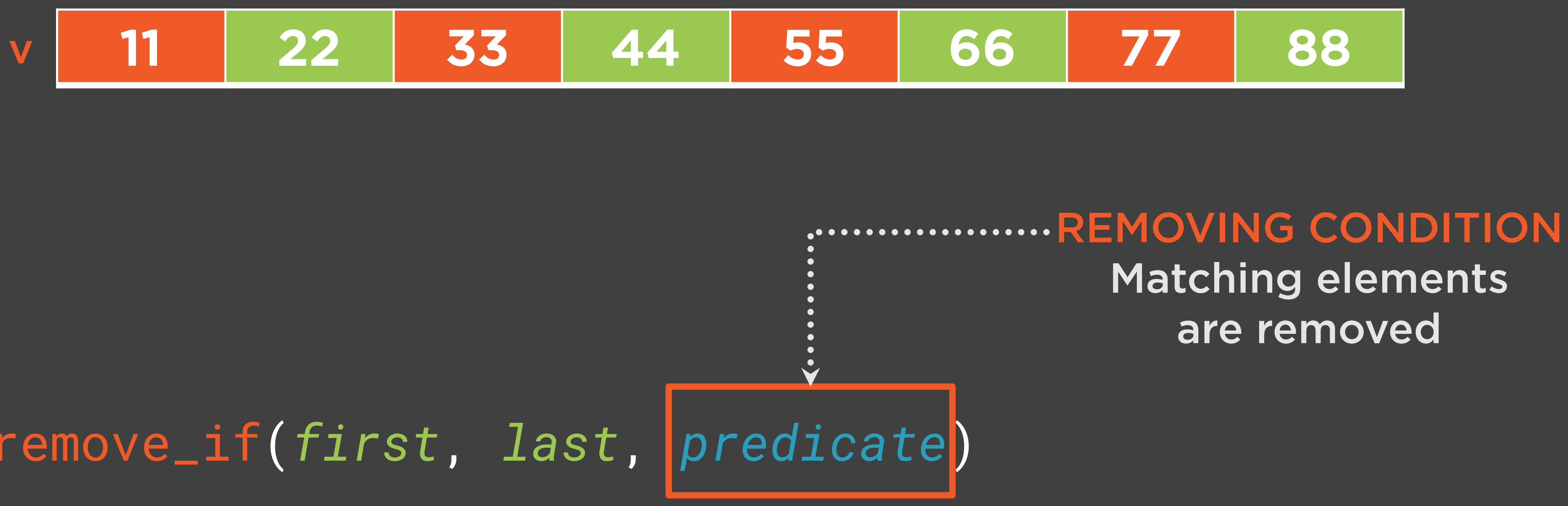
Removing Elements with std::remove_if



```
std::remove_if(first, last, predicate)
```

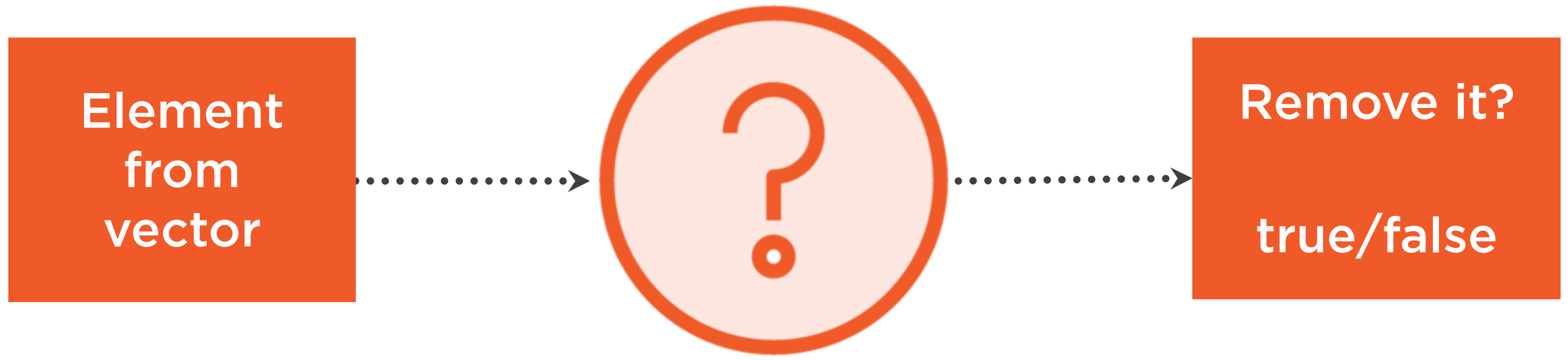
Removing Elements with std::remove_if

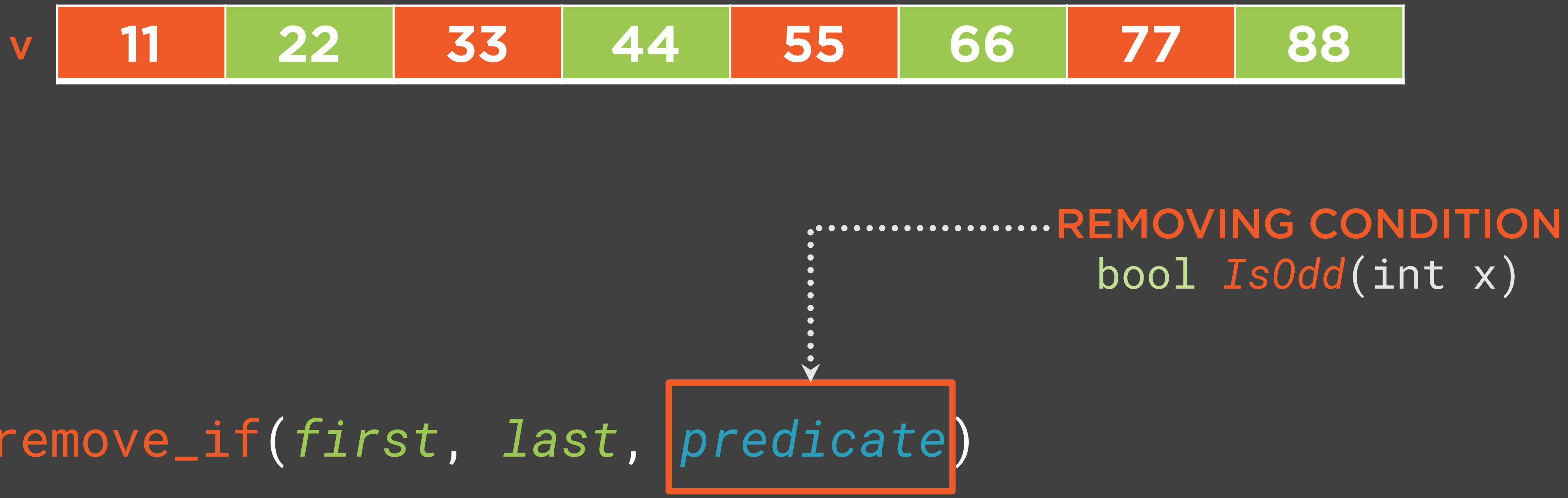
Process the whole vector content



```
std::remove_if(first, last, predicate)
```

Removing Elements with std::remove_if



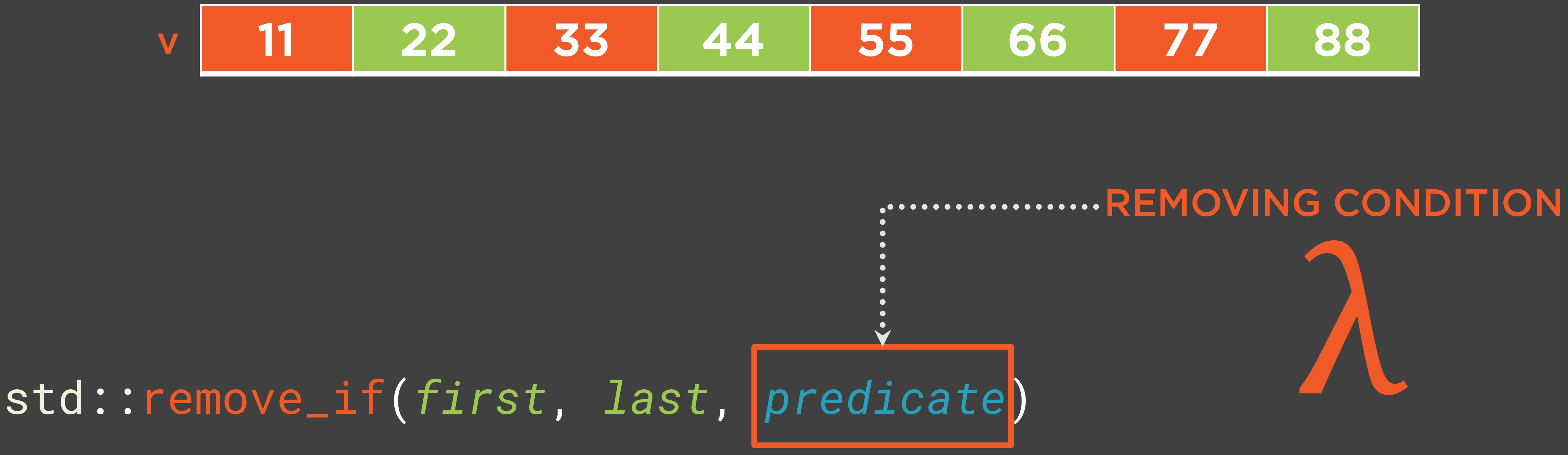


```
bool IsOdd(int x)
```

```
std::remove_if(first, last, predicate)
```

Removing Elements with std::remove_if

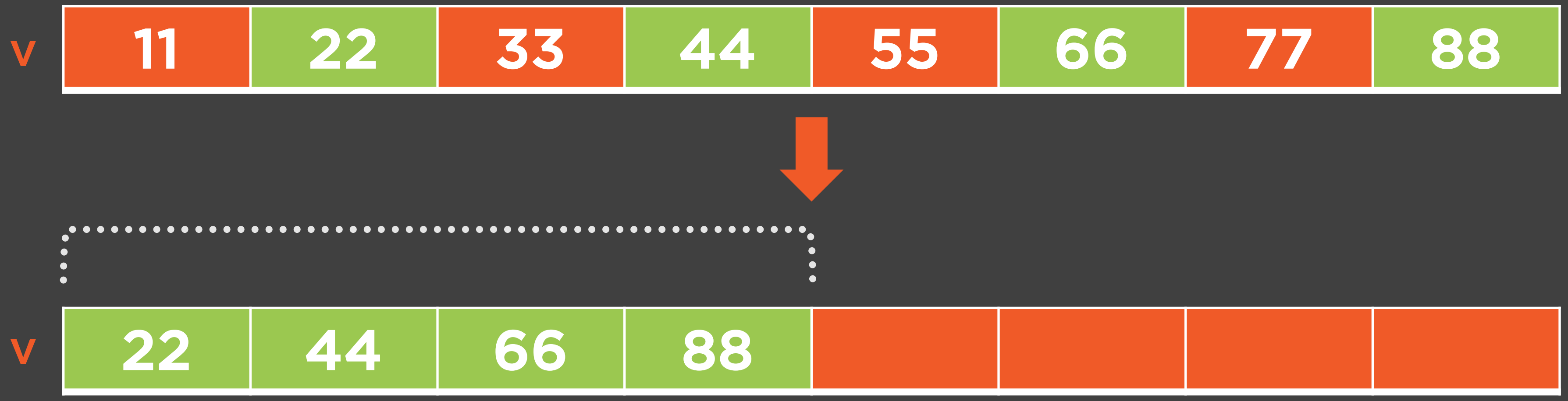
Remove odd numbers



```
std::remove_if(first, last, predicate)
```

Removing Elements with std::remove_if

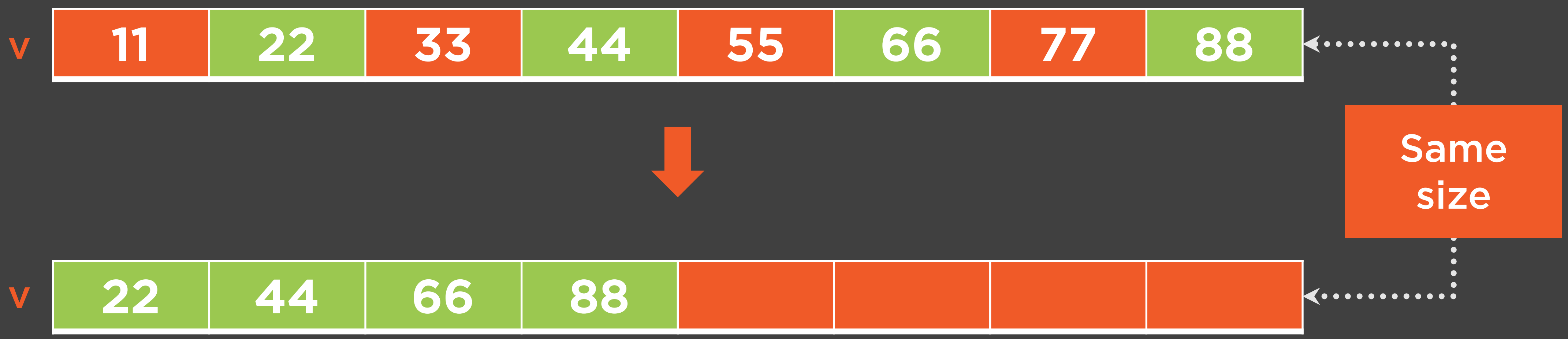
The removing condition can be written *locally* using a lambda



```
std::remove_if(begin(v), end(v), IsOdd)
```

Removing Elements with std::remove_if

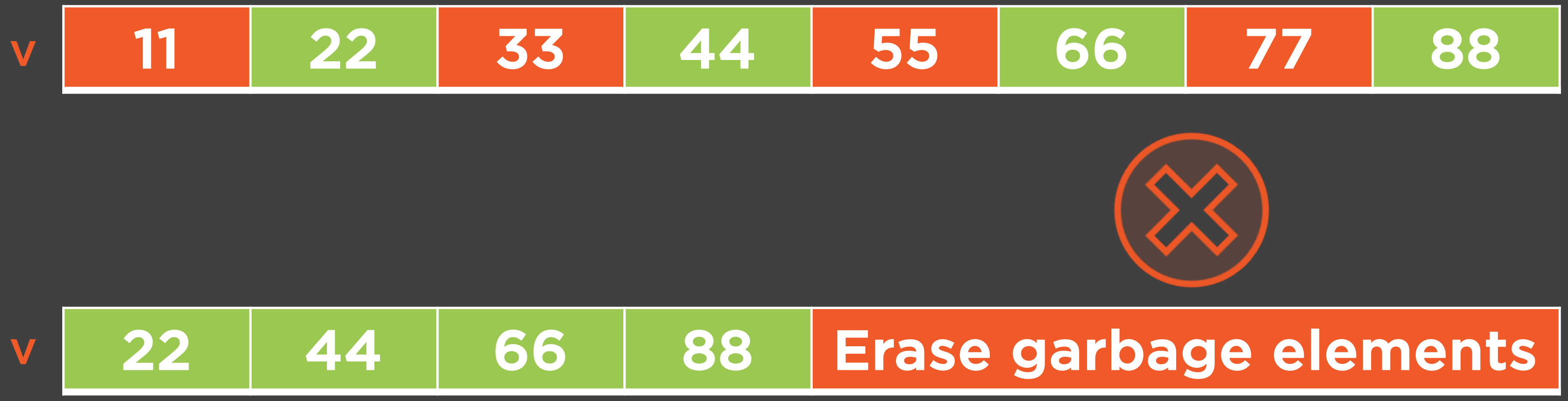
remove_if *shifts* the «good» elements at the beginning



```
std::remove_if(begin(v), end(v), IsOdd)
```

Removing Elements with std::remove_if

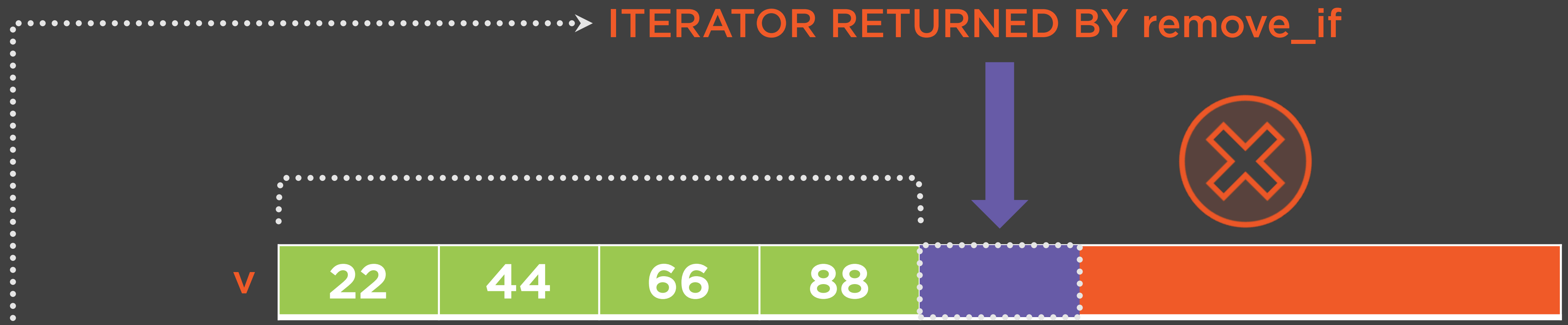
remove_if does *not* change the physical *size* of the container



```
std::remove_if(begin(v), end(v), IsOdd)
```

Removing Elements with std::remove_if

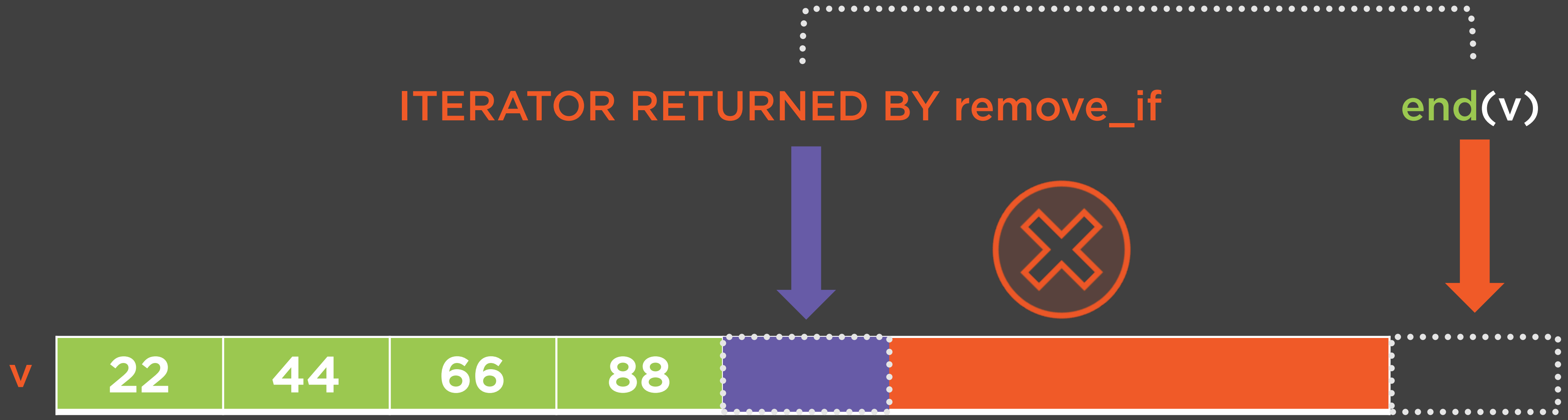
remove_if does *not* change the physical *size* of the container



```
std::remove_if(begin(v), end(v), IsOdd)
```

Removing Elements with std::remove_if

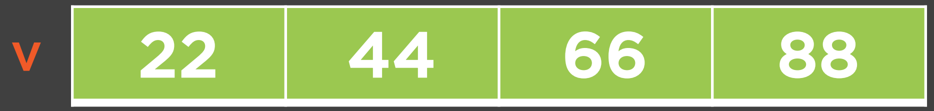
Use the iterator *returned* by remove_if as the new vector end



```
v.erase(/* Iterator returned by remove_if */, end(v));
```

Erase the Remaining Elements

Call std::vector::*erase*

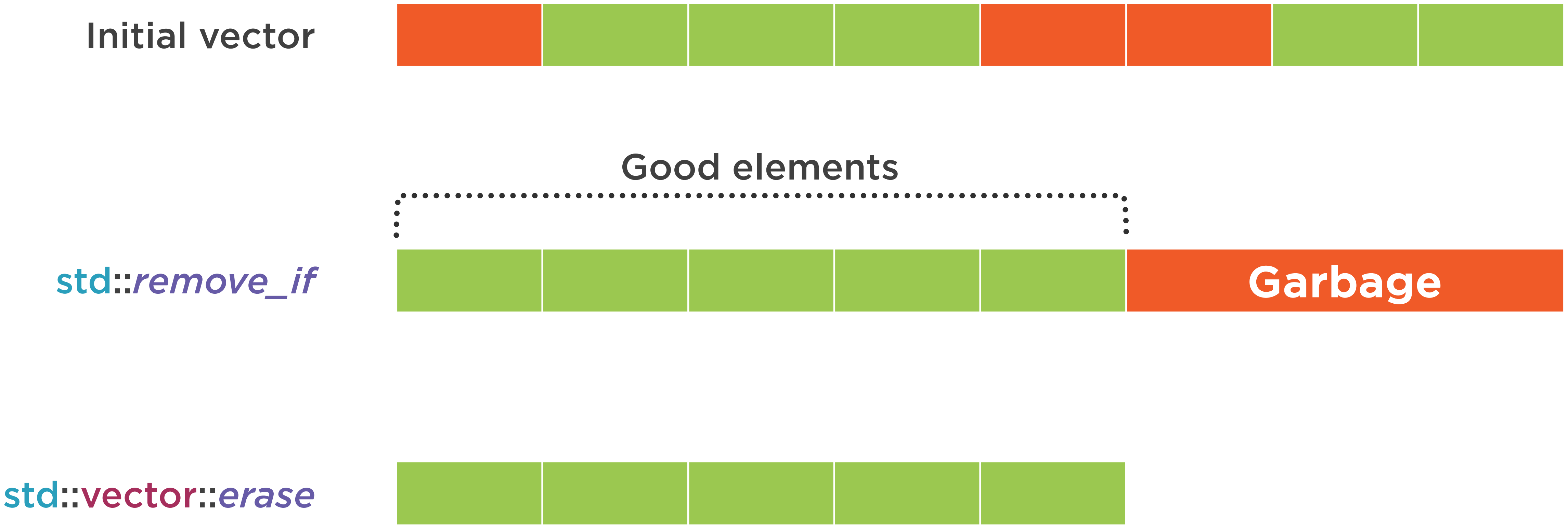


```
v.erase(/* Iterator returned by remove_if */, end(v));
```

Erase the Remaining Elements

After vector::*erase* call, only the good elements are left in the vector

Removing Elements from std::vector

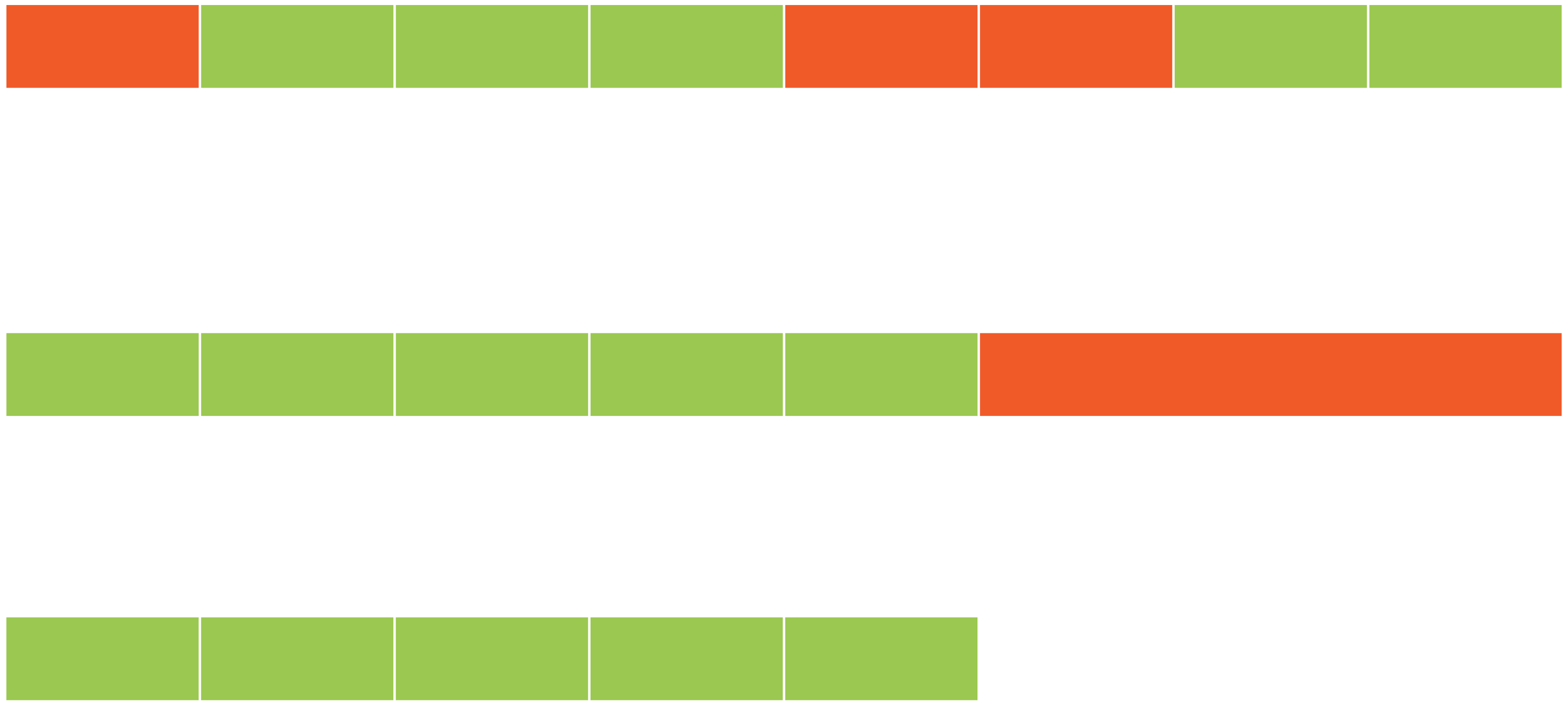


Removing Elements from std::vector

Erase-remove idiom

std::*remove_if*

std::vector::*erase*



Removing Elements: remove_if vs. remove

std::remove_if

All elements for which
a *predicate* is *true*

std::remove

All elements that are *equal*
to a given *value*

C++20

std::vector

```
void std::erase    (container, value     );
void std::erase_if (container, predicate);
```

Convenient Wrappers for Erase-remove Idiom

std::*remove* / *remove_if*

std::vector::*erase*

```
vector<string> v{
  "Galileo", "C64", "Connie", "Amiga", "C++"
};
```

Searching Elements

Searching Elements

Reuse Standard Library's Code

std::find

```
it = std::find(first, last, value);
```

Searching with std::find

SEARCH RANGE

```
it = std::find(first, last, value);
```

Searching with std::find

SEARCH RANGE

begin(v), *end*(v)

```
it = std::find(first, last, value);
```

Searching with std::find

SEARCH VALUE

```
it = std::find(first, last, value);
```

Searching with std::find

ITERATOR TO MATCHING ELEMENT

```
it = std::find(first, last, value);
```

Searching with std::find

‘last’ IF ELEMENT NOT FOUND

```
it = std::find(first, last, value);
```

Searching with std::find

Searching for a String in a vector with std::find

```
// vector<string> v{ … };

auto result = std::find(begin(v), end(v), "Connie");
```

Searching for a String in a vector with std::find

```
// vector<string> v{ … };
auto result = std::find(begin(v), end(v), "Connie");

if (result != end(v)) {
   // "Connie" is in the vector
} else {
   // Not found
}
```

```
it = std::find(first, last, value);

it = std::find_if(first, last, predicate);
```

Search for elements
for which *predicate* is *true*

Conditional Search with std::find_if

```cpp
vector<string> v{
  "Galileo", "C64", "Connie", "Amiga", "C++"
};


it = std::find(begin(v), end(v), "Connie");
```

Case-insensitive String Search

```cpp
vector<string> v{
  "Galileo", "C64", "Connie", "Amiga", "C++"
};

it = std::find(begin(v), end(v), "CONNIE");
```

Case-insensitive String Search

```cpp
vector<string> v{
   "Galileo", "C64", "Connie", "Amiga", "C++"
};

it = find_if(begin(v), end(v), CaseInsensitiveCompare);
```

"Connie"
"connie"
"COnniE"
"CONNIE"
...

Case-insensitive String Search

Summary

Inserting elements with vector::*insert*

Removing elements (*erase-remove*)

Searching with std::*find* and std::*find_if*